Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка веб-платформи «TutorSpace» для навчання онлайн
(комплексна робота)

Виконали: студенти IV курсу, групи СН-42
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Ношкалюк А.С.
(підпис) (прізвище та ініціали)
_____ Пазуханич О.В.
(підпис) (прізвище та ініціали)

Керівник _____ Небесний Р.М.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Оробчук О.Р.
(підпис) (прізвище та ініціали)

Рецензент _____ Голотенко О.С.
(підпис) (прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«27» січня 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)
за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)
Студенту Ношкалюк Арсен Сергійович
(прізвище, ім'я, по батькові)
1. Тема роботи Розробка веб-платформи TutorSpace для навчання онлайн

Керівник роботи Небесний Руслан Михайлович, доктор філософії (PhD), доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від «*7*» *травня* 2025 року № *4/7-444*
2. Термін подання студентом завершеної роботи червня 2025р.
3. Вихідні дані до роботи базуються на основі досліджених літературних та інтернет-джерелах
4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1 Аналіз дистанційної освіти, веб-розробки та постановка завдання. 1.1 Сучасний стан розвитку навчальних платформ. 1.2 Історія розвитку СДО та навчальних платформ. 1.3 Історія та значення Django в контексті створення навчальних веб-платформ. 1.4 Використання React у фронтенд-розробці освітніх веб-платформ. 1.5 PostgreSQL як надійна основа збереження та обробки даних у навчальних веб-платформах. 1.6 Постановка мети і задач дослідження. 1.7 Висновок до першого розділу. 2. Процес проєктування архітектури для навчальної платформи «TutorSpace». 2.1 Аналіз конкурентних рішень як основа для проєктування функціоналу та інтерфейсу. 2.1.1 Платформа Open edX. 2.1.2 Платформа Moodle 2.1.3 Платформа Canvas LMS. 2.2 Визначення цільової аудиторії та потреб користувачів, створення портрету «ідеального користувача». 2.3 Аналіз, вибір та аргументація технологій Django, React та PostgreSQL. 2.4 Визначення ключових потреб веб-платформи та дослідження функціональних можливостей «TutorSpace». 2.5 Висновок до другого розділу. 3 Розробка функціоналу для навчальної платформи «TutorSpace». 3.1 Опис основних технологій та засобів для розробки інтерфейсу користувача. 3.2 Опис основних структур для розробки бізнес логіки платформи. 3.3 Демонстрація розробленої платформи. 3.4 Висновок до третього розділу. 4 Безпека життєдіяльності, основи охорони праці. 4.1 Інтеграція принципів ергономіки та охорони праці в роботу розробника.4.2 Психофізіологічні ризики при тривалій роботі за
5. Висновок

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Сенчишин В. С., кандидат технічних наук, доцент кафедри МТ		

7. Дата видачі завдання 27 січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	28.01.2025	*Виконано*
2.	Підбір та опрацювання літературних джерел по темі кваліфікаційної роботи	31.01.2025-03.02.2025	*Виконано*
3.	Опрацювання джерел по темі кваліфікаційної роботи	04.02.2025-06.02.2025	*Виконано*
4.	Виконання дослідження щодо бізнес логіки бекенду програмної архітектури у навчальних платформах	07.02.2025-09.02.2025	*Виконано*
5.	Розроблення структури та бекенд функціоналу веб-платформи “TutorSpace”	20.05.2025-01.06.2025	*Виконано*
6.	Оформлення розділу «Аналіз дистанційної освіти, веб-розробки та постановка завдання»	02.06.2025-05.06.2025	*Виконано*
7.	Оформлення розділу «Процес проєктування архітектури для навчальної платформи «TutorSpace»»	06.06.2025-08.06.2025	*Виконано*
8.	Оформлення розділу «Розробка функціоналу для навчальної платформи «TutorSpace»»	09.06.2025-11.06.2025	*Виконано*
9.	Виконання завдання до підрозділу «Безпека життєдіяльності»	12.06.2025-13.06.2025	*Виконано*
10.	Виконання завдання до підрозділу «Основи охорони праці»	14.06.2025-15.06.2025	*Виконано*
11.	Оформлення кваліфікаційної роботи	16.06.2025-17.06.2025	*Виконано*
12.	Нормоконтроль	23.06.2025-24.06.2025	*Виконано*
13.	Перевірка на плагіат	23.06.2025	*Виконано*
14.	Попередній захист кваліфікаційної роботи	23.06.2025	*Виконано*
15.	Захист кваліфікаційної роботи	27.06.2025	

Студент _____ Ношкалюк А.С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Небесний Р.М
(підпис) (прізвище та ініціали)

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«27» січня 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)
за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)
Студенту Пазуханичу Олександру Васильовичу
(прізвище, ім'я, по батькові)
1. Тема роботи Розробка веб-платформи TutorSpace для навчання онлайн

Керівник роботи Небесний Руслан Михайлович, доктор філософії (PhD), доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від «*7*» *травня* 2025 року № *4/7-444*
2. Термін подання студентом завершеної роботи червня 2025р.
3. Вихідні дані до роботи базуються на основі досліджених літературних та інтернет-джерелах
4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1 Аналіз дистанційної освіти, веб-розробки та постановка завдання. 1.1 Сучасний стан розвитку навчальних платформ. 1.2 Історія розвитку СДО та навчальних платформ. 1.3 Історія та значення Django в контексті створення навчальних веб-платформ. 1.4 Використання React у фронтенд-розробці освітніх веб-платформ. 1.5 PostgreSQL як надійна основа збереження та обробки даних у навчальних веб-платформах. 1.6 Постановка мети і задач дослідження. 1.7 Висновок до першого розділу. 2. Процес проєктування архітектури для навчальної платформи «TutorSpace». 2.1 Аналіз конкурентних рішень як основа для проєктування функціоналу та інтерфейсу. 2.1.1 Платформа Open edX. 2.1.2 Платформа Moodle 2.1.3 Платформа Canvas LMS. 2.2 Визначення цільової аудиторії та потреб користувачів, створення портрету «ідеального користувача». 2.3 Аналіз, вибір та аргументація технологій Django, React та PostgreSQL. 2.4 Визначення ключових потреб веб-платформи та дослідження функціональних можливостей «TutorSpace». 2.5 Висновок до другого розділу. 3 Розробка функціоналу для навчальної платформи «TutorSpace». 3.1 Опис основних технологій та засобів для розробки інтерфейсу користувача. 3.2 Опис основних структур для розробки бізнес логіки платформи. 3.3 Демонстрація розробленої платформи. 3.4 Висновок до третього розділу. 4 Безпека життєдіяльності, основи охорони праці. 4.1 Інтеграція принципів ергономіки та охорони праці в роботу розробника.4.2 Психофізіологічні ризики при тривалій роботі за
5. Висновок

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Сенчишин В. С., кандидат технічних наук, доцент кафедри МТ		

7. Дата видачі завдання 27 січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	28.01.2025	*Виконано*
2.	Підбір та опрацювання літературних джерел по темі кваліфікаційної роботи	02.02.2025-06.02.2025	*Виконано*
3.	Опрацювання джерел по темі кваліфікаційної роботи	07.02.2025-10.02.2025	*Виконано*
4.	Виконання дослідження щодо підбору технологій для початку розробки	11.02.2025-12.02.2025	*Виконано*
5.	Початок розробки на основі обраних технологій	16.05.2025-01.06.2025	*Виконано*
6.	Оформлення розділу «Процес проєктування архітектури для навчальної платформи «TutorSpace»»	07.06.2025-09.06.2025	*Виконано*
7.	Оформлення розділу «Розробка функціоналу для навчальної платформи «TutorSpace»»	09.06.2025-12.06.2025	*Виконано*
8.	Виконання завдання до підрозділу «Безпека життєдіяльності»	12.06.2025-13.06.2025	*Виконано*
9.	Виконання завдання до підрозділу «Основи охорони праці»	14.06.2025-15.06.2025	*Виконано*
10.	Оформлення кваліфікаційної роботи	16.06.2025-17.06.2025	*Виконано*
11.	Нормоконтроль	23.06.2025-24.06.2025	*Виконано*
12.	Перевірка на плагіат	23.06.2025	*Виконано*
13.	Попередній захист кваліфікаційної роботи	23.06.2025	*Виконано*
14.	Захист кваліфікаційної роботи	27.06.2025	

Студент _____ Пазуханич О.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Небесний Р.М
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка веб-платформи «TutorSpace» для навчання онлайн // Кваліфікаційна робота освітнього рівня «Бакалавр» // Ношкалюк Арсен Сергійович, Пазуханич Олександр Васильович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-42 // Тернопіль, 2025 // с. – 101, рис. – 17, табл. – 3, слайди. – 0, додат. – 2, бібліогр. – 84.

Ключові слова: Веб-платформа, Онлайн-навчання, Django, React, PostgreSQL, Архітектура, Frontend, Backend, Адаптивність, REST API, Система управління навчанням (LMS).

Кваліфікаційна робота присвячена розробці веб-платформи для онлайн-навчання. У першому розділі розглянуто актуальність дистанційної освіти та проаналізовано існуючі рішення у цій сфері. Досліджено особливості сучасних навчальних платформ, їх функціональність, переваги та недоліки. Проведено аналіз архітектурних підходів до побудови систем онлайн-навчання.

У другому розділі описано процес проєктування системи. Наведено загальну архітектуру платформи, обґрунтовано вибір основних технологій, зокрема Django, React та PostgreSQL. Розглянуто побудову взаємодії між фронтендом і бекендом, а також реалізацію REST API.

У третьому розділі наведено етапи реалізації функціоналу платформи: реєстрація та авторизація користувачів, створення та управління курсами, призначення викладачів, проходження сертифікації, а також організація навчального процесу. Окрему увагу приділено забезпеченню безпеки, адаптивності інтерфейсу та зручності для кінцевого користувача.

Об'єкт дослідження кваліфікаційної роботи – веб-платформа для онлайн-навчання. Предмет дослідження – процес розробки архітектури, функціоналу та взаємодії компонентів системи онлайн-навчання.

ANNOTATION

Development of the “TutorSpace” Web Platform for Online Learning // Qualification work of the educational level "Bachelor" // Noshkaliuk Arseniy Serhiyovych, Pazukhanich Oleksandr Vasylovych // Ternopil Ivan Pulyu National Technical University, Computer and Information Systems and Software Engineering Faculty, Computer Sciences Department, Group SN-42 // Ternopil, 2025 // pp. – 101, figures – 17, tables – 3, slides – 0, appendices – 2, bibliography – 84.

Keywords: Web platform, Online learning, Django, React, PostgreSQL, Architecture, Frontend, Backend, Responsiveness, REST API, Learning Management System (LMS).

This qualification thesis is dedicated to the development of a web platform for online learning. The first chapter examines the relevance of distance education and analyzes existing solutions in this field. The features of modern learning platforms, their functionality, advantages, and disadvantages are studied. Architectural approaches to building online learning systems are analyzed.

The second chapter describes the system design process. The overall architecture of the platform is presented, and the choice of core technologies is justified, including Django, React, and PostgreSQL. The interaction between the frontend and backend is considered, along with the implementation of the REST API.

The third chapter outlines the stages of implementing the platform’s functionality: user registration and authentication, course creation and management, teacher assignment, certification process, and the organization of the learning workflow. Special attention is given to ensuring security, interface responsiveness, and user-friendliness. Object of the research: a web platform for online learning. Subject of the research: the process of developing the architecture, functionality, and interaction of components in an online learning system.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

LMS *(Learning Management System)* – система управління навчанням; платформа для організації, контролю та реалізації освітніх процесів онлайн.

REST API *(Representational State Transfer Application Programming Interface)* – архітектурний стиль програмних інтерфейсів для взаємодії клієнта та сервера через HTTP.

HTML *(HyperText Markup Language)* – мова розмітки гіпертексту, що використовується для створення структури вебсторінок.

CSS *(Cascading Style Sheets)* – каскадні таблиці стилів; мова для оформлення зовнішнього вигляду вебсторінок.

JS *(JavaScript)* – мова програмування для реалізації інтерактивності на вебсторінках.

SPA *(Single Page Application)* – односторінковий вебзастосунок, який завантажує єдиний HTML-файл і динамічно оновлює вміст без перезавантаження сторінки.

Django – високорівневий фреймворк для розробки вебзастосунків мовою Python.

React – розроблена Facebook JavaScript-бібліотека для створення користувацького інтерфейсу.

PostgreSQL – об'єктно-реляційна система керування базами даних із відкритим кодом.

JWT *(JSON Web Token)* – відкритий стандарт безпечного обміну інформацією між клієнтом і сервером у форматі JSON.

API *(Application Programming Interface)* – набір правил, який дозволяє взаємодію між різними програмними компонентами.

ЗМІСТ

ВСТУП

Актуальність теми. Дивлячись на стрімкий розвиток технологій та зростання потреби в дистанційному навчанні, постає потреба у створенні сучасних, технічно досконалих освітніх платформ. Більшість існуючих рішень мають певні обмеження – як у зручності користування, так і в гнучкості функціоналу чи масштабованості. Крім того, заклади освіти часто змушені комбінувати кілька сервісів одночасно: для відео-зв'язку, управління курсами, тестування та спілкування. У цьому контексті особливо актуальною є розробка єдиної платформи з чітко побудованою архітектурою, яка об’єднує всі необхідні інструменти. Використання стеку Django, React, PostgreSQL дозволяє створити масштабовану, безпечну та продуктивну систему, що забезпечує стабільну роботу як для студентів, так і для викладачів. Внаслідок цього, розробка веб-платформи TutorSpace для навчання онлайн є актуальним завданням у сфері освітніх технологій та програмування.

Мета і задачі дослідження. Ціль цієї кваліфікаційної роботи освітнього рівня «Бакалавр» є розробка веб-платформи TutorSpace для навчання онлайн, яка є повнофункціональною, технічно надійною та поєднує сучасні підходи до веб-розробки, гнучку архітектуру та зручний функціонал для користувачів різних ролей. Основна мета полягає у реалізації бекенд та фронтенд частин платформи з використанням сучасного стеку технологій Django, React та PostgreSQL.

Щоб досягти визначеної мети, слід виконати такі завдання:

– дослідити сучасні технічні рішення в галузі дистанційного навчання з погляду архітектурного підходу та функціоналу;

– визначити вимоги до функціональності платформи з урахуванням потреб студентів, викладачів та адміністраторів;

– побудувати архітектуру застосунку;

– розробити REST API для обміну даними між клієнтом і сервером;

– реалізувати бекенд-платформу на основі Django;

– створити фронтенд-інтерфейс на React з адаптивним дизайном;

– забезпечити підключення до бази даних PostgreSQL, організувати структуру моделей та зв’язки між ними;

– реалізувати функціональні можливості;

– провести тестування основних модулів системи;

– підготувати розгортання платформи на сервері та забезпечити її доступність у веб-середовищі.

Практичне значення одержаних результатів. Підсумки цієї кваліфікаційної роботи мають значну практичну цінність у сфері цифрових освітніх технологій. Реалізована веб-платформа «TutorSpace» може бути використана як повноцінне рішення для організації дистанційного навчання в освітніх установах різного рівня, а також як основа для подальшого розширення та впровадження нових функцій. Розроблена архітектура системи забезпечить гнучкість, масштабованість і безпеку, що дозволить легко адаптувати її під потреби користувачів та вийти на світову арену.

Частина зі сторони фронтенду реалізована з урахуванням принципів адаптивності, продуктивності та доступності, що гарантує комфортну роботу користувачам на різних пристроях. Бекенд система надає стабільну інфраструктуру для обробки запитів, управління даними, автентифікації користувачів та інтеграції сторонніх сервісів.

Результати дослідження можуть бути застосовані розробниками як для створення нових освітніх рішень, так і для подальшого вдосконалення існуючих систем онлайн-навчання. Впровадження подібної платформи може сприяти формуванню цілісного, функціонального та якісного освітнього середовища. Запропоновані технічні підходи сприяють підвищенню надійності, ефективності та гнучкості веб-платформ, орієнтованих на сучасного користувача.

Розроблена веб-платформа «TutorSpace» може стати основою для створення масштабованого, повнофункціонального програмного рішення з підтримкою інтеграції з додатковими сервісами, такими як календарі подій, системи електронних платежів, відео-зв’язку, чат-боти, системи оповіщення та інші інструменти. Такі принципи дозволять користувачам зосередитися

безпосередньо на навчальному процесі, мінімізувати технічні труднощі та покращити загальне враження від взаємодії із системою.

Таким чином, реалізація платформи «TutorSpace» не лише забезпечує технічну основу для проведення онлайн-навчання, а й робить внесок у цифрову трансформацію освітнього простору, надаючи всі необхідні інструменти для комфортної, логічно організованої та ефективної взаємодії між усіма учасниками навчального процесу.

Авторський внесок Ношкалюка Арсена Сергійовича: Вступ, Розділ 1 Аналіз дистанційної освіти, веб-розробки та постановка завдання, п. 1.1 Сучасний стан розвитку навчальних платформ, п. 1.2 Історія розвитку СДО та навчальних платформ, п. 1.3 Історія та значення Django в контексті створення навчальних веб-платформ, п. 1.5 PostgreSQL як надійна основа збереження та обробки даних у навчальних веб-платформах, п. 1.7 Висновок до першого розділу, Розділ 2 Процес проєктування архітектури для навчальної платформи «TutorSpace», п. 2.1 Аналіз конкурентних рішень як основа для проєктування функціоналу та інтерфейсу, п. 2.2 Визначення цільової аудиторії та потреб користувачів, створення портрету «ідеального користувача», п. 2.3 Аналіз, вибір та аргументація технологій Django, React та PostgreSQL, п. 2.4 Визначення ключових потреб веб-платформи та дослідження функціональних можливостей «TutorSpace», п. 2.5 Висновок до другого розділу, Розділ 3 Розробка функціоналу для навчальної платформи «TutorSpace», п. 3.2 Опис основних структур для розробки бізнес логіки платформи, Розділ 4 Безпека життєдіяльності, основи охорони праці, п. 4.1 Інтеграція принципів ергономіки та охорони праці в роботу розробника, п. 4.2 Психофізіологічні ризики при тривалій роботі за комп’ютером та шляхи, п. 4.5 Висновок до четвертого розділу, Перелік джерел.

Авторський внесок Пазуханич Олександра Васильовича: Розділ 1 Аналіз дистанційної освіти, веб-розробки та постановка завдання, п. 1.4 Використання React у фронтенд-розробці освітніх веб-платформ, п. 1.6 Постановка мети і задач дослідження, Розділ 2 Процес проєктування архітектури для навчальної платформи «TutorSpace», п. 2.3 Аналіз, вибір та аргументація технологій Django, React та PostgreSQL, Розділ 3 Розробка функціоналу для навчальної платформи

«TutorSpace», п. 3.1 Опис основних технологій та засобів для розробки інтерфейсу користувача, п. 3.3 Демонстрація розробленої платформи, п. 3.4 Висновок до третього розділу, Розділ 4 Безпека життєдіяльності, основи охорони праці, п. 4.3 Хто несе відповідальність за дотримання правил охорони праці під час роботи над проєктом у команді, п. 4.4 Які дії необхідно виконати у разі виникнення аварійної ситуації (наприклад, задимлення або короткого замикання) у приміщенні, де працює команда розробників, Висновки.

РОЗДІЛ 1. АНАЛІЗ ДИСТАНЦІЙНОЇ ОСВІТИ, ВЕБ-РОЗРОБКИ ТА ПОСТАНОВКА ЗАВДАННЯ

У цьому розділі розглянуто етапи становлення та сучасний стан розвитку освітніх веб-платформ у контексті дистанційного навчання. Проаналізовано наукові джерела, визначено ключові тенденції у використанні веб технологій у освітній сфері. Увагу зосереджено на архітектурних принципах, що використовуються при розробці сучасних платформ.

Крім того, у межах дослідження серед студентів було проведено опитування щодо найбільш зручних та бажаних функцій у системах онлайн-навчання, сформовано перелік вимог до функціональності платформи, проаналізовано найбільш популярні веб-технології, що використовуються для створення інтерфейсів користувача та серверної логіки у сфері EdTech. Розглянуто переваги використання стеку Django, React, PostgreSQL для створення масштабованої, безпечної та зручної платформи.

1.1 Сучасний стан розвитку навчальних платформ

Дистанційна освіта в Україні має більш ніж двадцятирічну історію. Її започаткування відбулося ще у 2002 році, коли Міністерство освіти і науки України розпочало експериментальне впровадження цієї форми навчання. Правову основу для її подальшого розвитку створили низка нормативних актів, таких як «Національна доктрина розвитку освіти», «Концепція розвитку дистанційної освіти в Україні», а також закони України «Про освіту», «Про вищу освіту», тощо [1].

Пандемія COVID-19, офіційно визнана Всесвітньою організацією охорони здоров’я у березні 2020 року, стала каталізатором різкого зростання попиту на онлайн-навчання у всьому світі [2]. В Україні цей процес посилився внаслідок повномасштабної військової агресії у 2022 році, яка спричинила значні втрати інфраструктури, зокрема навчальної, а також переміщення великої кількості

учнів, студентів та викладачів [3]. За цих умов було життєво необхідним створення рішень, що дозволяють забезпечити безперервність організації освіти без прив’язки до фізичного місцезнаходження учасників.

Саме тому системи дистанційного освіти (СДО) почали активно впроваджуватись як вищими навчальними закладами, так і приватними освітніми установами. На противагу класичному підходу, де викладач і студент перебувають в одному приміщенні, сучасні СДО базуються на веб-платформах із багаторівневою архітектурою. Їх реалізація вимагає продуманої технічної структури, яка поєднує серверну частину (бекенд), інтерфейс взаємодії з користувачем (фронтенд), а також надійну систему збереження даних. На рисунку 1.1 зображено структуру взаємодії користувача із системою.

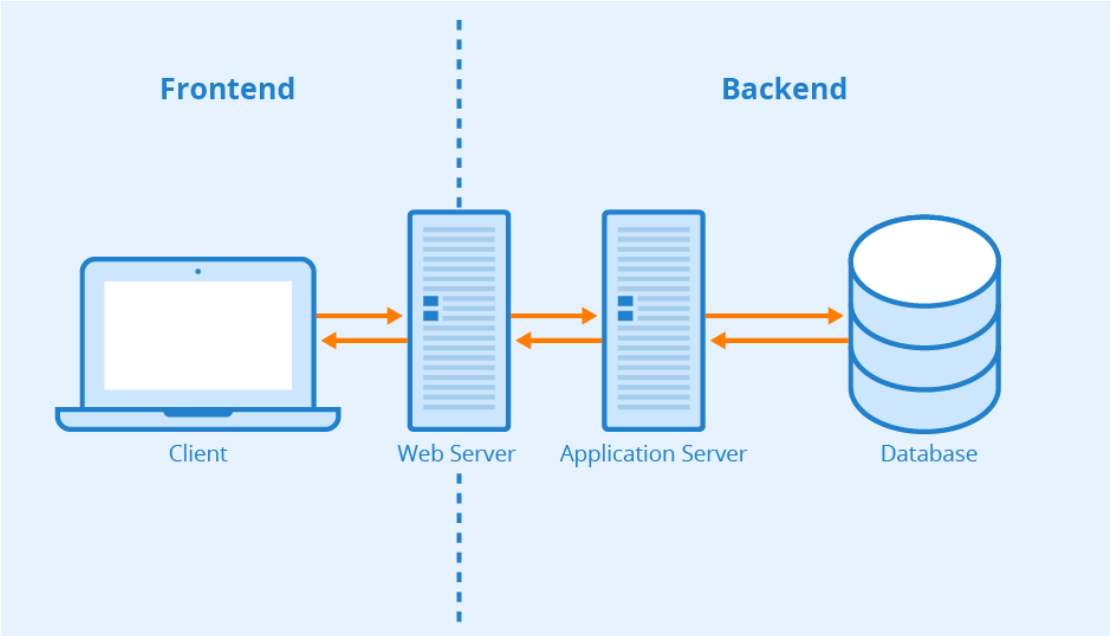

Рисунок 1.1 – Ілюстрація структури взаємодії користувача з системою.

Інформатизація освіти, як наголошують дослідники, є не лише процесом цифровізації існуючих матеріалів, але й глибокою трансформацією способів координації освітньої діяльності. Згідно з В. Ю. Биковим, інформатизація вимагає гнучких та адаптивних технічних рішень, які дозволяють відтворювати офлайн-формат у цифровому середовищі та створювати нові моделі навчання, орієнтовані на інтерактивність, персоналізацію та автономність [4].

У «Концепції розвитку дистанційної освіти в Україні» наголошується, що дистанційна освіта є рівноцінною традиційній і базується на використанні інформаційно-комунікаційних технологій для забезпечення підключення до навчальних матеріалів та спілкування між учасниками навчального процесу, а також підтримки високого рівня самостійної роботи [5]. Це передбачає створення стабільних і масштабованих технічних рішень, зокрема веб-платформ із модульною структурою, REST API, системою ролей користувачів, інтеграцією відео-конференцій, тестуванням, генерацією сертифікатів, тощо.

На думку В. Бикова [6], дистанційне навчання – це інструмент реалізації концепції відкритої освіти, який у значній мірі спирається на механізми інформатизації та технології керування навчальним процесом. У країнах Європи, наприклад, у Великій Британії, дистанційне навчання тривалий час сприймається як окрема педагогічна система, заснована на автономності студента у плануванні свого навчання [7].

Аналіз досвіду науковців та практиків призвів до висновку, що якість дистанційного навчання значною мірою залежить від реалізації функціональних можливостей платформи: зручної авторизації, стабільної роботи серверної логіки, надійного зберігання даних, підтримки інтерактивних форматів взаємодії, тощо. Саме ці аспекти потребують глибокого технічного опрацювання та відповідального підходу до проєктування архітектури веб-застосунку.

Сучасна освітня система поступово адаптується до викликів часу шляхом комбінування офлайн і онлайн форм навчання. Результати анкетування учасників освітнього процесу, зокрема у Тернопільському національному технічному університеті ім. Івана Пулюя, підтвердили ефективність такого підходу. На рисунку 1.2 зображено вищий навчальний заклад у якому проводилося опитування.

Рисунок 1.2 – Тернопільський національний технічний університет ім. Івана Пулюя

Платформи для онлайн-освіти отримали позитивні відгуки за можливості гнучко організовувати навчальний процес, поєднувати його з іншими активностями (зокрема працею), а також за підвищення рівня залученості та самодисципліни студентів [8].

Впровадження цифрових технологій у сфері освіти не лише змінило формат подачі матеріалу, але й істотно вплинуло на розвиток ключових навичок у здобувачів знань. Йдеться про підвищення рівня цифрової грамотності, уміння ефективно обробляти великий обсяг даних, ухвалювати самостійні рішення та користуватися різними формами комунікації в Інтернет-середовищі. Ці переваги було досягнуто завдяки впровадженню інтерактивного навчання, автоматизованих систем тестування, цифрових журналів та систем контролю прогресу [9].

Суттєвою перевагою для розробників сучасних освітніх платформ стало впровадження синхронного та асинхронного режимів взаємодії. Відповідно до чинного «Положення про дистанційне навчання» [10], перший режим забезпечує навчання онлайн (наприклад, через відео-конференції), у той час як другий дає студентам можливість навчатися у будь-який зручний для них час. Комбінація

цих підходів вимагає від платформи підтримки як потокової передачі даних, так і асинхронного обміну – зберігання лекцій, чатів, форумів, домашніх завдань.

Недостатня уніфікованість підходів до впровадження СДО, відсутність спільного державного стандарту для цифрових рішень у цій сфері, а також різний рівень підготовки фахівців – це додаткові труднощі, які ускладнюють ефективну розробку та впровадження освітніх веб-платформ [11].

Попри перешкоди, зацікавленість в онлайн-навчанні продовжує зростати [12]. Здобуття сертифікатів, які мають рівень офіційного підтвердження, гнучкість графіку навчання, легкий доступ до навчальних матеріалів, системи оцінювання та автоматизованого контролю за засвоєнням матеріалу – все це полегшує популяризацію онлайн-освіти [13]. Лише через технічно грамотне впровадження інноваційних платформ для керування навчальним процесом (LMS) можливо досягти високої якості цифрової освіти в умовах постійно зростаючих викликів.

1.2 Історія розвитку СДО та навчальних платформ

Еволюція дистанційної освіти та навчальних платформ відбувалася динамічно та стала логічною частиною розвитку цифрової освітньої інфраструктури. Поява перших відкритих освітніх ініціатив – таких як Open Course Ware (2002) та Open Learn (2006) започаткували новий підхід до поширення знань через Інтернет. Ці проєкти заклали основу для створення повноцінних веб-платформ з відкритим доступом до навчальних курсів [14].

Одним із ключових моментів став запуск платформи edX, розробленої Гарвардським університетом та Массачусетським технологічним інститутом. Водночас з’явилась платформа FutureLearn, орієнтована на професійний розвиток і широке охоплення аудиторії. Обидва рішення стали джерелом натхнення для десятків університетів у всьому світі, які почали створювати власні цифрові освітні ресурси [15].

Концепція масових відкритих онлайн-курсів набула популярності у 2008 році з появою курсу «Connectivism and Connective Knowledge», організованого Дж. Сіменсом і С. Даунсом. Початково курс був спрямований на обмежену аудиторію, на нього зареєструвалося понад 2300 учасників, що засвідчило інтерес до нової цифрової форми навчання [16].

Подальший прорив відбувся у 2011 році, коли Стенфордський університет відкрив безкоштовний онлайн-курс, який зібрав понад 160 000 слухачів з усього світу [17]. Це стало доказом того, що масштабовані веб-платформи здатні забезпечити доступ до знань для масової аудиторії при збереженні високої якості навчального контенту.

Станом на сьогодні існує понад 200 навчальних платформ різного масштабу. Серед найбільш відомих можна виокремити: Moodle, edX, Coursera, Google Classroom, ILIAS, Sakai, Blackboard, Prometheus, Classtime, Prosvita, WebTutor, eLearning Server та інші [18]. Частина з них – це системи з відкритим програмним кодом, які дають можливість розробникам і освітнім установам модифікувати функціональність відповідно до власних потреб [19], наприклад:

– Atutor активно використовується в ТНТУ;

– ILIAS застосовується в КНУ ім. Тараса Шевченка та інших вишах;

– Moodle – одна з найпоширеніших платформ, що використовується в освітніх установах у всьому світі.

Особливість платформ із відкритим кодом полягає у можливості кастомізації, інтеграції сторонніх сервісів (електронна пошта, платіжні системи, API-аналітика, відео-конференції), а також гнучкому масштабуванні під різні сценарії – від невеликих курсів до багатотисячних навчальних програм.

Крім самих платформ, активно розвивались і допоміжні веб-сервіси, що включають системи управління навчальним контентом, сервіси аналітики результатів, автоматичного оцінювання, електронні щоденники, адаптивні тести, тощо. Завдяки цьому було забезпечено не тільки комфорт студентів і викладачів, а й високий рівень автоматизації навчального процесу.

Таким чином, історія розвитку дистанційної освіти є водночас історією еволюції технічних платформ. Від простих веб-сайтів із лекційними матеріалами – до повноцінних систем із багаторівневими ролями користувачів, API-інтеграціями, адаптивною верстkою, підтримкою мобільних пристроїв та елементами штучного інтелекту. Ці зміни стали можливі завдяки поєднанню технічного прогресу, відкритості знань та потреби у гнучкому, доступному навчанні в умовах глобальних викликів.

1.3 Історія та значення Django в контексті створення навчальних веб-платформ

З розвитком цифрової освіти та поширенням дистанційного навчання виникла необхідність у надійних, масштабованих і функціональних серверних рішеннях. Одним із найпоширеніших інструментів для реалізації бекенд-частини веб-платформ є Django – високорівневий фреймворк для мови Python, що сприяє швидкому створенню веб-додатків з чіткою архітектурою, багатофункціональним API та захищеною логікою обробки даних.

Django було створено у 2003-2005 роках розробниками газети Lawrence Journal-World (Канзас, США), які прагнули спростити процес створення складних веб-сайтів, наповнених динамічним контентом. Перший публічний реліз фреймворку відбувся у 2005 році. Django швидко набув популярності завдяки концепції "DRY" – Don’t Repeat Yourself та філософії "Batteries included" – тобто вбудованої підтримки більшості необхідних компонентів для розробки: маршрутизації, Object-Relational Mapping (ORM), панелі адміністратора, форм, кешування, систем аутентифікації, захисту від атак CSRF/XSS, тощо [20].

Цей фреймворк орієнтований на розробку потужного серверного застосунку, який керує даними, обробляє запити та генерує відповіді на них через API або HTML-шаблони. Основні технічні переваги фреймворку:

– ORM – передбачає роботу з базою даних через класи, написані на Python без необхідності писати сирий SQL;

– панель адміністратора – автоматично згенерована адмін-панель для управління моделями;

– гнучка маршрутизація (URLs & views) – чітке розділення запитів та логіки їх обробки;

– модульність – підтримка застосунків дає змогу розбити платформу на незалежні частини: користувачі, курси, сертифікати, тощо;

– безпека – вбудовані механізми захисту від популярних атак (SQL injection, CSRF, clickjacking);

– підтримка REST API через Django REST Framework (DRF) – ключова складова при створенні Single Page Applications (SPA) з фронтендом на React [20].

У проектах типу LMS Django відіграє роль центрального серверного компонента, відповідального за:

– автентифікацію та авторизацію користувачів;

– збереження та видачу навчальних матеріалів;

– керування курсами, темами, тестами, результатами;

– взаємодію з фронтендом через API;

– логування дій та збір аналітики;

– генерацію сертифікатів та PDF-документів.

Оскільки освітні платформи повинні працювати з великою кількістю одночасних користувачів, Django добре себе проявляє в умовах високого навантаження завдяки сумісності з такими рішеннями як Redis, Celery, PostgreSQL, Docker, NGINX, Gunicorn [22].

Серед освітніх продуктів, які використовують Django – Open edX, Oppia, Canvas LMS, Moodle plugins (деякі через API) [23]. Це підкреслює придатність Django не лише для мінімально життєздатного продукту, але й для комерційних, масштабованих рішень.

Django став фундаментальним фреймворком для розробки сучасних, безпечних і функціональних освітніх платформ [24]. Його використання дає можливість досягти високого рівня продуктивності, захищеності та гнучкості у реалізації логіки навчального процесу [25]. Стабільна робота, багатий еко-системний стек та активна спільнота розробників забезпечують швидку розробку платформи та подальшу підтримку її розвитку [26].

1.4 Використання React у фронтенд-розробці освітніх веб-платформ

Зі зростанням вимог до інтерактивності, зручності та адаптивності веб-інтерфейсів, особливо у сфері онлайн-освіти, ключовою технологією у фронтенд-розробці стала бібліотека React. Її застосування дозволяє створювати гнучкі, масштабовані інтерфейси, які швидко реагують на дії користувача, не перевантажуючи сервер і не потребуючи повного перезавантаження сторінки.

React було створено інженерами компанії Facebook у 2011 році. Публічний реліз відбувся у 2013 році, і вже через кілька років бібліотека стала одним із найпопулярніших рішень у сфері фронтенду. Основною метою React було спростити розробку складних, інтерактивних інтерфейсів шляхом використання компонентної архітектури та віртуального Document Object Model (DOM) [27].

На відміну від традиційного підходу до побудови HTML-сторінок, React оперує декларативною моделлю – розробник описує, як має виглядати інтерфейс у певному стані, а бібліотека сама оновлює DOM при зміні даних.

У контексті розробки навчальних платформ, React забезпечує:

– компонентність – побудову інтерфейсу з незалежних елементів: кнопки, форми, модальні вікна, списки курсів, карточки матеріалів, тощо;

– JSX-синтаксис – поєднання HTML-подібної розмітки з логікою безпосередньо в коді компонентів;

– віртуальний DOM – пришвидшення оновлень сторінки завдяки зведення до мінімуму ручних операцій з реальним DOM;

– однобічну передачу даних (one-way data flow) – спрощення контролю станів;

– стан компонентів і хуки – використання useState, useEffect, useContext, тощо для обробки логіки поведінки інтерфейсу;

– підтримку роутінгу (React Router) – побудову багатосторінкових SPA без перезавантаження браузера;

– інтеграцію з Redux, Zustand, Context API – централізоване управління станом у великих додатках [28].

У системах онлайн-освіти інтерфейс відіграє критичну роль – саме через нього студент взаємодіє з платформою: читає лекції, виконує завдання, переглядає прогрес, проходить тести [29].

Використання React дозволяє значно зменшити навантаження на сервер, оскільки більша частина логіки (відображення, навігація, валідація) виконується на клієнтській стороні [30]. Таке рішення є критичним при одночасній роботі великої кількості користувачів (наприклад, під час масового тестування чи перегляду відео-лекцій) [31].

React працює в парі з Django через REST API. У цьому підході фронтенд є незалежною клієнтською частиною, яка:

– надсилає HTTP-запити до Django (POST, GET, PUT, DELETE);

– отримує дані у форматі JSON;

– обробляє відповіді для динамічного оновлення інтерфейсу;

– може використовувати токен-автентифікацію (JWT, Session, OAuth2) для захищеного доступу до персональних даних.

Такий розподіл дозволяє розвивати фронтенд і бекенд незалежно, а також легко створювати мобільні застосунки, які використовують ті самі API [32]. Серед освітніх рішень, де використовується React відомі такі як Khan Academy – платформа з відкритим кодом для інтерактивного навчання, Coursera – масові онлайн-курси, FreeCodeCamp – навчальна платформа для програмістів, EdX Studio – редактор курсів із компонентною архітектурою.

Усі ці системи мають складну логіку інтерфейсу, яка вимагає оперативного підходу до побудови UI, що й забезпечує React. Він став ключовим технологічним рішенням у реалізації сучасних освітніх веб-платформ завдяки своїй швидкодії, гнучкості та можливості створювати масштабовані інтерактивні інтерфейси [33].

1.5 PostgreSQL як надійна основа збереження та обробки даних у навчальних веб-платформах

У сучасних веб-застосунках, зокрема платформах дистанційної освіти, інструмент керування базами даних відіграє критичну роль [34]. Вона відповідає за зберігання, цілісність, швидкий доступ та опрацювання великих обсягів інформації, серед яких навчальні матеріали, тести, відповіді, результати, журнали активності та інше [35].

PostgreSQL бере свій початок ще з 1986 року в Каліфорнійському університеті в Берклі, де як частина дослідницького проєкту був розроблений Postgres [36]. У 1996 році система отримала підтримку SQL і була перейменована в PostgreSQL. Сьогодні вона є однією з найбільш популярних і надійних СУБД з відкритим кодом [37].

PostgreSQL – це надійна, безпечна і функціонально багата СУБД, яка є ідеальним вибором для сучасних навчальних платформ [38]. У поєднанні з Django та React вона дозволяє створити повноцінну систему, яка здатна обслуговувати тисячі користувачів, обробляти великі обсяги даних та адаптуватися до потреб навчального процесу [39].

PostgreSQL активно розвивається, підтримується великою міжнародною спільнотою, і використовується як у стартапах, так і у великих компаніях (Apple, Instagram, Spotify, Red Hat тощо).

PostgreSQL має ряд переваг, які роблять її ідеальним вибором для веб-платформ:

– підтримка ACID-транзакцій – гарантує цілісність даних (особливо важливо при збереженні тестів, оцінок, сертифікатів);

– масштабованість – справляється з великими навантаженнями;

– потужний SQL та розширення – дозволяє створювати складні запити, використовувати розширення (PostGIS, pg_trgm, unaccent тощо);

– JSONB – підтримка збереження напівструктурованих даних, які зручно використовувати разом із React;

– індекси та тригери – покращують продуктивність при пошуку й зміні даних;

– реплікація та резервне копіювання – забезпечення безпеки та відмовостійкості;

– безпека – контроль доступу до таблиць, стовпців, функцій та рядків на рівні політик.

У контексті Django PostgreSQL має повну сумісність з ORM, а також підтримує додаткові специфічні функції, такі як повнотекстовий пошук, унікальні типи даних, що дозволяють зберігати масиви або словники без створення додаткових таблиць, ефективну аналітику даних через агрегування, зручну міграцію схеми бази даних [40].

Це дозволяє швидко розгортати, оновлювати та масштабувати освітню платформу. Одні з популярних систем управління навчанням, такі як Open edX, Moodle, Canvas LMS використовують PostgreSQL для підтримки бази даних.

У сучасних LMS системах PostgreSQL застосовується для обробки:

– даних користувачів (викладачі, студенти, адміністратори);

– структури курсів, розкладів, модулів;

– тестів, запитань, відповідей, балів;

– записів активності та результатів проходження;

– логів дій, які важливі для адміністративного аудиту.

Таким чином, ця СУБД вже довела свою ефективність у реальних проектах.

Архітектурно PostgreSQL дозволяє реалізувати гнучку модель даних, орієнтовану на поділ користувачів за ролями, зберігання історії навчання, зв’язки між курсами, сертифікацію, тощо.

1.6 Постановка мети і задач дослідження

Метою цієї кваліфікаційної роботи на здобуття ступеня бакалавра є розробка веб-платформи TutorSpace для навчання онлайн, проведення дослідження, проєктування та розробка функціонального, масштабованого та технічно надійного середовища з використанням сучасних технологій веб-розробки: Django, React та PostgreSQL.

Платформа повинна забезпечити зручний інтерфейс для користувачів (студентів, викладачів, адміністраторів), підтримку адаптивного дизайну, організацію курсів, тестування, систему сертифікації, аналітику успішності та механізми безпечної аутентифікації.

З метою досягнення поставленої мети визначено такі завдання:

1. провести аналіз науково-технічної літератури та сучасних веб-технологій, що використовуються у створенні освітніх платформ;
2. дослідити архітектурні підходи до проєктування навчальних веб-додатків (моноліт, REST, SPA);
3. проаналізувати конкурентні рішення з технічного боку: стек технологій, структура API, база даних, масштабованість;
4. визначити технічні вимоги до функціоналу платформи TutorSpace;
5. обґрунтувати вибір основного стеку, середовища розробки, інструментів для розгортання (Docker, Git, CI/CD);
6. сформулювати структуру бази даних та зв’язки між сутностями;
7. розробити бекенд-платформу на основі Django із підтримкою REST API та токен-автентифікації;
8. реалізувати фронтенд-інтерфейс на React з підтримкою адаптивності та взаємодії з API;

9. налаштувати розгортання платформи в локальному та серверному середовищі;

10. провести тестування основних модулів;

11. підготувати документацію API та опис реалізованої структури проєкту.

Об’єктом дослідження цієї кваліфікаційної роботи спрямоване на створення освітньої веб-платформи, включаючи серверну логіку, структуру бази даних, клієнтську частину та взаємодію між ними.

Предметом дослідження виступає сукупність інструментів та технологій веб-розробки, які використовуються для побудови надійної, масштабованої та інтерактивної системи онлайн-навчання.

У межах дослідження аналізуються сучасні підходи до архітектури веб-застосунків, принципи побудови REST API, засоби інтеграції клієнтської частини з бекендом, а також особливості розробки баз даних для освітніх цілей. Окрему увагу приділено адаптивності платформи, зручності її використання на різних пристроях та безпеці обробки даних користувачів. У ході реалізації також проведено тестування працездатності окремих модулів, а розроблена структура коду дозволяє розширювати платформу в майбутньому.

1.7 Висновок до першого розділу

Перший розділ включає глибокий аналіз еволюційних змін дистанційної освіти та становлення навчальних веб-платформ, які відіграють ключову роль у трансформації освітнього процесу в умовах глобальних викликів. Відстежено основні етапи розвитку систем дистанційного навчання в Україні – від перших нормативно-правових актів на початку 2000-х років до сучасних технологічних рішень, що впроваджуються в умовах пандемії COVID-19 та повномасштабної війни.

Поглиблений аналіз присвячено актуальним технічним рішенням до побудови освітніх платформ, зокрема використання веб-технологій, що

дозволяють забезпечити масштабованість, доступність і ефективну інтерактивну взаємодію між користувачами. Визначено тенденції переходу від класичних веб-сайтів до повноцінних SPA-рішень, які побудовані на основі React та взаємодіють із REST API, реалізованими за допомогою Django. Розглянуто переваги використання PostgreSQL як стабільної, безпечної та масштабованої системи управління базами даних.

Зроблено висновок, що ефективність навчальних платформ великою мірою визначається не тільки педагогічними підходами, але й технічною архітектурою, яка забезпечує зручність використання, стабільну роботу системи та безпеку даних користувачів. Системи, які поєднують правильно спроєктовану серверну логіку, оптимізовану базу даних та інтерактивний інтерфейс, демонструють вищу продуктивність та залучення користувачів.

Можна стверджувати, що успішна реалізація цифрової платформи для онлайн-навчання потребує синергії сучасних технологій бекенду, фронтенду та обраного підходу до взаємодії з даними. Результати аналітичного огляду підтверджують актуальність дослідження та створюють міцну основу для подальшого розроблення архітектури та функціонального наповнення системи «TutorSpace».

РОЗДІЛ 2. ПРОЦЕС ПРОЄКТУВАННЯ АРХІТЕКТУРИ ДЛЯ НАВЧАЛЬНОЇ ПЛАТФОРМИ «TUTORSPACE»

У цьому розділі розглянуто етапи проєктування архітектури веб-платформи дистанційного навчання «TutorSpace» на основі сучасного технологічного стеку: Django, React та PostgreSQL. Проєктування охоплює опис функціональних і нефункціональних характеристик системи й аналіз альтернативних технічних рішень, побудову структури системи, вибір інструментів розробки та обґрунтування архітектурних рішень.

2.1 Аналіз конкурентних рішень як основа для проєктування функціоналу та інтерфейсу

Для створення ефективної навчальної веб-платформи необхідно детально вивчити вже існуючі рішення на ринку онлайн-освіти. Аналіз конкурентів дозволяє виявити переваги та слабкі сторони їхніх продуктів, дослідити сучасні підходи до реалізації функціоналу, інтерфейсів та структури даних, а також визначити ключові очікування користувачів. Таке дослідження слугує джерелом практичних рішень і надихає на створення нових підходів, адаптованих до конкретних цілей проєкту.

Особливу цінність становить вивчення технологій, які були використані для реалізації інтерфейс-частини, організації обміну даними з сервером, будови структури курсів і тестів, рівнів доступу для ролей користувачів, а також реалізації механізмів реєстрації, пошуку, аналітики та сертифікації.

Крім технічних рішень, аналіз охоплює й загальне враження користувача, зручну навігацію, логічно структурований контент, інформаційну ієрархію, адаптивність. Онлайн-джерела – офіційні сайти, публічні репозиторії, технічна документація, демо-версії, відгуки користувачів, форуми – виступають як основні ресурси для збору подібної інформації. Такий підхід дозволяє:

– зібрати приклади успішної реалізації функцій;

– вивчити UI-патерни та UX-поведінку;
– виявити візуальні елементи, які спрощують взаємодію;
– сформулювати вимоги до власного інтерфейсу та функціоналу.

У результаті систематичного аналізу, платформи можна класифікувати за типом контенту, рівнем доступу, орієнтацією на цільову аудиторію та типом сертифікації. Це дає змогу сформувати цілісне уявлення про ефективну архітектуру і користувацький досвід.

2.1.1 Платформа Open edX

Open edX – це потужна освітня платформа з відкритим вихідним кодом, яку було створено спільними зусиллями Массачусетського технологічного інституту та Гарвардського університету [41]. На рисунку 2.1 зображено цю навчальну платформу.

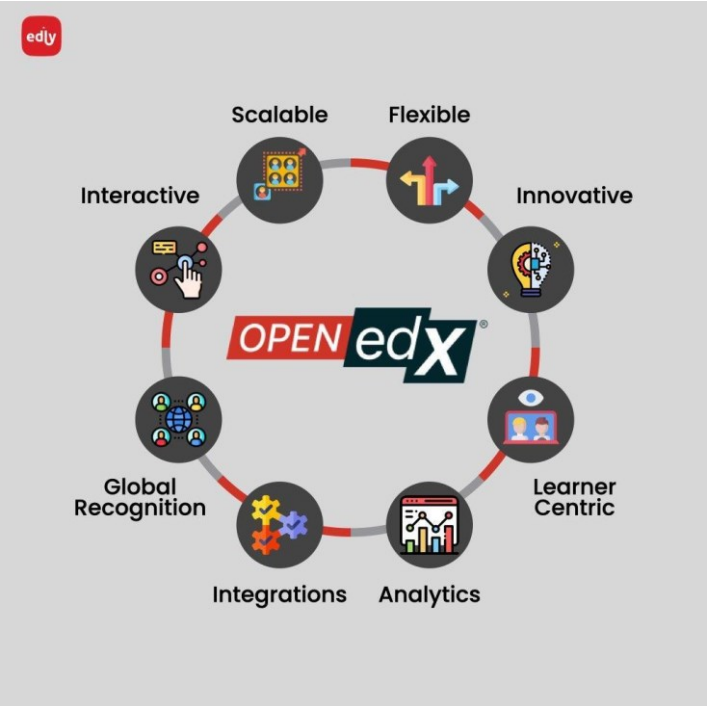

Рисунок 2.1 – Ілюстрація платформи Open edX.

Вона орієнтована на підтримку масових онлайн-курсів, корпоративного навчання та університетських програм і є однією з наймасштабніших платформ

у світі дистанційної освіти. Головною перевагою Open edX є її технологічна архітектура, побудована на фреймворку Django, з поступовим переходом фронтенду до React, а також використанням PostgreSQL як основної бази даних.

Бекенд системи реалізовані на Django, що забезпечує гнучке управління логікою роботи курсів, користувачами, доступами та інтеграцією з іншими сервісами. Open edX використовує Django у межах основного репозиторію edx-platform, який охоплює LMS та інструмент для створення курсів (Studio CMS). Ці два головні компоненти платформи забезпечують увесь життєвий цикл курсу: від створення матеріалу до його проходження студентами. Система побудована модульно, що дозволяє впроваджувати XBlock-плагіни – розширення для інтерактивних елементів, які також створюються в межах Django-екосистеми [42].

З погляду фронтенду, Open edX поступово відходить від традиційних шаблонів на основі Mako чи Jinja й активно впроваджує React, що дозволяє формувати сучасний, компонентний та інтерактивний інтерфейс користувача. Наприклад, нові сторінки панелі керування курсами, особистого кабінету або аналіз вже написані повністю на React із використанням Redux. Цей перехід не лише збагачує UI, а й робить архітектуру більш зрозумілою та гнучкою для масштабування, оскільки фронтенд стає відокремленим від бекенду.

Для бази даних Open edX використовує PostgreSQL, яка служить основою для збереження всієї навчальної інформації: користувачів, прогресу, курсів, результатів тестування, тощо. PostgreSQL обрана не лише за надійність, але й за сумісність із транзакціями, гнучкість при роботі з великими об’ємами даних і зручність у створенні складних запитів [43]. У масштабованих системах саме PostgreSQL дозволяє реалізувати ефективне кешування, реплікацію та забезпечення високої доступності.

Окрім цього, Open edX підтримує REST API для зовнішньої інтеграції, дозволяє взаємодіяти з мобільними застосунками, зовнішніми аналітичними системами та сервісами відео-конференцій. Усі API побудовані на основі Django REST Framework, що забезпечує масштабованість та безпечний доступ до даних.

В окремих модулях також реалізовані WebSocket-з'єднання для забезпечення реального часу в чатах або інтерактивних інструментах. Ще однією сильною стороною платформи є її орієнтація на DevOps: система підтримує розгортання через Docker, Helm та Kubernetes, що значно полегшує масштабування та CI/CD-процеси.

У результаті аналізу можна зробити висновок, що Open edX є зразком масштабованої системи для онлайн-освіти, яка демонструє ефективну взаємодію Django, React і PostgreSQL в одному технологічному стеку.

2.1.2 Платформа Moodle

Moodle – одна з найпоширеніших та найстаріших LMS, яка використовується в освітніх установах по всьому світу. На рисунку 2.2 зображено цю платформу.

Рисунок 2.2 – Ілюстрація платформи Moodle.

Платформа постійно розвивається та адаптується до нових викликів дистанційного навчання [44]. Основна реалізація Moodle базується на PHP, однак у контексті побудови сучасних систем, схожих за функціональністю, Moodle виступає орієнтиром щодо реалізації ключових сервісів – як backend-логіки для

управління курсами, ролями, оцінюванням, так і для адаптивного фронтенду [45].

З погляду фронтенду, Moodle поступово адаптує сучасні JavaScript-рішення, зокрема в окремих плагінах застосовується React або Vue. Тим не менше, більшість UI Moodle все ще побудована на шаблонах PHP з jQuery-орієнтованою логікою.

База даних Moodle реалізована на основі MySQL, PostgreSQL, MariaDB або SQLite, залежно від вибору під час інсталяції [46]. Однак PostgreSQL у Moodle визнана однією з найстабільніших і найпродуктивніших баз даних, що дозволяє ефективно обробляти великий обсяг користувацьких даних, історій навчання, журналів активності, тощо. Ще одним важливим аспектом є те, що Moodle підтримує плагін систему та інтерфейси зовнішньої інтеграції.

2.1.3 Платформа Canvas LMS

Canvas LMS – це сучасна, гнучка та масштабована система управління навчанням, розроблена компанією Instructure [47]. На рисунку 2.3 зображено платформу Canvas LMS.

Рисунок 2.3 – Ілюстрація платформи Canvas LMS

Платформа позиціонується як хмарне рішення для вищих навчальних закладів, шкіл і корпорацій, що забезпечує управління курсами, інтеграцію з інструментами третьої сторони, мобільний доступ та сучасний інтерфейс. Canvas LMS реалізована переважно на Ruby on Rails.

Canvas використовує чітке розділення логіки бекенду та фронтенду. Для взаємодії з фронтендом Canvas активно застосовує RESTful API [48], що дозволяє реалізовувати інтеграцію з мобільними додатками, зовнішніми сервісами, аналітичними інструментами та LTI-інструментами (Learning Tools Interoperability), що дає змогу гнучко управляти доступами, ролями та потоками даних між інтерфейсом та логікою серверу.

З погляду користувацького інтерфейсу, Canvas застосовує сучасні JavaScript-технології, зокрема React у нових модулях. Компонентна структура та односторінкові застосунки (SPA) дозволяють створювати адаптивний, зручний і швидкий інтерфейс [49].

Canvas також підтримує масштабовану багатокористувацьку архітектуру з ролями (адміністратор, викладач, студент), що реалізовано через модульну систему доступів.

Як база даних, Canvas LMS використовує PostgreSQL, що дає змогу обробляти великі обсяги навчальної інформації: користувачів, курсів, результатів тестування, медіа та аналітичних даних. Висока продуктивність PostgreSQL, підтримка складних зв’язків, JSON-полів, транзакцій та реплікацій позитивно вплинуло на інфраструктуру платформи. Завдяки PostgreSQL можливе гнучке масштабування системи, збереження історії дій користувачів, ведення логів та формування аналітичних звітів у реальному часі.

Canvas має відкриту документацію API, що забезпечує можливість глибокої кастомізації. Також вона підтримує LTI 1.3, що дозволяє легко інтегрувати сторонні навчальні інструменти.

Canvas LMS може слугувати прикладом грамотної реалізації архітектури освітньої платформи, побудованої за принципами розділення логіки, REST-архітектури та масштабованості.

2.2 Визначення цільової аудиторії та потреб користувачів, створення портрету «ідеального користувача»

Визначення користувачів, на яких орієнтований продукт, є необхідним кроком під час створення будь-якого цифрового продукту, зокрема – освітньої платформи. Саме розуміння потреб, очікувань та особливостей користувачів дозволяє точно сформувати функціональні модулі та визначити стратегічні напрями просування платформи. Цільова аудиторія (ЦА) – це група осіб, для яких продукт створюється, ті, хто зацікавлені в його застосуванні, що дозволяє отримати з нього практичну користь [50].

Коректно визначений сегмент користувачів збільшує шанси на успішну реалізацію ідеї та досягнення очікуваного результату. В основі дослідження лежить принцип розуміння потреб користувача, який може бути здійснено виключно після ґрунтовного аналізу його болей, мотивацій, стилю життя та цифрової поведінки [51].

Оскільки TutorSpace – це адаптивна навчальна платформа, основними групами користувачів є:

– учні та студенти (вікова категорія: 14–25 років, регіон: Україна), для яких важливо мати гнучкий графік, дистанційний доступ до навчання, підтримку з боку викладачів та можливість самостійного темпу;

– викладачі (вікова категорія: 26–50 років, регіон: Україна), які зацікавлені в переході на онлайн-формати, намагаються здобути нові знання й отримати прибуток, використовуючи можливості онлайн-курсів.

Методи дослідження потреб можуть включати як прямі інтерв’ю, анкетування, так і аналіз поведінки користувачів, роботу з відгуками та тестування прототипів [52]. Найбільш ефективним вважається комбінування кількох підходів, що дозволяє скласти об'єктивну картину про те, що саме очікують користувачі від платформи.

Після збору даних формують узагальнений образ «ідеального» користувача – це типова персона, яка репрезентує основну частину аудиторії та її запити. Такий підхід допомагає зробити платформу орієнтованою саме на реальні сценарії використання.

1 Ім’я: Андрій Литвин.

2 Вік: 23 роки.

3 Стать: чоловік.

4 Дохід: обмежений, студентська стипендія та фриланс-проєкти.

5 Статус: студент-магістрант.

6 Місце проживання: Чернівці, Україна.

7 Біографія: Андрій навчається за спеціальністю «Комп’ютерні науки» і активно цікавиться онлайн-курсами, що дозволяють йому здобувати додаткові знання у зручний час. Він живе окремо від сім’ї, тому змушений самостійно розподіляти свій час між навчанням, підробітком і саморозвитком. Через завантаженість розкладом університету та нестабільність графіку роботи, Андрій потребує платформи, яка дозволяє навчатись у будь-який момент та отримувати якісну підтримку від викладачів.

8 Цілі та мотивація: удосконалити технічні навички, знайти онлайн-курси, які не дублюють університетську програму, мати можливість спілкуватися з іншими слухачами, швидко отримувати доступ до навчальних матеріалів.

9 Больові точки: нестабільність часу, обмежений доступ до офлайн-навчання, складність у знаходженні якісного та зручного інструменту для онлайн-освіти.

10 Вплив платформи: TutorSpace забезпечує Андрію навчальний простір із гнучким графіком, адаптивним інтерфейсом, особистим кабінетом, можливістю слідкувати за прогресом, залишати запитання, отримувати миттєвий зворотний зв’язок від викладачів, і при цьому бути частиною навчальної спільноти.

11 Сценарій: переглядаючи Telegram-канал з технічними новинами, Андрій дізнається про платформу TutorSpace, проходить реєстрацію, обирає кілька курсів з темами, які його цікавлять, і починає навчання в зручний для себе час, поступово інтегруючись у спільноту та отримуючи нові знання.

Можна зазначити, що визначення цільової аудиторії, створення портрету користувача та глибоке розуміння його потреб дозволяє сформувати продукт, який відповідає очікуванням і забезпечує якісний користувацький досвід.

2.3 Аналіз, вибір та аргументація технологій Django, React та PostgreSQL

Під час розробки навчальної платформи особливо важливо обрати такий стек технологій, який забезпечить стабільність, масштабованість, швидкість розробки та зручність підтримки. Було прийнято рішення використовувати зв’язку Django, React та PostgreSQL, що забезпечує потужну, сучасну та актуальну архітектуру.

Django було обрано як серверний фреймворк завдяки його зрілості, безпеці та широкому функціональному набору [53]. Django – це фреймворк на мові Python, який дотримується принципу «батарейки включено», тобто містить у собі все необхідне для швидкого старту: систему автентифікації, ORM для роботи з базою даних, панель адміністратора, систему маршрутів і шаблонів [54]. Це дозволяє зосередитися на розробці бізнес-логіки замість створення базової інфраструктури. Його безпекові механізми (захист від XSS, CSRF, SQL-ін’єкцій) роблять його особливо привабливим для освітнього продукту, де важливо захищати персональні дані користувачів [55].

У таблиці 2.1 наведено порівняльну характеристику фреймворків для бекенду.

Таблиця 2.1 – Порівняльна характеристика фреймворків для бекенду

Критерій	Django	Laravel	Node.js
Мова програмування	Python	PHP	JavaScript
Архітектура	MTV (Model-Template-View)	MVC (Model-View-Controller)	Без шаблонної структури (гнучка)
Безпека	Високий рівень, вбудовані захисти	Середній, залежить від реалізації	Залежить від бібліотек
Вбудовані функції	Адмін-панель, ORM, аутентифікація	ORM, шаблони Blade, маршрути	Все реалізується вручну
Швидкість розробки	Висока	Середня	Висока, але потребує більше контролю
Документація	Дуже добра	Добра	Добра
Ліцензія	BSD	MIT	MIT

Найкращим вибором виступає Django завдяки його хорошій документації та активній спільноті, що пришвидшує пошук рішень у разі складностей, високий рівень безпеки та оптимізації процесу розробки.

За допомогою React можна розробляти інтерактивні користувацькі інтерфейси, які швидко реагують на взаємодію без необхідності повного перезавантаження сторінки [56]. Завдяки ньому стає можливим створення повторно використовуваних компонентів інтерфейсу, що спрощує розробку та супровід великих застосунків [57]. У таблиці 2.2 наведено порівняльну характеристику фреймворків для фронтенду.

Таблиця 2.2 – Порівняльна характеристика фреймворків для фронтенду

Критерій	**React**	**Angular**	**Vue.js**
Тип фреймворку	Бібліотека	Повноцінний фреймворк	Прогресивний фреймворк
Мова	JavaScript / JSX	TypeScript	JavaScript
Продуктивність	Висока	Висока	Висока
Крива навчання	Середня	Стрімка	Пологa
Інтеграція з моб. додатками	React Native	Capacitor / NativeScript	Vue Native / Quasar
Підтримка від спільноти	Величезна	Висока	Висока
Ліцензія	MIT	MIT	MIT

React було обрано для реалізації фронтенд-частини через його компонентну архітектуру, високу продуктивність та широке поширення у галузі. Важливою перевагою також є його сумісність із React Native, що забезпечує перспективу майбутньої розробки мобільного додатку без значного переписування логіки інтерфейсу [58]. Це критично важливо для освітньої платформи, орієнтованої на мобільних користувачів – студентів та викладачів.

PostgreSQL виступає в ролі основної реляційної бази даних платформи. Це потужна СУБД з відкритим кодом, що забезпечує стабільність, масштабованість і підтримку складних запитів [59]. Вона дозволяє працювати з транзакціями, зовнішніми ключами, тригерами, JSON-полями, що дає змогу гнучко зберігати як структуровані, так і напівструктуровані дані [60]. У таблиці 2.3 наведено порівняльну характеристику СУБД.

Таблиця 2.3 – Порівняльна характеристика СУБД

Критерій	PostgreSQL	MySQL	MongoDB
Тип СУБД	Реляційна	Реляційна	Документно-орієнтована
Підтримка транзакцій	Повна	Повна	Часткова
Підтримка JSON	Так (глибока інтеграція)	Так (обмежено)	Основний тип
Масштабованість	Горизонтальна і вертикальна	Вертикальна	Горизонтальна
Високонавантажені системи	Підходить	Підходить	Підходить
Підтримка складних запитів	Висока	Середня	Низька
Ліцензія	PostgreSQL License (Open Source)	GPL	Server Side Public License(SSPL)

PostgreSQL має чудову підтримку індексів, що дозволяє забезпечити високу продуктивність навіть за умов великої кількості користувачів та запитів. Її можна ефективно масштабувати вертикально або горизонтально, що відкриває можливості подальшого зростання проєкту без необхідності повної зміни архітектури [61].

Таким чином, поєднання Django, React і PostgreSQL утворює гнучку, надійну та масштабовану архітектуру для розробки сучасної навчальної платформи. Кожен із цих компонентів має високу адаптивність і спільно забезпечує не лише функціональність і продуктивність, а й готовність до подальшого розширення – включно з мобільною версією, інтеграціями через API, багатомовністю, а також глибоким аналітичним модулем. Такий вибір технологій є обґрунтованим кроком у розробці ефективного та конкурентоспроможного освітнього продукту.

2.4 Визначення ключових потреб веб-платформи та дослідження функціональних можливостей «TutorSpace»

Визначення ключових потреб користувачів і дослідження структури платформи «TutorSpace» є критично важливими етапами у процесі розробки системи. Зважаючи на те, що TutorSpace реалізована на основі Django як бекенд-фреймворку, React як клієнтської частини та PostgreSQL як основної СУБД, функціональне планування базується на чіткому розподілі ролей, даних і взаємодії між ними.

Навчальні платформи повинні враховувати ключові технічні виклики: вибір середовища розробки, зручність інтеграції сторонніх сервісів (наприклад, календарів, платіжних шлюзів, відео-зв'язку), стабільну роботу в умовах низької якості інтернет-з'єднання, а також масштабованість системи для обробки великої кількості одночасних користувачів. У цьому контексті особливої уваги заслуговує розробка надійної технічної платформи, яка підтримує багатофункціональність, адаптивний інтерфейс, швидкий обмін даними, захист інформації та просту інтеграцію з іншими освітніми інструментами.

На рівні клієнтської частини платформи реалізовано окремі функціональні модулі: реєстрація, автентифікація, пошук і фільтрація курсів, доступ до розкладу, сторінки з деталями курсу, підписка, перегляд прогресу, чат для спілкування, система сертифікації. Усі компоненти React зв'язані між собою через маршрутизацію (React Router) та працюють із централізованим API, розробленим на основі Django REST Framework.

React забезпечує інтерактивні курси та списки матеріалів, адаптивну верстку для різних пристроїв, форми реєстрації, входу, особистого кабінету, зв'язок із серверною частиною через REST API, реалізоване в Django, асинхронне завантаження даних через fetch або axios, оптимізацію продуктивності через React.memo, lazy loading, code splitting.

Комунікація фронтенда з бекендом відбувається через HTTP-запити, які забезпечують асинхронне оновлення даних. Для реалізації такої логіки

інтегровано системи управління контентом, вебсокети, черги повідомлень або cron-задачі для асинхронної логіки.

Бекенд-платформа на Django обробляє запити, виконує перевірку прав доступу, логіку запису на курс, перевірку тестів, керування чатами, а також фіксує всю інформацію про користувачів, курси, уроки та статистику активності. Реалізовано підтримку різних типів користувачів: студентів, викладачів і адміністраторів, що дозволяє відображати специфічний функціонал, як-от створення курсів, модерація контенту або генерація звітів про навчання.

У розробці платформи «TutorSpace» Django використовується спільно з DRF – бібліотекою, яка надає зручні інструменти для побудови RESTful API. Це дає змогу повністю відокремити фронтенд від бекенду та реалізувати сучасну архітектуру SPA та API, що у свою чергу:

– покращує масштабованість;
– дозволяє паралельну розробку frontend та backend;
– спрощує тестування;
– відкриває можливість підключення мобільних додатків.

PostgreSQL виступає як стабільна основа зберігання даних. Модельна структура Django використовує реляційні зв’язки (ForeignKey, ManyToMany), які відповідають за зв’язки між студентами й курсами, матеріалами, результатами завдань, записами в чаті, тощо. Завдяки використанню PostgreSQL можливо реалізувати складні запити, агрегації, статистику успішності та моніторинг активності користувачів без втрати продуктивності. PostgreSQL виступає як база даних, що поєднується з Django, pgAdmin або psql, Celery/Redis.

Чітко визначена інформаційна структура TutorSpace допомагає структуровано оглянути використані рішення та відслідкувати шлях користувача від реєстрації до отримання сертифікату. Основні функціональні блоки включають:

– головну сторінку з інформацією про доступні курси, прогрес користувача, рекомендовані напрями;

– особистий кабінет користувача, який відображає записи на курси, історію активності, доступ до сертифікатів;

– панель викладача з можливістю створення, редагування й видалення курсів, завантаження матеріалів, перегляду відгуків і прогресу студентів;

– сторінки пошуку з реалізованим механізмом фільтрації за категоріями, рейтингами, популярністю;

– модуль повідомлень та обговорень (чат), реалізований через WebSocket або асинхронні запити;

– API для мобільного застосунку або сторонніх систем, які можуть взаємодіяти з TutorSpace через токенізовану авторизацію.

Ця функціональна архітектура забезпечує масштабованість, чітке розмежування обов'язків між складовими системи та дозволяє підтримувати стабільну роботу навіть при зростанні навантаження. Вся логіка зосереджена на досягненні повного покриття освітнього процесу, з урахуванням ролей, прав доступу, звітності й гнучкого керування контентом.

2.5 Висновок до другого розділу

У другому розділі розглянуто комплексний функціональний аналіз, що заклав основу для технічної реалізації навчальної платформи «TutorSpace». Вивчення існуючих рішень у сфері дистанційної освіти дозволило оцінити архітектурні особливості конкурентних систем, їхні переваги та недоліки, які стали базою для вибору найкращого програмного стеку.

Завдяки технічному аналізу платформ, таких як Open edX, Moodle і Canvas LMS, вдалося виявити найбільш ефективні підходи до побудови модульної структури, підтримки масштабованості, розмежування доступу та використання API. Це дозволило сформулювати ключові технічні вимоги до власного продукту та обґрунтувати вибір архітектури, заснованої на Django, React і PostgreSQL.

Окрему увагу приділено дослідженню цільової аудиторії. На основі аналізу потреб студентів та викладачів сформовано модель «ідеального»

користувача, що забезпечило клієнтоорієнтований підхід до структури платформи та розробки її функціоналу. Під час проєктування враховувалися ключові запити, такі як доступ до навчання в зручний час, підтримка мобільних пристроїв, індивідуальні освітні траєкторії та інтерактивна взаємодія.

Важливим аспектом стала побудова інформаційної архітектури, яка передбачає логічну структуру навігації між основними модулями: курсами, профілем користувача, чата, системою тестування та курсами. Вона стала основою для реалізації функціональних модулів за допомогою REST API і компонентного підходу у фронтенді.

У розділі також розглянуто переваги обраного стеку технологій: Django як надійного фреймворку для серверної логіки, React як інструменту побудови інтерактивного інтерфейсу та PostgreSQL як стабільної й потужної бази даних. Їх сумісність, масштабованість і гнучкість є критично важливими для створення сучасного, конкурентоспроможного освітнього веб-застосунку, що може бути легко адаптований до потреб мобільної розробки.

Загалом проведене дослідження заклало надійне технічне підґрунтя для наступного етапу реалізації – побудови функціонального прототипу платформи «TutorSpace», що відповідатиме як технологічним вимогам, так і реальним потребам кінцевих користувачів.

РОЗДІЛ 3. РОЗРОБКА ФУНКЦІОНАЛУ ДЛЯ НАВЧАЛЬНОЇ ПЛАТФОРМИ «TUTORSPACE»

У цьому розділі розглянуто реалізацію основних складових інтерфейсу користувача та логіки для освітньої платформи «TutorSpace» з точки зору архітектури системи на основі React і Django. Платформа побудована як SPA, де React відповідає за рендеринг інтерфейсу, а Django реалізує логіку обробки даних і API через REST.

3.1 Опис основних технологій та засобів для розробки інтерфейсу користувача

Архітектура фронтенд частини застосунку побудована на основі підходу FSD (Feature-Sliced Design) [62] – сучасного принципу організації структури фронт-енд проєктів, орієнтованого на масштабування, модульність та підтримуваність коду. Цей підхід дозволяє відокремлювати логіку бізнес-функціональності від інфраструктурної частини. Основні принципи FSD:

– поділ за рівнями абстракції (layers);
– модульність та ізоляція;
– принцип "використовуй, але не заходь усередину";
– гнучкість і масштабованість.

Використання FSD значно підвищило структурованість проєкту, а також забезпечило гнучку основу для майбутнього розширення функціоналу (див. рисунок 3.1).

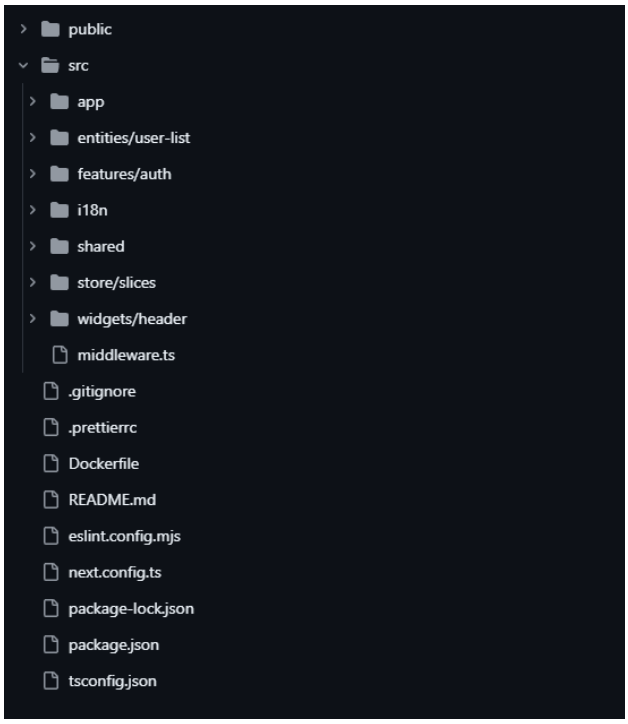

Рисунок 3.1 – Організація структури фронтенд застосунку

Можна виділити основні директорії застосунку:

– App – ця директорія виступає в ролі маршрутизатора для створення роутингу та навігації по проєкту. Для правильного підключення з використанням Next.js (App router) потрібно створити директорію з необхідною назвою, яка буде виступати в ролі URL шляху, для відображення потрібно створити файл page.tsx та повернути звичний функціональний компонент React. Також в кожній директорії окрім файлу page.tsx можуть міститись додаткові файли як: layout.tsx, not-found.tsx, loading.tsx.

– Entities – ця директорія містить предметні сутності, які є базовими структурними одиницями застосунку. Вони не залежать від конкретних бізнес-функцій, але можуть використовуватись у багатьох фічах. До прикладу, це можуть бути компоненти: список користувачів, список курсів, коментар.

– Features – ця директорія містить функціональні одиниці – окремі бізнес-функції, які вирішують одну задачу (авторизація, зміна профілю, фільтрація). Кожна фіча працює з однією або кількома сутностями.

– I18n – ця директорія містить усю логіку, пов’язану з інтернаціоналізацією. Тут знаходиться конфігурація бібліотеки i18next, та мовні переклади у внутрішній директорії messages.

– Shared – містить універсальні, повторно використовувані частини, які не прив’язані до бізнес-логіки або сутностей. Це бібліотеки, UI-компоненти, утиліти, константи, хелпери, типи, підключення до API тощо.

– Store – у цій директорії знаходяться частини єдиного сховища застосунку розділеного на окремі директорії зі своєю внутрішньою структурою, так звані слайси. Структура кожного слайсу містить такі файли: index.ts, initial-state.ts, selectors.ts, types.ts.

– Widgets – ця директорія містить композиційні UI-блоки, які збирають кілька features, entities або shared-компонентів. Вони не є сторінками, але складають частину інтерфейсу (наприклад, хедер, сайдбар, картка профілю).

Для реалізації користувацького інтерфейсу було обрано UI бібліотеку Material UI. Цей фреймворк є однією з найпопулярніших бібліотек компонентів для React, вона допомагає реалізувати основні принципи дизайну Material Design [63], розроблені компанією Google. Завдяки своїй гнучкості, широкому набору готових елементів та активній спільноті розробників, Material UI дозволяє швидко та ефективно створювати інтерфейси, що відповідають сучасним вимогам до зручності, естетики та доступності.

Серед ключових переваг бібліотеки можна виділити наступні:

– великий набір шаблонних компонентів Material UI надає великий вибір вже готових елементів інтерфейсу – кнопок, форм, таблиць, діалогових вікон тощо, що значно пришвидшує процес розробки;

– підтримка адаптивного дизайну, компоненти автоматично підлаштовуються під різні розміри екранів, що дозволяє створювати мобільно-дружні інтерфейси без значних додаткових зусиль;

– гнучка система стилізації, розробник має змогу легко змінювати вигляд компонентів за допомогою CSS-in-JS підходу або використання тем.

Material UI дозволяє змінювати глобальний вигляд додатку за допомогою системи тем. Кастомізація тем забезпечується за рахунок створення об'єкта теми за допомогою функції createTheme, який далі передається через ThemeProvider. В лістингу 3.1 зображено створення провайдера як клієнтського компонента та присвоювання глобального об’єкту теми до обгортки проєкту.

Лістинг 3.1 – Структура підключення об’єкту теми Material UI

```
'use client';
import type { IThemeRegistryProps } from '@/shared/types/theme';
import createTheme from '@/shared/theme';

import { CssBaseline } from '@mui/material';
import { ThemeProvider } from '@emotion/react';

export default function ThemeRegistry(props:
Readonly<IThemeRegistryProps>) {
  const { options, children, themeMode } = props;

  const theme = createTheme({ themeMode });

  if (!window) return null;

  return (
    <>
      <CacheProvider value={cache}>
        <ThemeProvider theme={theme}>
          <CssBaseline />
          {children}
        </ThemeProvider>
      </CacheProvider>
    </>
  );
}
```

З метою забезпечення надійної, передбачуваної та масштабованої роботи системи, було прийнято рішення використовувати TypeScript як основну мову для написання фронтенд частини. Одним з ключових напрямів застосування TypeScript у проєкті стало створення типів і інтерфейсів для UI-компонентів, функцій, API-запитів та структури даних. Це дало змогу не лише покращити взаємодію між окремими модулями, а й зробити код самодокументованим, безпечним та зручним для підтримки. Основні переваги типізації у проєкті:

– інтерфейси та типи дозволили описувати чіткі контракти для props компонентів, відповідей API, форм введення та інших частин системи;

– кожен компонент, створений з використанням бібліотеки Material UI, має строго типізовані властивості, що полегшує автодоповнення в редакторі коду та знижує ймовірність помилок у передачі параметрів;

– завдяки типам стало можливим гарантувати, що дані, передані між фічами, сутностями та формами, відповідають очікуваній структурі, що особливо важливо у проєктах побудованих на архітектурі FSD.

Для ефективного керування станом у межах фронтенд частини застосунку було обрано Redux Toolkit [64] – сучасний і стандартизований інструмент для роботи з Redux, що спрощує процес створення, оновлення та організації глобального стану. Redux Toolkit надає оптимізований, безпечний і зрозумілий API, який значно зменшує обсяг шаблонного коду порівняно з класичним Redux. Основні причини вибору Redux Toolkit:

– усі ключові частини стану застосунку (наприклад, інформація про користувача, налаштування, повідомлення, завантаження даних) зберігаються в одному місці, що забезпечує легку інтеграцію між різними модулями та функціональністю;

– Redux Toolkit має вбудовану підтримку TypeScript, що дозволяє строго типізувати стан, дії (actions), редюсери та асинхронні запити (thunks), зменшуючи ризик помилок;

– Redux Toolkit організовує стан у вигляді окремих "слайсів", кожен з яких відповідає за певну область застосунку. Це відповідає принципам FSD-архітектури та покращує масштабованість.

Впровадження Redux Toolkit дозволило зробити роботу зі станом чіткою, масштабованою та контрольованою, що є критично важливим при створенні складних фронтенд застосунків з багатьма взаємодіючими компонентами. Ініціалізацію глобального сховища зображено в лістингу 3.2.

Лістинг 3.2 – Ініціалізація глобального сховища для фронтенд застосунку

```
import type { ThunkAction, UnknownAction } from
'@reduxjs/toolkit';

import authSlice from '@/store/slices/auth';
import { baseApi } from '@/shared/api';
import { configureStore } from '@reduxjs/toolkit';
import { UseExtraArguments } from '@/shared/hooks/store/use-extra-
arguments';

export const makeStore = <T>(extraArgument: T) => configureStore({
  reducer: {
    [baseApi.reducerPath]: baseApi.reducer,
    [authSlice.name]: authSlice.reducer,
  },
  devTools: process.env.NODE_ENV !== 'production',
  middleware: (getDefaultMiddleware) => {
    return getDefaultMiddleware({
      thunk: { extraArgument }
    }).concat(baseApi.middleware);
  }
});
type TAppStore = ReturnType<typeof makeStore>;
export type TExtraArguments = ReturnType<typeof
UseExtraArguments>;
export type TAppStoreState = ReturnType<TAppStore['getState']>;
export type TAppStoreDispatch = TAppStore['dispatch'];
export type TAppThunk<R = void> = ThunkAction<R, TAppStoreState,
any, UnknownAction>;
```

Ще одним потужним інструментом Redux Toolkit є RTK Query – офіційна бібліотека для спрощеного виконання запитів до бекенду, з кешуванням, оновленням, інвалідованістю та автоматичним керуванням статусами запиту. Переваги використання RTK Query:

– автоматичне керування статусами (isLoading, isError, data, error);

– вбудоване кешування результатів та повторне використання запитів;

– просте оновлення даних після мутацій;

– зниження потреби у створенні окремих thunks та редюсерів.

Прикладом поєднання вище перечислених технологій виступає написане під’єднання до API для авторизації користувача в систему за допомогою RTK Query, зображене в лістингу 3.3.

Лістинг 3.3 – Реалізація RTK Query для пыдключення до API.

```
import {
  TLoginResponse
} from '@/shared/api/auth/types';

import { baseApi } from '@/shared/api';
import {
  loginResponseSchema
} from '@/shared/api/auth/models';


import {
  API_GET_AUTH_ME,
  API_POST_AUTH_LOGIN
} from '@/shared/constants';

export const authApi = baseApi.injectEndpoints({
 endpoints: create => ({
    // POST
    postLogin: create.mutation<TLoginResponse,
IAuthLoginPayload>({
      query: payload => ({
        url: API_POST_AUTH_LOGIN,
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(payload)
      }),
      transformResponse: (res: unknown) =>
loginResponseSchema.parse(res),
      invalidatesTags: ['User']
    }),
  overrideExisting: true
});
```

Щоб дотримуватись типізації в даних, які відправляються до запитів та відповіді від сервера були описані типи та інтерфейси на основі TypeScript, зображені в лістингу 3.4.

Лістинг 3.4 – Написання інтерфейсів для типізації запитів RTK Query.

```
import type { TSocialTypes } from '@/features/auth/types';

import { z } from 'zod';
import {currentUserSchema, loginResponseSchema,
registerResponseSchema} from '@/shared/api/auth/models';

export interface IAuthLoginPayload {
  email: string;
  password: string;
```

```
}

export type TCurrentUser = z.infer<typeof currentUserSchema>;
export type TLoginResponse = z.infer<typeof loginResponseSchema>;
export type TRegistrationResponse = z.infer<typeof
registerResponseSchema>;
```

Однією з основних вимог до сучасних веб-застосунків є надійна валідація вхідних даних – як тих, що надходять від користувачів через форми, так і тих, що отримуються з зовнішніх джерел. Щоб забезпечити надійність та відповідність типів не лише під час компіляції, а й у процесі виконання застосунку, у проєкті було вирішено використовувати бібліотеку Zod.

Zod – це декларативна бібліотека, орієнтована на TypeScript, розроблена для перевірки схем [65]. Вона дозволяє визначити структуру даних як схему та гарантує, що фактичні значення відповідають цій схемі. За допомогою Zod можна безперешкодно поєднувати типізацію та перевірку, забезпечуючи подвійний рівень коректності даних.

Також значною перевагою є повноцінна інтеграція з React Hook Form – сучасною бібліотекою для побудови форм у React. Завдяки цьому поєднанню стало можливим ефективно реалізувати гнучку, типізовану та перевірену валідацію користувацьких даних без надмірної складності. React Hook Form дозволяє працювати з формами з високою продуктивністю та мінімальними рендерингами, а інтеграція з Zod – через адаптер zodResolver – забезпечує централізовану валідаційну логіку, яка є зрозумілою, декларативною та легко масштабованою.

За допомогою цієї інтеграції були розроблені всі форми для введення даних користувачем. Наведено для прикладу форму авторизації на рисунку 3.2.

Рисунок 3.2 – Форма авторизації користувача

Для створення візуального інтерфейсу форми було використано компоненти з бібліотеки Material UI. Як контейнер для обробки введених даних застосовано стандартний HTML-елемент <form>, який відповідає за відправлення даних та підтримує базову семантику. Обробка значень полів вводу, керування станом форми та її валідація реалізовані за допомогою бібліотеки React Hook Form, що забезпечує високу продуктивність та просту інтеграцію з зовнішніми валідаційними механізмами. Для реалізації сильної типізації та runtime-валідації було використано бібліотеку Zod, яка дозволяє описати схему даних, виводити типи та контролювати коректність введеної інформації.У лістингу 3.5 зображено інтеграцію Zod з React Hook Form для побудови форми авторизації.

Лістинг 3.5 – Інтеграція Zod з React Hook Form для роботи з формою.

```
export const AuthForm = (props: Readonly<IAuthFormProps>) => {
const {
    register,
    formState: { errors },
    handleSubmit
  } = useForm<TAuthFormSchema>({
    mode: 'onSubmit',
    resolver: zodResolver(
```

```
      formType === 'login' ? loginFormSchema :
registrationFormSchema
    ),
    defaultValues: authFormDefaultValues
  });

return (
    <form
      onSubmit={handleSubmit(data => {
        startAuthorizationTransition(async () => {
          await onSubmit(data);
        });
      })}
    >
          …
    </form>
)
};
```

Для забезпечення контролю доступу до захищених розділів застосунку було реалізовано спеціальне middleware-рішення, яке перевіряє, чи користувач є авторизованим у системі. Це є необхідним з міркувань безпеки та захисту приватних даних користувача, а також для коректного функціонування застосунку згідно з його бізнес-логікою. Механізм працює на рівні маршрутизації і дозволяє обмежити доступ до внутрішніх сторінок, таких як профіль, панель керування або особистий кабінет. У випадку, якщо користувач не проходив автентифікацію або його сесія недійсна, middleware автоматично перенаправляє його на сторінку входу.

Middleware реалізовано з урахуванням особливостей обраного фреймворку. Перевірка автентифікації базується на наявності збережених облікових даних у глобальному стані об’єкта Auth.js. Приклад реалізації наведено в лістингу 3.6.

Лістинг 3.6 – Реалізація middleware для захищеного роутингу.

```
import type { NextRequest } from 'next/server';

import routes from '@/shared/config/routes';
import { auth } from '@/shared/config/auth';
import { NextResponse } from 'next/server';

export default async function middleware(request: NextRequest) {
```

```
  const { pathname } = request.nextUrl;

  const session = await auth();

  const isProtected = routes.some(route =>
pathname.startsWith(route));
  if (isProtected && !session) {
    return NextResponse.redirect(new URL(routes.login.path,
request.url));
  }

  return NextResponse.next();
}
```

Використання middleware для контролю автентифікації дозволяє гарантувати, що доступ до важливих частин застосунку отримують лише авторизовані користувачі. Це є стандартною практикою у сучасних веб-застосунках, яка підвищує захищеність, стабільність і надійність всієї системи.

Для розширення аудиторії користувачів та забезпечення інтернаціональної доступності веб-застосунку було реалізовано підтримку кількох мов інтерфейсу. Це дозволяє користувачам взаємодіяти з платформою у звичному мовному середовищі, підвищує зручність користування, довіру та відповідність сучасним вимогам до інтерфейсів. З цією метою до проєкту було інтегровано бібліотеку i18next – один з найпопулярніших інструментів для реалізації багатомовності у застосунках, зокрема у зв'язці з React через плагін react-i18next. У лістингу 3.7 зображено підключення мовного середовища до проєкту.

Лістинг 3.7 – Підключення інтегнаціональної бібліотеки i18n до проєкту.

```
import type { TLocalesList } from '@/shared/types/i18n';

import * as constants from '@/shared/constants';

import { getRequestConfig } from 'next-intl/server';
import { cookies } from 'next/headers';

export default getRequestConfig(async () => {
  const locale: TLocalesList = (await
cookies()).get(constants.COOKIE_LOCALE)?.value as TLocalesList |
undefined || 'en';

  return {
    locale,
```

```
    messages: (await
import(`@/i18n/messages/${locale}.json`)).default
  };
});
```

У межах розробки веб-застосунку важливим аспектом стало збереження персоналізованих налаштувань користувача та критичних даних для роботи застосунку. До таких даних належать: токен авторизації, обрана тема оформлення, обрана мова інтерфейсу.

В лістингу 3.8 зображено механізм збереження параметрів обраної мови до сховища cookies.

Лістинг 3.8 – Збереження обраної мови до сховища cookies.

```
export const useLocations = (): IUseLocations => {
  const router = useRouter();
  const [locale, setLocale] = useState<TLocalesList>('en');
  const changeLocale = useCallback(
    (locale: TLocalesList) => {
      setLocale(locale);
      document.cookie = `${constants.COOKIE_LOCALE}=${locale}`;
      router.refresh();
    },
    [router]
  );

  useEffect(() => {
    const cookieLocale = getCookieOnTheClient<TLocalesList>(
      constants.COOKIE_LOCALE
    );
    if (cookieLocale) {
      setLocale(cookieLocale);
      return;
    }
    const browserLocale = navigator.language
      .slice(3, 5)
      .toLowerCase() as TLocalesList;
    setLocale(browserLocale);
    document.cookie =
`${constants.COOKIE_LOCALE}=${browserLocale}`;
    router.refresh();
  }, [router]);
  return {
    locale,
    changeLocale
  };
};
```

З урахуванням вимог до безпеки, доступності в SSR [66] та необхідності взаємодії з бекендо, у цьому проєкті було прийнято рішення використовувати cookies як основний спосіб збереження ключових користувацьких параметрів.

3.2 Опис основних структур для розробки бізнес логіки платформи

Було використано низку інструментів, які дозволили забезпечити повноцінний процес розробки, тестування, документування та розгортання застосунку. Центральним середовищем розробки для backend-частини виступала PyCharm Community Edition – потужне інтегроване середовище, орієнтоване на розробку Python-проєктів. Завдяки підтримці фреймворку Django, з яким реалізовано логіку серверної частини системи, PyCharm дозволяє ефективно керувати структурою додатку, міграціями бази даних, а також містить комфортний інтерфейс для налагодження коду та інтеграції з Git [67].

Одним з важливих інструментів у контексті роботи з базою даних є DataGrip – універсальний клієнт для SQL-серверів, зокрема PostgreSQL. У рамках проєкту він використовувався для візуального перегляду та редагування таблиць, тестування SQL-запитів, налагодження зв’язків між сутностями та контролю стану збережених даних. DataGrip допоміг у глибшому аналізі структури бази, перевірці запитів, які формує ORM, та у вирішенні задач, пов’язаних із продуктивністю [68].

Для організації командної роботи, зберігання коду та контролю версій використовувався GitHub. Репозиторій містив основну логіку платформи, окремі гілки для нових функцій, пул-реквести для рев’ю та систему тегів для стабільних релізів. GitHub дозволив ефективно координувати розробку між учасниками, документувати зміни та автоматизувати частину перевірок за допомогою GitHub Actions [69].

Управління завданнями у проєкті велося через Trello – інструмент з організації роботи у вигляді дошок. Розробка була поділена на фази: планування, розробка, тестування та перевірка. Завдання позначалися мітками, мали чіткі

дедлайни, контроль виконання, перевірку залежностей. Таке середовище дозволяло легко координувати як backend-, так і frontend-команди, забезпечуючи прозору логіку виконання задач [70].

Для тестування API та перевірки відповіді від серверної частини застосовувався Insomnia – сучасний REST-клієнт, що дозволяє створювати колекції запитів, швидко перемикатися між оточеннями (наприклад, staging/production), а також комфортно працювати з JWT-авторизацією. Він був особливо корисним під час налаштування аутентифікації, тестування складних endpoint'ів та перевірки, як frontend-логіка взаємодіє з backend [71].

Комплексне використання цих інструментів дозволило реалізувати розробку навчальної платформи «TutorSpace» на високому рівні, забезпечивши якісну інтеграцію між компонентами, контроль стабільності, продуктивність і масштабованість. Усі застосунки були використані у своїх останніх стабільних версіях, що дозволяє підтримувати актуальність архітектури проєкту відповідно до сучасних вимог галузі.

Під час розробки навчальної платформи «TutorSpace» використовувався набір бібліотек, що забезпечують функціональність, безпеку, масштабованість і зручність інтеграції. Django 4.2.16 став основою серверної частини, забезпечуючи структуру MVC, маршрутизацію, ORM, систему міграцій і адмін-панель. Службові бібліотеки, як-от asgiref, sqlparse, pytz і typing_extensions, відповідають за асинхронну підтримку, парсинг SQL, роботу з часовими зонами та типізацію.

Для побудови REST API використано djangorestframework, а для реалізації безпечної аутентифікації – djangorestframework_simplejwt, який реалізує JWT-токени. Бібліотека djoser забезпечує швидке розгортання ендпоінтів для авторизації, реєстрації та відновлення пароля. Для автоматичної генерації Swagger-документації API застосовано drf-yasg.

Підтримка авторизації через соціальні мережі реалізована за допомогою social-auth-app-django, social-auth-core та python3-openid. Робота з OAuth2 токенами реалізована бібліотеками PyJWT, oauthlib, requests-oauthlib, що

забезпечують безпечну інтеграцію з зовнішніми сервісами. Бібліотеки cryptography, cffi, pycparser і defusedxml відповідають за шифрування, криптографічний захист та безпечну обробку XML.

Робота з PostgreSQL реалізована через драйвер psycopg2-binary, який забезпечує стабільне з'єднання з базою даних. Inflection – ця бібліотека використовується для обробки назив у генерації моделей.

Хмарна інтеграція з AWS виконана через бібліотеки boto3, botocore, s3transfer і jmespath, що дозволяють завантажувати та зчитувати файли з Amazon S3, надсилати пошту через SES та керувати іншими ресурсами AWS.

Для надсилання поштових повідомлень через Amazon SES використовується django-ses. HTTP-запити реалізовано через requests, а супутні бібліотеки urllib3, certifi, idna, chardet забезпечують підтримку безпечних з'єднань.

Крос-доменна взаємодія з фронтендом реалізована за допомогою django-cors-headers, який дозволяє обробляти CORS-запити. Для зберігання конфіденційних налаштувань середовища (ключі доступу, email-параметри, тощо) застосовано python-dotenv.

Серед інших бібліотек: six – для забезпечення сумісності між версіями Python, uritemplate – допомагає у роботі із URI-шаблонами, PyYAML – для обробки YAML-файлів, а packaging – для коректного порівняння версій залежностей.

Фреймворк django сформував основну директорію проєкту у якій зберігаються основні файли для роботи. Також він формує декілька дочірніх файлів за межами цієї папки. Вони також обов’язкові для коректної роботи системи. Тому розглянуто структуру його директорій та файлів. На рисунку 3.3 зображено структуру файлової системи бекенду.

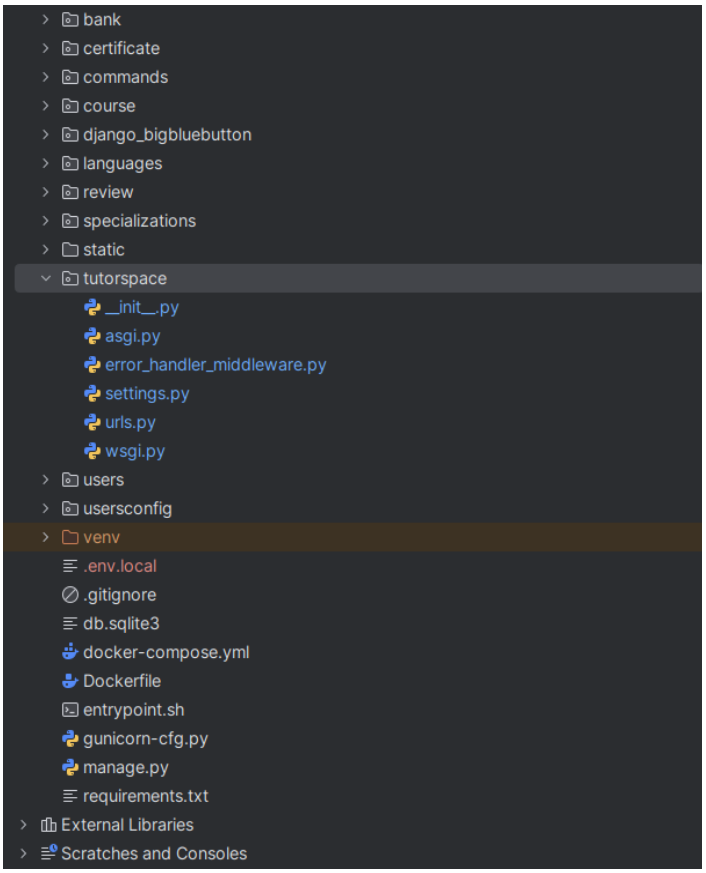

Рисунок 3.3 – Файлова структура проєкта

Директорія static відіграє важливу роль у забезпеченні клієнтської частини додатку. Вона використовується для зберігання статичних ресурсів – таких як стилі CSS, JavaScript-файли, шрифти, зображення. Ці файли не змінюються під час виконання програми і використовуються як фронтенд-ресурси для побудови візуального середовища користувача. Django під час розгортання автоматично збирає вміст усіх static-папок з кожного додатку в одну спільну директорію (за допомогою команди collectstatic), що дозволяє ефективно обслуговувати їх за допомогою HTTP-сервера у production-середовищі. Під час розробки ці файли доступні через шаблонний тег {% static %}.

Директорія tutorspace є ядром конфігураційної частини проєкту Django. Вона була створена автоматично після ініціалізації проєкту командою django-admin startproject tutorspace і містить усі важливі елементи для запуску серверної частини застосунку. Зокрема, файл __init__.py вказує Python, що ця директорія є пакетом. Файл asgi.py відповідає за запуск додатку у середовищах, які

підтримують асинхронну обробку запитів (наприклад, WebSocket або SSE), тоді як wsgi.py використовується як точка входу для класичних WSGI-серверів на зразок Gunicorn чи uWSGI. Ці два файли дозволяють запускати Django в різних типах середовищ.

Файл settings.py – це головне місце, де зберігаються всі конфігураційні параметри: список встановлених додатків, база даних, middleware, автентифікація, CORS, JWT, email-інтеграція через Amazon SES, тощо. Його структура дозволяє легко змінювати ключові параметри без необхідності втручання в логіку додатку. Окремий файл urls.py виконує функцію центрального маршрутизатора, зв'язуючи запити до URL-адрес з конкретними views, а також дозволяє підключати маршрути окремих модулів проєкту. Файл error_handler_middleware.py містить реалізацію власного middleware, що перехоплює помилки і дозволяє обробляти їх на рівні всієї програми – наприклад, замінювати стандартні помилки Django на JSON-відповіді для API. Такий підхід дозволяє централізовано керувати помилками.

Директорія venv є віртуальним середовищем Python, яке створюється з метою ізоляції середовища розробки від глобальних пакетів системи. Це дозволяє встановлювати залежності лише для цього конкретного проєкту, уникаючи конфліктів між різними версіями бібліотек. У цій директорії містяться всі необхідні пакети, такі як Django, DRF, boto3, djoser, social-auth-app-django тощо. При роботі над проєктом середовище активується локально, після чого розробник працює у замкнутому контексті з відповідними версіями бібліотек. У git-репозиторій venv ніколи не включається, оскільки її можна відновити за допомогою файлу requirements.txt.

Файл .env.local є конфігураційним і використовується для зберігання змінних середовища, які не слід розкривати у відкритому коді – таких як ключі доступу до Amazon SES, секретні ключі Google OAuth, токени Facebook або внутрішні параметри, як-от DJANGO_DEBUG, ALLOWED_HOSTS, DATABASE_URL, тощо. Цей файл читається через бібліотеку dotenv в settings.py

і дозволяє відокремити конфігурацію від логіки, що спрощує управління різними середовищами (розробка, staging, production).

Файл .gitignore містить список файлів і директорій, які не слід включати до системи контролю версій Git. До них належать venv/, __pycache__/, .env.local, бази даних у форматі SQLite (наприклад, db.sqlite3) та інші службові артефакти.

Файл db.sqlite3 зазвичай присутній у процесі розробки, коли використовують SQLite як тимчасову базу даних. Проте у цьому проєкті основна база визначена як PostgreSQL, отже цей файл може використовуватися лише для тестування чи локальних швидких перевірок.

Файл manage.py є стандартним скриптом Django, що забезпечує точку входу для виконання команд: запуск сервера (runserver), застосування міграцій (migrate), створення користувачів (createsuperuser), запуск shell-сесій, тестування, тощо. Він завжди розташовується в кореневій директорії та викликає налаштування, вказані в tutorspace.settings.

Файл requirements.txt перелічує всі необхідні пакети Python, з фіксацією їх конкретних версій, щоб забезпечити стабільність. Він використовується під час першого запуску або розгортання, зазвичай з командою pip install -r requirements.txt. У цьому проєкті файл містить такі ключові залежності як Django, Django REST Framework, SimpleJWT, boto3, django-ses, djoser, drf-yasg та інші.

Файл Dockerfile містить інструкції для створення образу Docker, в якому розгортається проєкт. Він визначає базовий образ Python, встановлює залежності, копіює код у контейнер і вказує команду запуску. Зазвичай він використовується у зв’язці з docker-compose.yml, який забезпечує конфігурацію багатоконтейнерного середовища (наприклад, додаток, база даних).

Файл docker-compose.yml є конфігураційним і визначає сервіси, які мають бути запущені одночасно. У типовій конфігурації для Django вказується web (сам застосунок) та db (контейнер PostgreSQL). Також тут задаються змінні середовища, порти, залежності між контейнерами, точки монтованих томів.

Файл entrypoint.sh використовується як скрипт ініціалізації при запуску контейнера. У ньому зазвичай міститься логіка, яка перевіряє, чи застосовано міграції, створено суперкористувача, запущено сервер, тощо. Це ключовий елемент CI/CD, особливо коли застосунок розгортається автоматично на сервері.

Файл gunicorn-cfg.py – це конфігураційний файл для запуску WSGI-серверу Gunicorn. Тут можна визначити кількість воркерів, тайм-аути, логування та інші параметри, які впливають на продуктивність застосунку у продакшн-середовищі. Gunicorn разом із nginx часто використовується для сервінгу Django у production.

У першій частині файлу settings.py здійснюється імпорт модулів, які використовуються у роботі із файловою системою, керування середовищем, а також безпекових та конфігураційних параметрів застосунку. У лістингу 3.9 наведено фрагмент коду імпорту модулів.

Лістинг 3.9 – Імпорт модулів

```
import os

from os import getenv, path
from pathlib import Path
from django.core.management.utils import get_random_secret_key
import dotenv
from django_ses.settings import AWS_SES_REGION_NAME
from datetime import timedelta
```

Модулі os, pathlib і getenv використовуються для правильної роботи файлової системи та зчитування змінних середовища, що дозволяє гнучко керувати налаштуваннями залежно від середовища запуску (наприклад, локально чи на сервері). Модуль dotenv відповідає за підключення .env.local файлу, в якому зберігаються чутливі дані, такі як ключі API. Це дозволяє уникати прямого запису таких даних у коді. У лістингу 3.10 наведено фрагмент коду для завантаження ключів доступу та встановлення базової директорії.

Лістинг 3.10 – Завантаження файлу з чутливими даними

```
BASE_DIR = Path(__file__).resolve().parent.parent

dotenv_file = BASE_DIR / ".env.local"

if path.isfile(dotenv_file):
    dotenv.load_dotenv(dotenv_file)
```

Встановлюється базова директорія проєкту (BASE_DIR) та перевіряється наявність .env.local. Якщо файл існує, змінні середовища з нього завантажуються в середовище виконання. Це забезпечує зручне розділення налаштувань розробки і продакшну без потреби змінювати код.

Далі конфігуруються основні параметри безпеки, зокрема секретний ключ (SECRET_KEY), який або зчитується з середовища, або генерується випадково. Увімкнення або вимкнення режиму налагодження (DEBUG) також залежить від змінної середовища. У лістингу 3.11 наведено фрагмент коду з режимом налагодження, та секретним ключем.

Лістинг 3.11 – Отримання секретного ключа та режиму налагодження

```
SECRET_KEY = getenv('DJANGO_SECRET_KEY', get_random_secret_key())
DEBUG = getenv('DJANGO_DEBUG', 'False') == 'True'
```

У цьому ж блоці вказано ALLOWED_HOSTS, тобто список хостів, з яких дозволяється звертатися до застосунку. У лістингу 3.12 наведено фрагмент коду для отримання списку хостів.

Лістинг 3.12 – Завантаження списку хостів

```
ALLOWED_HOSTS = ['*']
```

Це значення використовується переважно на етапі розробки – у продакшн-режимі слід вказати конкретні доменні імена або IP-адреси, щоб уникнути потенційних загроз безпеці.

Наступний логічний блок – це перелік встановлених додатків у проєкті, які визначають основний функціонал платформи. Django дозволяє розділяти код на

окремі модулі або "додатки", кожен з яких виконує конкретне завдання (наприклад, обробка користувачів, керування курсами, огляд сертифікатів тощо). У лістингу 3.13 наведено список модулів проєкту.

Лістинг 3.13 – Список модулів проєкту, задіяні в роботі

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django_bigbluebutton',
    'corsheaders',
    'rest_framework',
    'drf_yasg',
    'djoser',
    'social_django',
    'users',
    'bank',
    'commands',
    'languages',
    'specializations',
    'usersconfig',
    'review',
    'course',
    'certificate'
]
```

Системні додатки django.contrib.* – це вбудовані модулі Django, які забезпечують базову функціональність: адміністрування, автентифікація, сесії, повідомлення та статику. Інші сторонні бібліотеки:

– django_bigbluebutton – інтеграція з системою онлайн-конференцій BigBlueButton для проведення відеозанять;

– corsheaders – обробка CORS-запитів, потрібна для взаємодії з фронтендом на React;

– rest_framework – REST API-фреймворк, основа для побудови API-запитів;

– social_django – інтеграція авторизації через Google та Facebook.

Кастомні додатки проєкту:

– users – логіка користувачів та профілів;
– bank – функціональність платіжної системи;
– commands – окремі команди керування;
– languages, specializations, usersconfig – модулі конфігурацій;
– review – огляди курсів чи викладачів;
– course – логіка побудови курсів;
– certificate – генерація та обробка сертифікатів.

Ці модулі утворюють загальну структуру платформи, забезпечуючи поділ на відповідальні компоненти, що підвищує зручність підтримки і розширення системи.

Наступним важливим елементом є middleware – проміжне програмне забезпечення, яке обробляє запити та відповіді у процесі взаємодії клієнта з сервером. Middleware в Django – це стек функцій, що виконуються при кожному HTTP-запиті до сервера. Вони дозволяють реалізувати такі речі, як безпека, управління сесіями, обробка помилок та підтримка автентифікації. У лістингу 3.14 наведено список міделверів.

Лістинг 3.14 – Список міделверів, які завантажуються з проєктом

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'tutorspace.error_handler_middleware.ErrorHandlingMiddleware',
]
```

У цьому списку SecurityMiddleware відповідає за базовий захист запитів, наприклад за налаштування заголовків для підвищення безпеки. SessionMiddleware зберігає дані про сесію користувача між запитами. CorsMiddleware додається для підтримки запитів з інших доменів – необхідно для коректної роботи з фронтендом на React, що запускається окремо.

CommonMiddleware та CsrfViewMiddleware – це вбудовані засоби Django для обробки типових HTTP-викликів і захисту від CSRF-атак. AuthenticationMiddleware забезпечує доступ до інформації про користувача у запитах, а MessageMiddleware керує flash-повідомленнями. XFrameOptionsMiddleware блокує вбудовування сайту в iframe з інших джерел, захищаючи від атак типу clickjacking.

Останнім у списку є кастомний middleware ErrorHandlingMiddleware, який, був реалізований спеціально для перехоплення і обробки помилок у зручному форматі для API – наприклад, щоб повертати стандартну структуру JSON-відповідей з повідомленням про помилку замість HTML-сторінки.

Далі у файлі settings.py описується конфігурація URL-роутінгу та шаблонів, що дозволяє керувати шляхами доступу до сторінок і логікою рендерингу, якщо проєкт має виводити HTML-сторінки.

Змінна ROOT_URLCONF вказує на модуль, де знаходиться головний файл маршрутизації URL – у даному випадку це tutorspace.urls. Django при обробці кожного запиту шукає відповідний шлях саме в цьому файлі.

Секція TEMPLATES описує механізм рендерингу HTML-шаблонів. Встановлено бекенд DjangoTemplates, який дозволяє працювати з .html файлами, розташованими у templates-каталогах кожного додатку. Параметр APP_DIRS=True активує автоматичний пошук шаблонів у структурах застосунків. context_processors – це функції, що додають змінні до контексту шаблону, такі як інформація про користувача, повідомлення або поточний запит. У лістингу 3.15 сформовано конфігурацію роунтингу та шаблонів.

Лістинг 3.15 – Основний файл для роутингу та конфігурація шаблонів

```
ROOT_URLCONF = 'tutorspace.urls'

TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
```

```
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Далі вказується точка входу до WSGI-сервера. Цей параметр вказує, який об'єкт WSGI використовує сервер для запуску проєкту. Це стандартна вимога для деплойменту Django-проєктів на сервері, який підтримує WSGI, наприклад Gunicorn або uWSGI. У лістингу 3.16 показано точка входу для WSGI-сервера.

Лістинг 3.16 – точка входу WSGI-сервера

```
WSGI_APPLICATION = 'tutorspace.wsgi.application'
```

Наступна секція відповідає за конфігурацію бази даних, яка є критичною частиною кожного Django-проєкту, адже саме тут зберігаються всі дані: користувачі, курси, сертифікати, запити тощо. У даній конфігурації використовується драйвер PostgreSQL ('ENGINE': 'django.db.backends.postgresql'), що є потужним і рекомендованим варіантом для продакшн-середовищ Django. Ім’я бази (NAME), користувач (USER), пароль (PASSWORD) та хост (HOST) – стандартні параметри для з’єднання з базою. Значення HOST = 'db' вказує, що база даних працює як окремий сервіс у Docker-контейнері з назвою db.

Опція ATOMIC_REQUESTS = True означає, що кожен HTTP-запит буде обгорнутий у транзакцію. Це підвищує надійність, оскільки будь-які помилки в ході запиту автоматично скасовують зміни в БД, що дозволяє уникнути частково збережених даних. У лістингу 3.17 зображено повну конфігурацію бази даних з django.

Лістинг 3.17 – Конфігурація бази даних

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'postgres',
        'USER': 'test',
        'PASSWORD': 'qa123123',
        'HOST': 'db',
        'PORT': '5432',
        'ATOMIC_REQUESTS': True
    }
}
```

Далі налаштовується надсилання електронної пошти через Amazon SES. У цьому фрагменті налаштовано інтеграцію з Amazon Simple Email Service (SES) – це сервіс для надсилання email-повідомлень. Django використовує власний бекенд для SES (django_ses.SESBackend), а ключі та параметри конфігурації завантажуються з .env файлу через getenv. Це дозволяє безпечно управляти обліковими даними, не зберігаючи їх напряму в коді. Змінна USE_SES_V2 = True вказує на використання актуальної версії API Amazon SES.

У лістингу 3.18 сформовано детальну конфігурацію для надсилання емейлів використовуючи інтеграцію Amazon SES.

Лістинг 3.18 – Конфігурація надсилання емейлів

```
EMAIL_BACKEND = 'django_ses.SESBackend'
DEFAULT_FROM_EMAIL = getenv('DJANGO_AWS_SES_FROM_EMAIL')

AWS_SES_ACCESS_KEY_ID = getenv('DJANGO_AWS_SES_ACCESS_KEY_ID')
AWS_SES_SECRET_ACCESS_KEY =
getenv('DJANGO_AWS_SES_SECRET_ACCESS_KEY')
AWS_SES_REGION_NAME = getenv('DJANGO_AWS_SES_REGION_NAME')
AWS_SES_REGION_ENDPOINT =
f'email.{AWS_SES_REGION_NAME}.amazonaws.com'
AWS_SES_FROM_EMAIL = getenv('DJANGO_AWS_SES_FROM_EMAIL')
USE_SES_V2 = True
```

Далі описується налаштування безпеки паролів, міжнародної локалізації, а також автентифікації через JWT та соціальні платформи. У лістингу 3.19 зображено реалізацію валідаторів паролів.

Лістинг 3.19 – Валідатори паролів

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityVa
lidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator'
,
    },
]
```

Ці валідатори забезпечують базові вимоги до складності паролів: мінімальна довжина, відсутність занадто простих чи поширених комбінацій, а також заборона лише числових паролів. Це сприяє покращенню захисту користувацьких акаунтів від злому. У лістингу 3.20 формують базові часові налаштування.

Лістинг 3.20 – Локалізація та час

```
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_TZ = True
```

Ці параметри задають базові налаштування мови та часового поясу проєкту. У цьому випадку обрано англійську мову та час UTC. Увімкнено інтернаціоналізацію (USE_I18N) та підтримку часових зон (USE_TZ), що особливо важливо при роботі з користувачами з різних країн. У лістингу 3.21 зображено налаштування статичної та медіа директорій.

Лістинг 3.21 – Статичні та медіафайли

```
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

Налаштування STATIC_URL та STATIC_ROOT визначають шлях до статичних файлів (CSS, JS, зображення), які потрібні для відображення інтерфейсу. MEDIA_URL та MEDIA_ROOT – для завантажених користувачем файлів. Це дозволяє зберігати та обробляти медіа окремо від коду. У лістингу 3.22 показано реалізацію автентифікації.

Лістинг 3.22 – Конфігурація автентифікації

```
AUTHENTICATION_BACKENDS = [
    'social_core.backends.google.GoogleOAuth2',
    'social_core.backends.facebook.FacebookOAuth2',
    'django.contrib.auth.backends.ModelBackend',
]
```

Тут вказані бекенди автентифікації: стандартна автентифікація Django (ModelBackend), а також соціальні – Google та Facebook через бібліотеку python-social-auth. Це дозволяє користувачам входити до системи через свої облікові записи в Google або Facebook.

Наступна частина коду коду відповідає за Django REST Framework, а він є основою для побудови API. Встановлено кастомну JWT-автентифікацію та дозвіл доступу лише для автентифікованих користувачів за замовчуванням. У лістингу 3.23 записано конфігурації для основи формування REST запитів.

Лістинг 3.23 – Основа API

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'users.authentication.CustomJWTAuthentication',
    ],
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
    ]
}
```

Для забезпечення безпечного трафіку запитів потрібно подбати про захист даних, які циркулюють в системі. Тому було використано базову систему захисту JSON Web Token. JWT дозволяє реалізувати безпечну авторизацію. Токен доступу живе 15 хвилин, оновлення – 7 днів. Користувачі отримують токени у відповідь на логін, і можуть використовувати їх для доступу до захищених ресурсів. Юзер буде отримувати два ключі, які будуть передаватися в cookie для надійного захисту. Якщо час ключу мине, відбудеться запит на отримання нового ключу. У лістингу 3.24 показано реалізацію конфігурації для токенів.

Лістинг 3.24 – Конфігурація ключів доступу

```
SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=15),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=7),
    'ROTATE_REFRESH_TOKENS': False,
    'BLACKLIST_AFTER_ROTATION': True,
    'AUTH_HEADER_TYPES': ('Bearer',),
}
```

Наступні налаштування файлу settings.py, стосуються реєстрації користувачів, автентифікації через сторонні сервіси, міждоменної взаємодії, налаштувань кастомної моделі користувача та інтеграції з BigBlueButton.

Djoser – це бібліотека, що полегшує реалізацію аутентифікації через Django REST Framework. Вона використовується для реєстрації, входу в систему, підтвердження акаунту та скидання паролю. Вказані URL-шаблони для цих дій. Також налаштовані кастомні серіалізатори для створення користувача та отримання поточного. У лістингу 3.25 сформовано детальний конфіг для менеджменту даних користувача.

Лістинг 3.25 – Налаштування для Djoser

```
DJOSER = {
    'PASSWORD_RESET_CONFIRM_URL': 'password-reset/{uid}/{token}',
    'SEND_ACTIVATION_EMAIL': False,
    'ACTIVATION_URL': 'activation/{uid}/{token}',
    'USER_CREATE_PASSWORD_RETYPE': True,
    'PASSWORD_RESET_CONFIRM_RETYPE': True,
    'TOKEN_MODEL': None,
```

```
    'SOCIAL_AUTH_ALLOWED_REDIRECT_URIS': None,
    'SERIALIZERS': {
        'user_create_password_retype':
'users.serializers.userCreateSerializer.UserAccountCreateSerialize
r',
        'user_create':
'users.serializers.UserAccountCreateSerializer',
        'current_user': 'users.serializers.UserAccountSerializer',
    }
}
```

Наступна частина коду відповідає за соціальну автентифікацію Google. Цей блок налаштовує автентифікацію через Google OAuth2. Він забезпечує отримання базової інформації користувача (email, ім’я, прізвище) та дозволяє легко реєструвати або входити у систему через Google акаунт. Вказано, які дані потрібно отримувати при вході, щоб забезпечити створення або авторизацію користувача у системі. У лістингу 3.26 показано налаштування для обох соціальної автентифікації.

Лістинг 3.26 – Конфіг для Google OAuth2 та Facebook

```
SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = getenv('GOOGLE_AUTH_KEY')
SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET =
getenv('GOOGLE_AUTH_SECRET_KEY')
SOCIAL_AUTH_GOOGLE_OAUTH2_SCOPE = [
    'https://www.googleapis.com/auth/userinfo.email',
    'https://www.googleapis.com/auth/userinfo.profile',
    'openid'
]
SOCIAL_AUTH_GOOGLE_OAUTH2_EXTRA_DATA = ['first_name', 'last_name']
SOCIAL_AUTH_FACEBOOK_KEY = getenv('FACEBOOK_AUTH_KEY')
SOCIAL_AUTH_FACEBOOK_SECRET = getenv('FACEBOOK_AUTH_SECRET_KEY')
SOCIAL_AUTH_FACEBOOK_SCOPE = ['email']
SOCIAL_AUTH_FACEBOOK_PROFILE_EXTRA_PARAMS = {
    'fields': 'email, first_name, last_name'
}
```

Наступний блоку коду відповідає за CORS, кастомну модель користувача та базові налаштування для користування BigBlueButton. Зображені параметри в наступному лістингу дозволяють клієнтським застосункам (наприклад, React або мобільним додаткам) взаємодіяти з API на іншому домені. Параметр CORS_ALLOW_CREDENTIALS = True дає змогу передавати cookie з

авторизаційними токенами під час запитів. Зображений рядок AUTH_USER_MODEL вказує Django використовувати власну модель користувача (UserAccount), замість стандартної User. Це дозволяє розширити структуру користувачів новими полями, наприклад, додати аватар, країну, роль користувача, тощо. Інтеграція з BigBlueButton – це один із ключових функціоналів платформи, що забезпечує проведення відеозустрічей, лекцій, вебінарів тощо. BBB_API_URL – це базовий шлях до API сервісу, а BBB_SECRET_KEY використовується для підпису запитів, що гарантує їхню безпеку. У лістингу 3.27 наведено конфігурації для, кастомної моделі користувача, налаштування взаємодії між бекендом та фронтендом, основне налаштування для BigBlueButton.

Лістинг 3.27 – Налаштування відео зв’язку, моделі користувача та CORS

```
CORS_ALLOWED_ORIGINS = getenv('DJANGO_CORS_ALLOWED_ORIGINS',
'http://localhost:3000,http://127.0.0.1:3000').split(',')
CORS_ALLOW_CREDENTIALS = True
AUTH_USER_MODEL = 'users.UserAccount'
BBB_API_URL = 'https://test.com/bigbluebutton/api/'
BBB_SECRET_KEY = 'abcdefgabcdefgabcdefgabcdefgabcdefg'
```

Файл settings.py проєкту TutorSpace відіграє ключову роль у конфігурації всієї платформи, забезпечуючи гнучкість, масштабованість та безпеку. Його структура охоплює всі критично важливі аспекти сучасного веб-застосунку: від інтеграції з зовнішніми сервісами до підтримки REST API, автентифікації, поштових сервісів, обробки CORS-запитів, взаємодії з базою даних та управління середовищем розгортання.

Основна ідея реалізації полягає у поділі функціоналу на незалежні, але взаємопов’язані компоненти, з чітко визначеною відповідальністю: кожен модуль (курс, користувачі, відгуки, сертифікати тощо) підключений окремо, але взаємодіє з іншими через REST API, JWT, соціальні логіни та інші механізми. Такий підхід дозволяє розробляти систему за модульною архітектурою з можливістю подальшого масштабування, рефакторингу або заміни частин функціоналу.

Окремої уваги заслуговує використання бібліотек djoser, drf-yasg, social-auth, а також інтеграція з сервісом BigBlueButton. Це свідчить про високий рівень технічної реалізації платформи, спрямованої на потреби освітнього середовища.

Налаштування параметрів безпеки, зокрема SECRET_KEY, токенів доступу, валідаторів паролів, доступу до API та міждоменної взаємодії (CORS), демонструють розуміння викликів, пов’язаних з розгортанням системи в публічному середовищі. Також важливою особливістю є підтримка .env.local і використання змінних середовища, що дозволяє відокремити чутливі дані від логіки коду.

Таким чином, settings.py виконує не лише роль конфігураційного файлу, а й є своєрідною основою для архітектури, безпеки та адаптивності навчальної платформи TutorSpace, що відповідає сучасним вимогам до веб-розробки.

У директорії tutorspace зберігаються основні конфігураційні файли Django-проєкту, які забезпечують функціональність та керованість усієї серверної частини застосунку. Файл __init__.py визначає цю директорію як Python-пакет, завдяки чому модулі всередині неї можуть імпортуватися у проєкті. Хоча найчастіше він залишається порожнім, його присутність критично важлива для коректної роботи всієї структури.

Файл asgi.py виконує роль точки входу для ASGI-сервера (Asynchronous Server Gateway Interface), який використовується для асинхронного обслуговування запитів. Цей файл особливо важливий, якщо платформа повинна підтримувати WebSocket, стрімінг або асинхронні операції – наприклад, підключення відеоконференцій або чатів у реальному часі.

Файл error_handler_middleware.py реалізує власний middleware, який перехоплює помилки, що виникають під час HTTP-запитів, і формує стандартизовану відповідь у форматі JSON. Це забезпечує стабільну структуру відповідей API, незалежно від того, яка помилка виникла – це критично для фронтенду, особливо у випадках, коли платформа інтегрується з мобільним застосунком або React-інтерфейсом.

Файл urls.py відповідає за маршрутизацію – тобто, визначає, який URL-шлях веде до якого функціонального блоку застосунку. У цьому файлі

імпортуються маршрути з усіх застосунків платформи, а також конфігуруються адміністративний інтерфейс, Swagger-документація, API та інші ендпоінти.

Спершу здійснюється імпорт необхідних модулів. Це стандартні компоненти Django для маршрутизації (path, include, re_path, admin), конфігураційні дані проєкту (settings), а також бібліотеки drf_yasg, що використовуються для побудови OpenAPI-документації. На лістингу 3.28 показано фрагмент імпорту всіх необхідних залежностей.

Лістинг 3.28 – Імпорт бібліотек для маршрутизації та документації

```
from django.contrib import admin
from django.urls import path, re_path, include
from rest_framework import permissions
from drf_yasg.views import get_schema_view
from drf_yasg import openapi
from django.conf import settings
from django.conf.urls.static import static
```

Наступний етап – конфігурація Swagger-документації API. Це важлива складова сучасних REST-застосунків, яка дає можливість користувачам або іншим розробникам бачити всі доступні ендпоінти, параметри запитів та приклади відповідей. Для цього використовується об'єкт schema_view, який створюється за допомогою функції get_schema_view. У лістингу 3.29 наведено приклад ініціалізації цього об'єкта.

Лістинг 3.29 – Налаштування Swagger-документації

```
schema_view = get_schema_view(
    openapi.Info(
        title="TutorSpace API",
        default_version="v1",
        description="TutorSpace API documentation",
        terms_of_service="https://www.test.com/terms/",
        contact=openapi.Contact(email="test@test.com"),
        license=openapi.License(name="MIT License"),
    ),
    public=True,
    permission_classes=[permissions.AllowAny],
)
```

Основна логіка маршрутизації представлена у списку urlpatterns, який включає всі шляхи до частин платформи. Сюди входять як стандартна панель

адміністратора (admin/), так і окремі підмодулі системи, наприклад: users, course, review, certificate тощо. Також у цей список інтегровані шляхи до сторонніх бібліотек, зокрема djoser для автентифікації, та django_bigbluebutton для відеоконференцій. У лістингу 3.30 наведено повну структуру маршрутів.

Лістинг 3.30 – Основні маршрути API-платформи

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('tutor/', include('django_bigbluebutton.urls')),
    path('api/', include('djoser.urls')),
    path('api/', include('users.urls')),
    path('swagger/', schema_view.with_ui('swagger',
cache_timeout=0), name='schema-swagger-ui'),
    path('redoc/', schema_view.with_ui('redoc', cache_timeout=0),
name='schema-redoc'),
    re_path(r'^swagger(?P<format>\.json|\.yaml)$',
schema_view.without_ui(cache_timeout=0), name='schema-json'),
    path('api/languages/', include('languages.urls')),
    path('api/specializations/', include('specializations.urls')),
    path('api/bank-cards/', include('bank.urls')),
    path('api/review/', include('review.urls')),
    path('api/course/', include('course.urls')),
    path('api/certificate/', include('certificate.urls')),
]
```

Окремим блоком до urlpatterns додаються конфігурації для обробки статичних і медіафайлів. Це потрібно для коректного завантаження зображень, PDF-документів, аватарів користувачів або сертифікатів, які є частиною функціоналу платформи. Такий підхід дозволяє виводити медіа навіть під час розробки без налаштування окремого веб-сервера. На лістингу 3.31 зображено обробку статичних і медіафайлів.

Лістинг 3.31 – Налаштування доступу до медіа та статичних файлів

```
urlpatterns+=static(settings.MEDIA_URL,document_root=settings.MEDI
A_ROOT)
urlpatterns+=static(settings.STATIC_URL,document_root=settings.STA
TIC_ROOT)
```

Таким чином, urls.py відіграє критичну роль у забезпеченні взаємодії між зовнішніми клієнтами та логікою платформи, централізуючи всі точки доступу до сервісів. Завдяки підтримці OpenAPI-документації через Swagger і ReDoc, а

також правильному структуруванню маршрутів – реалізовано гнучку, масштабовану та зручну у підтримці архітектуру веб-застосунку.

Файл wsgi.py використовується як точка входу до застосунку при запуску через WSGI-сервер, наприклад Gunicorn. Цей серверний інтерфейс працює у синхронному режимі і є стандартом для розгортання Django у production-середовищі. Він ініціалізує застосунок і передає запити серверу, що забезпечує стабільну роботу навіть при високому навантаженні.

Усі ці файли в сукупності формують конфігураційне ядро серверної частини платформи TutorSpace, забезпечуючи її масштабованість, безпеку та готовність до продакшну.

3.3 Демонстрація розробленої платформи

Заключним етапом у розробці навчальної платформи слугують альфа- та бета-тестування, а також демонстрація продукту. Цей момент є чи не найважливішим у циклі життя продукту, оскільки від зворотного зв’язку користувачів залежить подальший розвиток платформи. На рисунку 3.4 зображено сторінку реєстрації на платформу. На цій сторінці видно що при реєстрації потрібно ввести такі дані як пошта, ім’я, прізвище, пароль та його підтвердження.

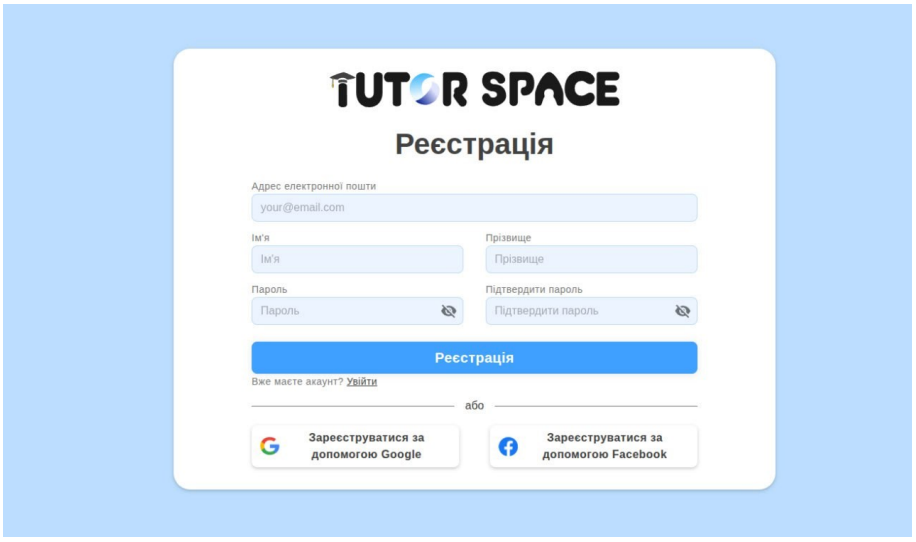

Рисунок 3.4 – Сторінка реєстрації

Крім того користувач може зареєструватися через облікові записи Google або Facebook, а також перейти на сторінку входу, якщо акаунт вже створено.

Після реєстрації у користувача з’являється сторінка, яка повідомляє про те що авторизація пройшла успішно. Клієнту пропонується почекати доки відбудеться перенаправлення на домашню сторінку або натиснути на посилання у випадку технічних неполадок. Також до уваги користувача представлено можливість ознайомитися із соціальними мережами платформи. Ці всі процеси зображено на рисунку 3.5.

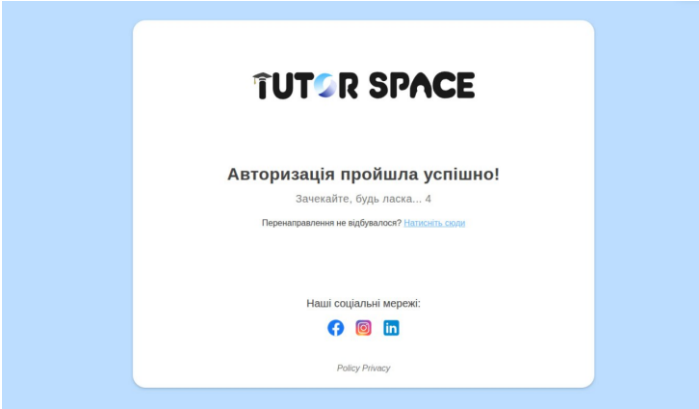

Рисунок 3.5 – Сторінка повідомлення про успішну авторизацію

Не зважаючи на безліч порад як безпечно можна зберігати паролі, ніхто не застрахований від втрати такої важливої інформації. Тому було створено сторінки для відновлення паролю, які зображені на рисунку 3.6.

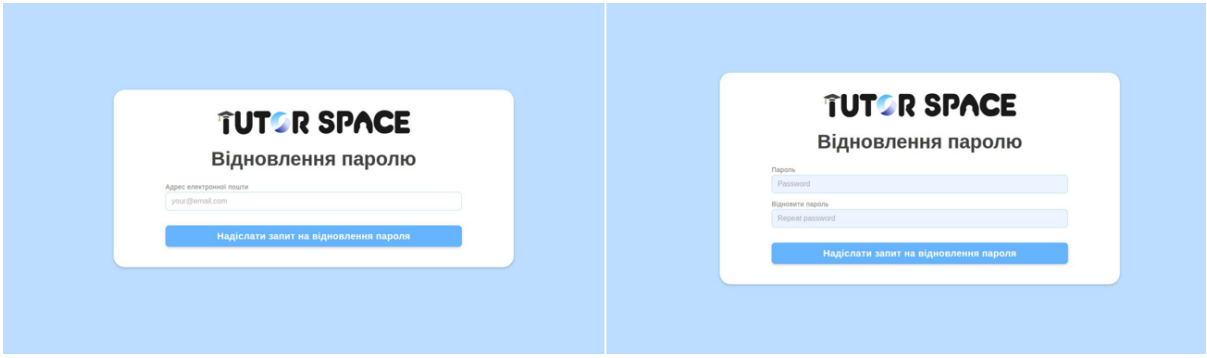

Рисунок 3.5 – Сторінка відновлення паролю

На рисунку 3.6 зображено сторінку налаштування профілю, а саме вкладка особистої інформації. Ім’я, прізвище, пошта підтягуються, оскільки були введені при реєстрації. Користувач може ввести такі дані як номер телефону, додатку інформацію про стать, день народження та посилання на соціальні мережі. Також у користувача є можливість приховати деякі дані.

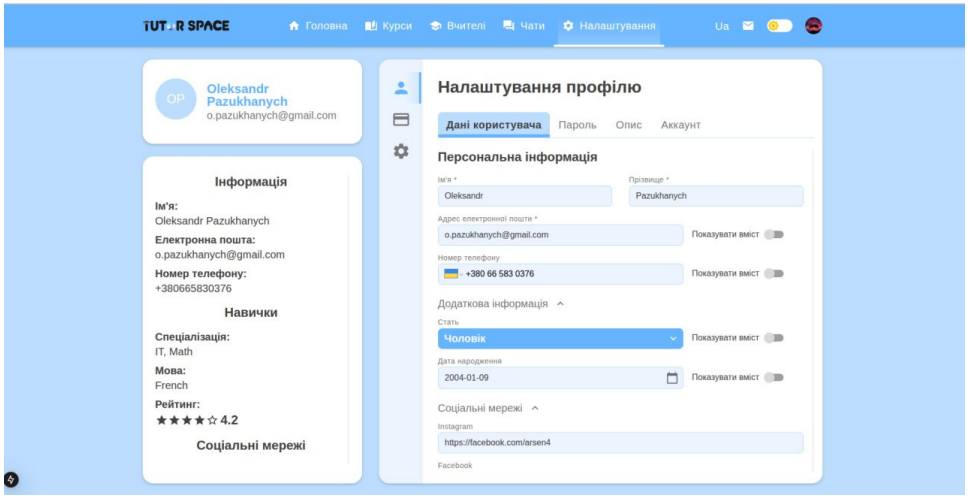

Рисунок 3.6 – Сторінка із персональними даними

На рисунку 3.7 зображено вкладку для зміни паролю. Для виконання цієї дії потрібно ввести теперішній пароль, новий, а також його повторити. До кожної вкладки прикріплена окрема кнопка «Зберегти зміни», а також модальне вікно – попередження про не збережені зміни. Така система дозволяє не втратити дані при переключенні або випадкового закриття сторінки.

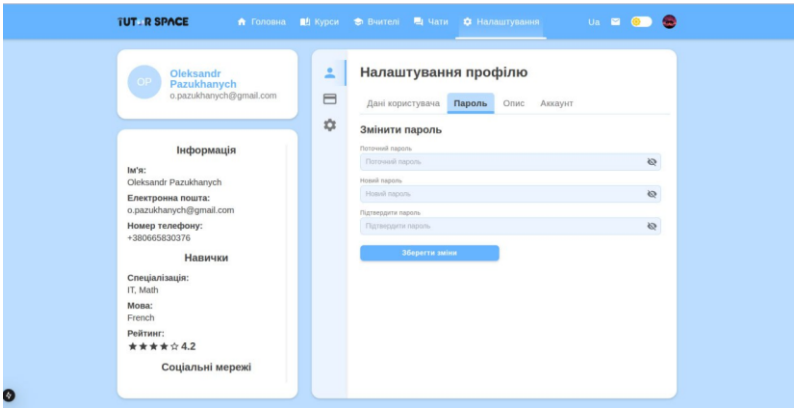

Рисунок 3.7 – Сторінка зміни паролю

На рисунку 3.8 зображено вкладку опису, у якій міститься інформація про напрямок, який викладач буде вести, чи студент вивчати, мовні навички – для вибору комфортної мови навчання, а також про мене – поле у якому можна ввести додаткову інформацію, розказати про себе. Це поле містить редактор, тому інформацію можна структурувати та відредагувати до такого стилю, якого хоче користувач.

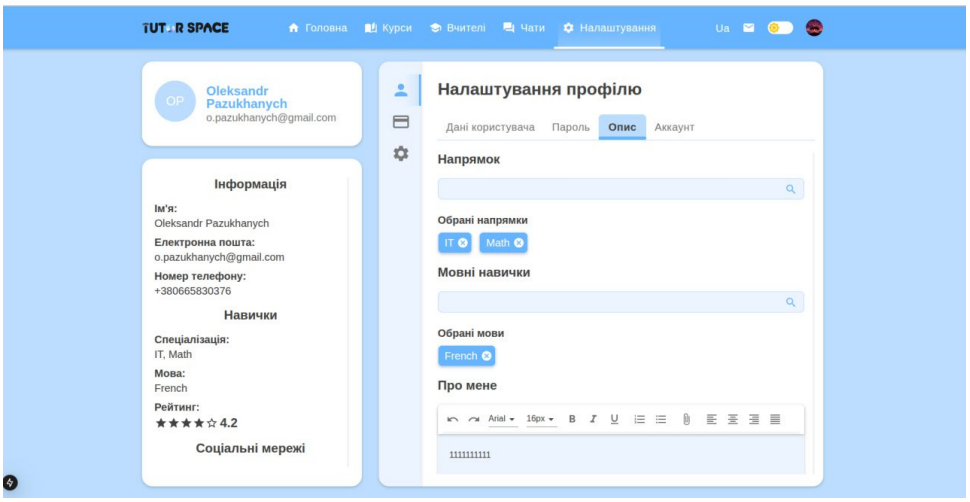

Рисунок 3.8 – Сторінка опису

На рисунку 3.9 зображено вкладку стану акаунту. Тут користувачу пропонується деактивувати акаунт, тобто вилучити його з пошуку вчителів/учнів, а також видалити обліковий запис. Зі сторони викладача пропонується вибрати спосіб оплати до або після заняття.

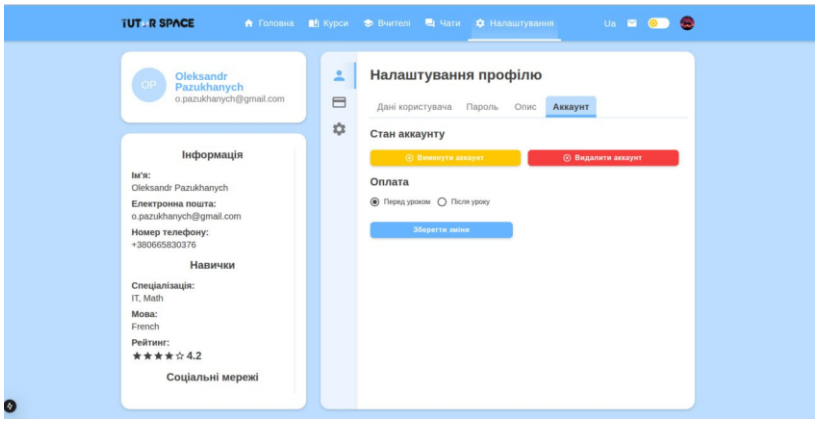

Рисунок 3.9 – Сторінка стану акаунту

У процесі реалізації платформа та її функціональні можливості будуть оновлюватися та вдосконалюватися для досягнення статусу однієї із найкромфортніших існуючих навчальних платформ.

3.4 Висновок до третього розділу

У третьому розділі кваліфікаційної роботи було розглянуто реалізацію веб-платформи TutorSpace, що поєднує сучасні технології фронтенду та бекенду. Фронтенд частина побудована з використанням React, архітектурного підходу Feature-Sliced Design, типізації на основі TypeScript та бібліотек React Hook Form і Zod для роботи з формами. Для побудови інтерфейсу застосовано Material UI, що забезпечило адаптивність і візуальну узгодженість. Для керування станом застосунку було використано Redux Toolkit та RTK Query, що дозволило централізувати логіку обміну даними з сервером. Забезпечено підтримку багатомовності через i18next, а також реалізовано middleware для перевірки автентифікації користувача при маршрутизації. Завдяки модульності та строгій типізації досягнуто високої надійності та масштабованості клієнтської частини.

Бекенд частина реалізована за допомогою Django та Django REST Framework, із впровадженням механізмів JWT-автентифікації, соціального входу (Google, Facebook), та автоматичної документації API через Swagger. Налаштовано роботу з базою даних PostgreSQL, статичними файлами, поштовим сервісом Amazon SES та відеоконференціями через BigBlueButton. Використання .env-файлів забезпечило безпечне зберігання конфіденційних даних.

Загалом, реалізація платформи TutorSpace підтвердила доцільність застосування сучасних технологій та архітектурних рішень у створенні веб-застосунків. Досягнуто основної мети — розробки функціональної, надійної та масштабованої системи, яка відповідає актуальним вимогам галузі інформаційних технологій.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

У цьому розділі буде розглянуто питання безпеки життєдіяльності та охорони праці в умовах професійної діяльності розробників веб-застосунків. Особливу увагу приділено організації робочого місця відповідно до ергономічних принципів.

Також буде ознайомлено із основними психофізіологічними ризиками, які пов’язані з тривалою роботою програмістів і технічних спеціалістів за комп’ютером та ефективні методи мінімізації цих ризиків.

4.1 Інтеграція принципів ергономіки та охорони праці в роботу розробника

У професійній діяльності розробника програмного забезпечення питання ергономіки та охорони праці є не менш важливими, ніж технічна компетентність. Постійна робота з комп’ютером, багатогодинна концентрація, тривале статичне сидіння, інтенсивне зорове навантаження та психоемоційна напруга створюють передумови для розвитку низки професійних захворювань. Саме тому особливу увагу слід приділяти організації робочого місця з урахуванням принципів ергономіки, викладених у дослідженнях з людських факторів, зокрема у Human Factors in Engineering and Design [72].

Правильно організоване робоче місце програміста повинно враховувати декілька базових параметрів: висоту і положення монітора, налаштування клавіатури і миші, тип сидіння та положення тіла. Монітор повинен знаходитися на відстані 50–70 см від очей, його верхня межа – на рівні або трохи нижче рівня очей. Кут огляду монітора має бути нахиленим на 10–20°, що знижує напруження шийного відділу хребта [73].

Крісло розробника повинно мати регульовану висоту, підтримку попереку, зручні підлокітники та поворотну основу. Правильне положення тіла (коліна під

кутом 90°, стопи на підлозі, руки – паралельно до клавіатури) допомагає зменшити ризик розвитку тунельного синдрому, болей у спині та шиї. Важливо також чергувати періоди роботи з короткими активними перервами кожні 45–60 хвилин, щоб запобігти стомленню м’язів і нервової системи [74].

На рисунку 4.1 зображено правильне розміщення користувача у роботі з комп’ютером.

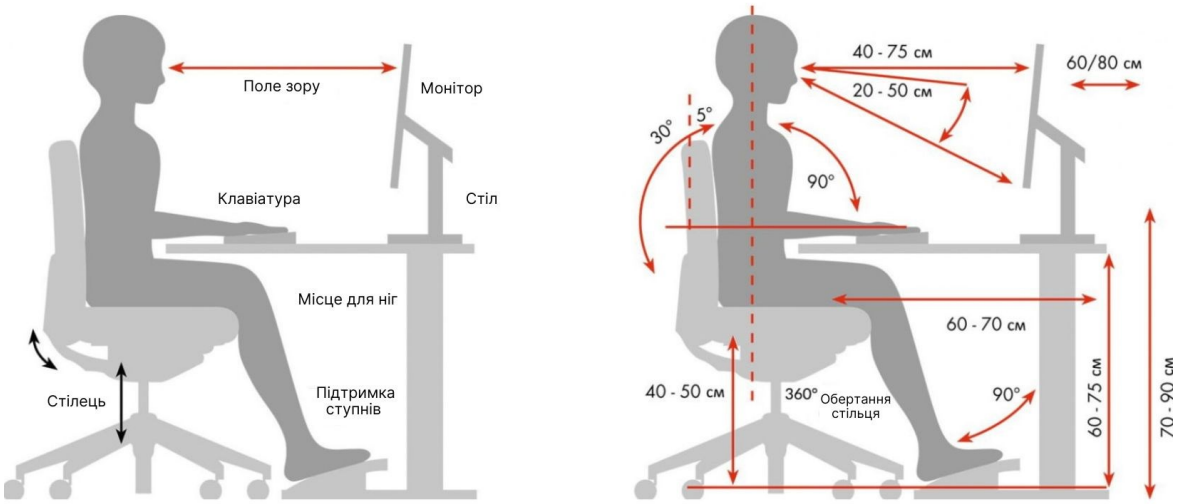

Рисунок 4.1 – Правильна позиція для сидячої роботи за комп’ютером [75]

Освітлення робочого простору має бути рівномірним, без відблисків на екрані. Рекомендується використовувати комбіноване освітлення: природне світло вдень у поєднанні з локальним штучним освітленням у вечірній час. Джерела світла не повинні спрямовуватись безпосередньо на монітор або в очі користувача. Водночас, налаштування яскравості екрана та включення нічного режиму (або фільтра синього світла) допомагає зменшити навантаження на зір.

Ще одним важливим фактором є оптимізація простору для багатомоніторної роботи, оскільки розробники часто працюють з кількома екранами (код, документація, браузер). Монітори варто розміщувати так, щоб основний екран знаходився прямо перед користувачем, а допоміжні – під легким кутом, не змінюючи значно положення голови.

Для профілактики втомлюваності важливою є не лише фізична ергономіка, але й організаційна. Розробникам слід користуватись тайм-менеджментом (наприклад, техніка Pomodoro), використовувати програми для нагадування про перерви (Stretchly, Workrave), планувати мікропаузи на дихальні вправи або легку гімнастику.

Дотримання цих принципів сприяє зниженню ризиків виникнення хронічних розладів, збереженню продуктивності та концентрації під час роботи з інтенсивними проєктами – зокрема в середовищі створення складних інформаційних систем на базі Django, React і PostgreSQL. Раціональна організація простору, грамотне навантаження зору і тіла, а також психофізіологічне розвантаження стають важливою складовою сучасної професії програміста, що працює у гнучкому цифровому середовищі.

4.2 Психофізіологічні ризики при тривалій роботі за комп'ютером та шляхи їх мінімізації

У професійній діяльності розробника, зокрема фахівців, які працюють із Django, React і PostgreSQL, важливе місце займає тривала зосереджена робота за комп'ютером. Такий режим створює ризики для психофізіологічного здоров'я. Це стосується не лише фізичних навантажень на зір і опорно-руховий апарат, але й психоемоційного стану, що значною мірою впливає на продуктивність і загальний добробут спеціаліста [76].

Одним із ключових ризиків є цифрова зорова напруга (Computer Vision Syndrome, CVS), яка виникає через тривалу фіксацію погляду на екран, рідке моргання, недостатню яскравість або некоректні налаштування монітора. Прояви CVS включають сухість очей, головний біль, тимчасове погіршення зору, напруження в області лоба. Щоб запобігти цьому, розробникам рекомендується використовувати техніку «20-20-20»: кожні 20 хвилин переводити погляд на об'єкт, розташований на 20 футах (близько 6 метрів) і тримати фокус на ньому щонайменше 20 секунд. Також важливе значення має правильне освітлення в

робочій зоні та використання режимів "ночі" або зменшення синього світла [77]. На рисунку 4.2 зображена техніка «20-20-20».

Рисунок 4.2 – Техніка «20-20-20»[78]

Ще один поширений ризик – порушення в роботі опорно-рухового апарату. Тривале перебування в сидячому положенні, незручна постава, відсутність підтримки спини або погане розташування рук на клавіатурі й миші можуть призводити до болю у попереку, шиї, плечах, а також розвитку тунельного синдрому зап’ястя. Для зниження навантаження на тіло розробникам рекомендовано використовувати ергономічні крісла з регульованою висотою, стільниці з відповідною висотою, а також спеціальні підставки для ноутбуків або моніторів, щоб уникати нахилів голови. Окрім цього, періодична зміна положення тіла (сидяча/стояча поза), короткі фізичні вправи кожні кілька годин, а також розминки для шиї та кистей допомагають зменшити ризики фізичної втоми [79].

Варто також зосередити увагу на психоемоційних ризиках. Високе навантаження, жорсткі дедлайни, багатозадачність, технічні проблеми, нестабільні середовища розгортання – усе це сприяє розвитку тривалого стресу, а в перспективі – синдрому професійного вигорання. Серед проявів: хронічна втома, зниження мотивації, дратівливість, проблеми з концентрацією. Для

профілактики таких станів рекомендується впроваджувати практики тайм-менеджменту, наприклад Pomodoro-метод, планування завдань з пріоритетами, використання таймерів на перерви, а також ведення щоденника або рефлексії щодо навантаження. Корисним також є впровадження періодів "цифрового детоксу", зменшення часу в екранах після завершення робочого дня та наявність гнучкого графіку роботи.

Компаніям рекомендується забезпечувати здорове мікросередовище – від організації зон відпочинку до запровадження політик work-life balance. Забезпечення психологічної підтримки, консультації з корпоративними психологами або доступ до ресурсів ментального здоров’я може суттєво знизити тривожність і сприяти стабільній роботі команди.

Таким чином, попри високу інтелектуальну складову діяльності розробника, важливою частиною залишається турбота про фізичне і психоемоційне здоров’я. Систематичне застосування методів профілактики дозволяє підвищити якість роботи, уникнути хронічного стресу та зберегти ефективність у довгостроковій перспективі.

4.3 Відповідальність за дотримання правил охорони праці під час роботи над проєктом у команді

У процесі командної розробки дотримання правил охорони праці є критично важливим аспектом, який безпосередньо впливає як на фізичне та психічне здоров’я працівників, так і на продуктивність команди загалом. Відповідальність за дотримання норм охорони праці розподіляється між усіма учасниками проєкту, проте кожен із них має свій рівень впливу та обов’язків у цій сфері. Насамперед, головну роль у забезпеченні безпеки праці відіграє роботодавець або керівник підприємства. Відповідно до Закону України «Про охорону праці», саме він зобов’язаний створити умови, що відповідають вимогам безпечного виконання професійних обов’язків. У контексті веб-розробки це означає не лише надання якісного програмного та апаратного забезпечення, але

й облаштування робочих місць відповідно до ергономічних стандартів, наявність належного освітлення, вентиляції, а також впровадження регламентів щодо тривалості робочого часу та перерв. Якщо працівники виконують свої обов’язки віддалено, роботодавець має забезпечити їм інструктаж та методичні рекомендації щодо самостійного облаштування домашнього робочого простору.

Ключова роль у дотриманні охорони праці всередині команди покладається на керівника групи розробників або проєктного менеджера. Саме він контролює щоденну діяльність команди, розподіляє завдання, визначає темп виконання роботи та впливає на загальний психологічний клімат у колективі. В його обов’язки входить не лише формальне дотримання норм, але й проактивне виявлення потенційних ризиків для здоров’я співробітників, зокрема – перевтоми, конфліктів, надмірного стресу або зниження мотивації. Успішні керівники впроваджують політику гнучкого графіка, створюють умови для регулярного відпочинку під час робочого дня, використовують практики тайм-менеджменту (наприклад, метод «Помодоро»), обмежують кількість одночасних онлайн-зустрічей, щоб уникнути ефекту «зустрічного вигорання», та забезпечують прозору комунікацію в команді.

Охорона праці – це не лише обов’язок керівництва, а спільна відповідальність усієї команди. Кожен розробник зобов’язаний дотримуватись установлених правил, проходити інструктажі, дбати про власне здоров’я та повідомляти про будь-які ознаки дискомфорту або потенційні ризики. Важливо уникати шкідливих звичок у роботі – зокрема нічних змін без перерв, надмірного навантаження, пропусків обіду та ігнорування болю в очах, руках чи спині, які можуть свідчити про розвиток професійних захворювань.

Таким чином, охорона праці ґрунтується на принципі взаємної відповідальності, а саме на злагодженій участі всіх: адміністрації, яка забезпечує безпечні умови; керівника проєкту, який організовує процеси; і працівників, які дотримуються правил безпеки. Така взаємодія сприяє створенню здорового робочого середовища, покращенню результатів проєктів, зменшенню кадрової плинності та розвитку зрілої корпоративної культури.

4.4 Дії, у разі виникнення аварійної ситуації у приміщенні, де працює команда розробників

Незважаючи на те, що розробка програмного забезпечення зазвичай не пов’язана з використанням небезпечного обладнання або матеріалів, працівники офісних приміщень, зокрема команди IT-фахівців, також можуть опинитися в аварійних ситуаціях. Серед найтиповіших загроз – задимлення або коротке замикання, які можуть призвести до пожежі, ураження електричним струмом чи отруєння чадним газом. В таких випадках надзвичайно важливо діяти швидко, чітко та відповідно до затверджених інструкцій із безпеки. Перш за все, кожен працівник повинен мати чітке уявлення про алгоритм дій у разі виявлення задимлення або запаху горілого. Якщо працівник помітив перші ознаки аварії – дим, характерний запах перегрітого пластику, іскри з електроприладу, – він зобов’язаний негайно повідомити про це керівника групи або відповідальну особу за охорону праці. Однак очікувати вказівок не завжди доречно: якщо ситуація розвивається стрімко, важливо діяти самостійно, але обережно.

Одним із першочергових кроків є вимкнення електроживлення, якщо джерело загрози пов’язане з електроприладом (наприклад, коротке замикання в розетці або серверному обладнанні). Це можна зробити за допомогою вимикача в розподільному щиті або витягнувши вилку з розетки, дотримуючись правил безпеки – лише сухими руками і без дотику до оголених проводів. Якщо є підозра, що загорілась електропроводка, – підходити до джерела без захисних засобів не рекомендується, аби уникнути ураження струмом. У разі появи диму чи відкритого полум’я потрібно негайно організувати евакуацію працівників. Всі члени команди повинні залишити приміщення у чіткій та спокійній послідовності, користуючись основними або запасними виходами згідно з планом евакуації. Категорично заборонено користуватись ліфтами під час пожежі. Якщо в приміщенні щільний дим – слід пересуватись низько, притиснувшись до підлоги, оскільки токсичні гази концентруються у верхніх

шарах повітря. Доцільно прикривати рот і ніс змоченою тканиною для фільтрації повітря. Після виходу з небезпечної зони необхідно повідомити відповідні служби, зателефонувавши на номер 101 (пожежна служба) та за можливості – 103 (медична допомога), якщо є постраждалі. Повідомлення має містити точну адресу, поверх, характер загрози (задимлення, коротке замикання, вибух тощо) і кількість людей, які можуть залишатись у будівлі.

Вкрай важливо не повертатися в приміщення до прибуття рятувальників і офіційного дозволу від відповідальних служб. Керівник команди або відповідальна особа має перевірити, чи всі члени колективу безпечно евакуювалися, та повідомити рятувальникам про можливу наявність людей усередині, якщо когось бракує. У компаніях, де дотримуються правил охорони праці, зазвичай розроблений план евакуації – візуальна схема, розміщена в доступному місці, що показує розташування виходів, вогнегасників, аварійних кнопок вимкнення електропостачання та місце збору після евакуації. Усі працівники повинні бути ознайомлені з цим планом під час вступного інструктажу або спеціальних навчань. Варто також періодично проводити тренувальні евакуації, щоб кожен член команди міг відпрацювати свої дії у надзвичайній ситуації, не піддаючись паніці.

Якщо аварійна ситуація сталася під час роботи команди розробників у гібридному чи віддаленому форматі, і частина фахівців знаходиться в офісі, то керівник має оперативно повідомити всіх учасників про інцидент та тимчасово припинити роботу. Після ліквідації наслідків, у разі потреби, можуть бути проведені психологічні консультації або додаткові заходи щодо відновлення безпечного середовища.

На завершення, варто зазначити, що найважливішим є вміння діяти зібрано і без паніки. Наявність чітко прописаних інструкцій, регулярне навчання персоналу, технічне оснащення приміщення засобами пожежогасіння, справна електромережа та культура безпеки в колективі – усе це дозволяє мінімізувати ризики для життя та здоров’я людей навіть у разі надзвичайної ситуації.

4.5 Висновок до четвертого розділу

У четвертому розділі кваліфікаційної роботи було висвітлено ключові аспекти безпеки життєдіяльності та охорони праці в контексті професійної діяльності веб-розробників. Особливу увагу приділено ергономічним принципам організації робочого місця, які мають безпосередній вплив на фізичне самопочуття, продуктивність та довготривале здоров'я фахівців у сфері ІТ.

Було детально розглянуто оптимальне розміщення обладнання, використання ергономічних меблів, освітлення, важливість правильного положення тіла під час роботи, а також організаційні аспекти, які забезпечують зміну робочих поз і запобігають розвитку професійних захворювань. Це створює безпечне середовище для інтелектуальної праці, знижує ризики м'язової втоми, порушень опорно-рухового апарату та перевантаження зору.

Окремо проаналізовано психофізіологічні ризики, що виникають при тривалій роботі за комп'ютером, включаючи синдром зорового перенапруження, емоційне вигорання, хронічний стрес і когнітивну перевтому. Запропоновано низку превентивних заходів, зокрема: дотримання режиму праці та відпочинку, регулярні активні перерви, вправи для очей (включаючи техніку «20-20-20»), фізична активність, організація акустичного середовища та управління емоційним станом за допомогою тайм-менеджменту та усвідомлених практик.

Розгляд зазначених питань є актуальним для професійної діяльності веб-розробників, які працюють із сучасними інструментами та технологіями, такими як Django, React і PostgreSQL. Підвищення рівня ергономіки робочого місця та психофізіологічного комфорту напряму впливає на якість виконання завдань, дотримання термінів розробки та збереження здоров'я працівників галузі інформаційних технологій.

Загалом, дотримання принципів ергономіки та врахування психофізіологічних потреб розробника не лише підвищує рівень безпеки праці, але й сприяє довготривалій професійній витривалості, якості виконання завдань,

та забезпечує комфортне робоче середовище в умовах високої інтелектуальної навантаженості.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено веб-застосунок TutorSpace — освітню платформу, що поєднує сучасні підходи до побудови клієнтської та серверної частин.

Клієнтська частина побудована із застосуванням архітектурного підходу Feature-Sliced Design, що забезпечило логічну структурованість коду, розділення відповідальностей та зручність у подальшій підтримці та розширенні системи. Всі компоненти були типізовані за допомогою TypeScript, що сприяло підвищенню надійності, зрозумілості та безпеки обміну даними між модулями. Для керування глобальним станом застосунку було обрано Redux Toolkit, що дало змогу ефективно обробляти дані та взаємодіяти з сервером завдяки інтеграції RTK Query. Розробка форм користувацького вводу реалізована з використанням бібліотек React Hook Form та Zod, які забезпечили типізовану валідацію даних та високий рівень продуктивності. В якості бібліотеки компонентів обрано Material UI, яка дозволила реалізувати сучасний інтерфейс, що відповідає принципам адаптивного дизайну та доступності.

Серверна частина реалізована на основі Django та Django REST Framework, що дозволило побудувати надійний API та реалізувати всі необхідні функції аутентифікації, авторизації, керування курсами та користувачами. Для безпечної автентифікації було впроваджено механізм JWT-токенів, зокрема через бібліотеки SimpleJWT та Djoser. Інтеграція з сервісом BigBlueButton надала можливість організовувати відеоконференції, що є важливою функцією платформи.

Для забезпечення багатомовності застосунку було реалізовано інтеграцію з бібліотекою i18next, а персоналізовані налаштування користувача зберігалися за допомогою cookies, з урахуванням вимог до SSR. База даних побудована на основі PostgreSQL, доступ до якої забезпечується через ORM Django.

Таким чином, у ході виконання кваліфікаційної роботи було розроблено повноцінний веб-застосунок, який відповідає сучасним вимогам до програмного

забезпечення в галузі веб-технологій. Отримані результати можуть бути використані для подальших досліджень, вдосконалення функціоналу платформи або як основа для реального впровадження в освітнє середовище.

ПЕРЕЛІК ДЖЕРЕЛ

1 Коваленко Д. Р. Інструментальні засоби мобільного застосунку для організації контролю відвідувань студентів [Електронний ресурс]. Режим доступу: https://ela.kpi.ua/server/api/core/bitstreams/843f7c33-2d3a-4fed-83ed-61549eb0ec30/content (дата звернення: 09.06.2025).

2 ВООЗ оголосила COVID-19 пандемією: що це таке та чим відрізняється від епідемії - світові новини [Електронний ресурс]. Режим доступу: https://tsn.ua/svit/vooz-ogolosila-covid-19-pandemiyeyu-scho-ce-take-ta-chim-vidriznyayetsya-vid-epidemiyi-1506420.html (дата звернення: 09.06.2025).

3 Пасічник В., Кут В. Інформаційні технології та системи дистанційного навчання осіб з особливими потребами [Електронний ресурс]. Режим доступу: https://elartu.tntu.edu.ua/bitstream/123456789/2003/2/TNTUB_2012_v65_No1-Pasechnik_V_Kut_V-Information_technology_and_distance_learning_system_of__127.pdf (дата звернення: 09.06.2025).

4 Биков В. Ю. Сучасні завдання інформатизації освіти [Електронний ресурс]. Режим доступу: https://core.ac.uk/download/pdf/19088460.pdf (дата звернення: 09.06.2025).

5 Кремень В. Концепція розвитку дистанційної освіти в Україні [Електронний ресурс]. Режим доступу: https://vnmu.edu.ua/downloads/other/konc_rov_dystan_osv.pdf (дата звернення: 09.06.2025).

6 Биков В. Ю. Моделі організаційних систем відкритої освіти: монографія [Електронний ресурс]. Режим доступу: https://core.ac.uk/download/32305552.pdf (дата звернення: 09.06.2025).

7 Самойленко О. М. Моделі дистанційної освіти та основні етапи їх розвитку [Електронний ресурс]. Режим доступу: https://scholar.google.com.ua/citations?view_op=view_citation&hl=ru&use

r=cycpzAUAAAAJ&citation_for_view=cycpzAUAAAAJ:RHpTSmoSYBkC (дата звернення: 09.06.2025).

8 Назарко І. Використання засобів дистанційної освіти для підвищення ефективності навчального процесу у ВНЗ [Електронний ресурс]. Режим доступу: https://elartu.tntu.edu.ua/bitstream/123456789/17337/1/konferencija_.pdf (дата звернення: 09.06.2025).

9 Гайдар А., Готович В. Розробка платформи для перевірки знань шляхом тестування [Електронний ресурс]. Режим доступу: https://elartu.tntu.edu.ua/bitstream/lib/37592/2/IMST_2021_Haidar_A-Development_of_platforms_for_37.pdf (дата звернення: 09.06.2025).

10 Про затвердження Положення про дистанційне навчання [Електронний ресурс]. Режим доступу: https://zakon.rada.gov.ua/laws/show/z0703-13#Text (дата звернення: 09.06.2025).

11 Гнатюк О. Дистанційне навчання: проблеми, пошуки, виклики [Електронний ресурс]. Режим доступу: https://lib.iitta.gov.ua/id/eprint/728350/1/Текст.pdf (дата звернення: 09.06.2025).

12 Ягупов В. В., Петренко Л. М., Кравець С. Г. Дистанційне навчання в системі професійно-технічної освіти монографія [Електронний ресурс]. Режим доступу: https://lib.iitta.gov.ua/id/eprint/721757/1/Дистанц_моногр.pdf (дата звернення: 09.06.2025).

13 Кормило І., Небесний Р. Побудова інформаційної технології візуалізації інформаційних ресурсів [Електронний ресурс]. Режим доступу: https://elartu.tntu.edu.ua/bitstream/lib/23079/2/CAZST_2017v2_Kormilo_I-Construction_of_information_96-97.pdf (дата звернення: 09.06.2025).

14 Головна - Open edX [Електронний ресурс]. Режим доступу: https://openedx.org/uk/ (дата звернення: 09.06.2025).

15 Чим платформа FutureLearn корисна для вашого бізнесу? [Електронний ресурс]. Режим доступу: https://www.visa.com.ua/uk_UA/run-your-business/small-business-tools/partners/futurelearn.html (дата звернення: 09.06.2025).

16 Кіяновська Н. М., Рашевська Н. В., Семеріков С. О. Електронна бібліотека НАПН України [Електронний ресурс]. Режим доступу: https://lib.iitta.gov.ua/id/eprint/706819/1/krs_mono.pdf (дата звернення: 09.06.2025).

17 Бацуровська І. Історичні аспекти розвитку масових відкритих он-лайн курсів у вищій освіті [Електронний ресурс]. Режим доступу: https://doi.org/10.24139/2312-5993/2018.04/250-258 (дата звернення: 09.06.2025).

18 Пугач В. Сучасні освітні платформи для дистанційного навчання [Електронний ресурс]. Режим доступу: https://doi.org/10.52058/2786-5274-2023-13(27)-811-823 (дата звернення: 09.06.2025).

19 Що таке програми з відкритим кодом та які їх переваги [Електронний ресурс]. Режим доступу: https://nadiyno.org/shho-take-programy-z-vidkrytym-kodom-ta-yaki-yih-perevagy/ (дата звернення: 09.06.2025).

20 Django Project – Офіційна документація. [Електронний ресурс]. Режим доступу: https://docs.djangoproject.com/en/stable/ (дата звернення: 11.06.2025).

21 Django REST framework. [Електронний ресурс]. Режим доступу: https://www.django-rest-framework.org/ (дата звернення: 11.06.2025).

22 Karimov, N. *Django 4 By Example: Build powerful and reliable Python web applications from scratch, 4th Edition*, Packt Publishing, 2022. 86–87. (дата звернення: 11.06.2025).

23 Open edX Platform. [Електронний ресурс]. Режим доступу: https://openedx.org/ (дата звернення: 11.06.2025).

24 Vincent, W. *Django for Professionals: Production websites with Python & Django*, 2021. 34–35. (дата звернення: 11.06.2025).

25 Sharma, A. *Hands-On Django REST Framework*, Packt Publishing, 2019. 78–79. (дата звернення: 11.06.2025).

26 Деркач Я. О. Веб-розробка з використанням Django. [Електронний ресурс]. Режим доступу: https://repository.kpi.kharkov.ua/handle/KhPI-Press/52690 (дата звернення: 11.06.2025).

27 React – офіційна документація. [Електронний ресурс]. Режим доступу: https://reactjs.org/ (дата звернення: 11.06.2025).

28 Wieruch, R. *The Road to React*, 2021. 45–46. (дата звернення: 12.06.2025).

29 Banks, A., Porcello, E. *Learning React: Functional Web Development with React and Redux*, O'Reilly, 2020. 15–17. (дата звернення: 12.06.2025).

30 Lidner, A. *React Projects*, Packt Publishing, 2021. 10–11. (дата звернення: 12.06.2025).

31 Мельник О. В. Компонентний підхід до розробки веб-додатків на основі React. [Електронний ресурс]. Режим доступу: https://ela.kpi.ua/handle/123456789/47678 (дата звернення: 12.06.2025).

32 Довбня Д. О. React: підходи до розробки SPA-додатків [Електронний ресурс]. Режим доступу: https://openarchive.nure.ua/handle/document/18422 (дата звернення: 12.06.2025).

33 Khan Academy Engineering Blog – React migration. [Електронний ресурс]. Режим доступу: https://engineering.khanacademy.org/posts/2020-03-13-react-migration.htm (дата звернення: 12.06.2025).

34 PostgreSQL – офіційний сайт. [Електронний ресурс]. Режим доступу: https://www.postgresql.org/ (дата звернення: 12.06.2025).

35 Momjian, B. *PostgreSQL: Introduction and Concepts*, Addison-Wesley, 2001. 78–79. (дата звернення: 12.06.2025).

36 Korry Douglas. *PostgreSQL 13 Administration Cookbook*, Packt Publishing, 2021. 37–38. (дата звернення: 12.06.2025).

37 Beaulieu, A. *Learning SQL: Master SQL Fundamentals*, O'Reilly Media, 2020. 34–35. (дата звернення: 12.06.2025).

38 Django documentation – PostgreSQL features. [Електронний ресурс]. Режим доступу: https://docs.djangoproject.com/en/stable/ref/contrib/postgres/ (дата звернення: 12.06.2025).

39 Дідух В. С. PostgreSQL у веброзробці. [Електронний ресурс]. Режим доступу: https://lib.pnu.edu.ua/handle/123456789/7342 (дата звернення: 12.06.2025).

40 Глущенко О. В. Особливості використання PostgreSQL для навчальних онлайн-систем. [Електронний ресурс]. Режим доступу: https://ela.kpi.ua/handle/123456789/45612 (дата звернення: 12.06.2025).

41 Open edX Overview. [Електронний ресурс]. Режим доступу: https://openedx.org/about-open-edx/ (дата звернення: 12.06.2025).

42 GitHub: edx-platform. [Електронний ресурс]. Режим доступу: https://github.com/openedx/edx-platform (дата звернення: 12.06.2025).

43 Scalable Architecture of Open edX . [Електронний ресурс]. Режим доступу: https://openedx.org/blog/scaling-open-edx-platform/ (дата звернення: 12.06.2025).

44 Dougiamas M., Taylor P. (2003). Moodle: Using Learning Communities to Create an Open Source Course Management System. 49–51. (дата звернення: 12.06.2025).

45 Moodle Developer Documentation. [Електронний ресурс]. Режим доступу: https://moodledev.io/ (дата звернення: 13.06.2025).

46 Pappas C. (2022). Everything You Need to Know About Moodle LMS. // eLearning Industry. [Електронний ресурс]. Режим доступу: https://elearningindustry.com/moodle-lms-overview (дата звернення: 13.06.2025).

47 Instructure. Canvas LMS Overview. [Електронний ресурс]. Режим доступу: https://www.instructure.com/canvas (дата звернення: 13.06.2025).

48 Canvas API Documentation. [Електронний ресурс]. Режим доступу: https://canvas.instructure.com/doc/api/index.html (дата звернення: 13.06.2025).

49 Pappas C. (2021). Canvas LMS Review: Features, Pricing, and Comparison. [Електронний ресурс]. Режим доступу: https://elearningindustry.com/canvas-lms-review-features-pricing-comparison (дата звернення: 13.06.2025).

50 Методологія навчання в Інтернеті [Електронний ресурс]. Режим доступу: https://kwiga.com/ua/help-center/metodologiya-navchannya-v-interneti/analiz-cilovoyi-auditoriyi-dlya-stvorennya-onlajn-kursu (дата звернення: 13.06.2025).

51 Що таке цільова аудиторія: керівництво [Електронний ресурс]. Режим доступу: https://sendpulse.ua/support/glossary/target-audience (дата звернення: 13.06.2025).

52 Побудова стратегії виходу продукту на ринок [Електронний ресурс]. Режим доступу: https://strum.education/lesson/playground/65a15caeb45708c7bf99496b (дата звернення: 13.06.2025).

53 Django Software Foundation. Official Documentation. [Електронний ресурс]. Режим доступу: https://docs.djangoproject.com (дата звернення: 14.06.2025).

54 Holzner, S. (2008). *Beginning Django E-Commerce*. Apress. 35–38. (дата звернення: 14.06.2025).

55 Vincent, A. (2022). *Django for Professionals*. WelcomeToCode. 29–30. (дата звернення: 14.06.2025).

56 Facebook Inc. React Official Documentation. [Електронний ресурс]. Режим доступу: https://reactjs.org (дата звернення: 14.06.2025).

57 Banks, A. & Porcello, E. (2020). *Learning React: Modern Patterns for Developing React Apps*. O'Reilly Media. 77–79. (дата звернення: 14.06.2025).

58 Majors, B. (2021). *React Native in Action*. Manning Publications. 67–68. (дата звернення: 14.06.2025).

59 PostgreSQL Global Development Group. PostgreSQL Documentation. [Електронний ресурс]. Режим доступу: https://www.postgresql.org/docs/ (дата звернення: 14.06.2025).

60 Douglas, J. (2020). *PostgreSQL: Up and Running*. O'Reilly Media. 22–23.(дата звернення: 14.06.2025).

61 Younis, A. (2021). *Mastering PostgreSQL 13*. Packt Publishing. 99–100. (дата звернення: 14.06.2025).

62 FSD (Feature-Sliced Design). [Електронний ресурс]. Режим доступу: https://medium.com/@sriramanvellingiri/feature-sliced-design-in-next-js-7d20be4338de (дата звернення: 14.06.2025).

63 Material Design. [Електронний ресурс]. Режим доступу: https://m3.material.io/ (дата звернення: 15.06.2025).

64 Redux Toolkit. [Електронний ресурс]. Режим доступу: https://redux-toolkit.js.org/ (дата звернення: 15.06.2025).

65 Zod. [Електронний ресурс]. Режим доступу: https://zod.dev/ (дата звернення: 15.06.2025).

66 SSR [Електронний ресурс]. Режим доступу: https://nextjs.org/docs/pages/building-your-application/rendering/server-side-rendering/ (дата звернення: 15.06.2025).

67 JetBrains. PyCharm — Python IDE for Professional Developers. [Електронний ресурс]. Режим доступу: https://www.jetbrains.com/pycharm/ (дата звернення: 15.06.2025).

68 JetBrains. DataGrip: The Cross-Platform IDE for Databases & SQL. [Електронний ресурс]. Режим доступу: https://www.jetbrains.com/datagrip/ (дата звернення: 15.06.2025).

69 GitHub Docs. [Електронний ресурс]. Режим доступу: https://docs.github.com/en (дата звернення: 15.06.2025).

70 Trello. Collaborate on anything. [Електронний ресурс]. Режим доступу: https://trello.com (дата звернення: 15.06.2025).

71 Insomnia. The API Design Platform. [Електронний ресурс]. Режим доступу: https://insomnia.rest/ (дата звернення: 15.06.2025).

72 Sanders, M. S., & McCormick, E. J. (1993). *Human Factors in Engineering and Design*. McGraw-Hill. 44–45. (дата звернення: 15.06.2025).

73 Sheedy, J. E. (2003). *Vision problems at video display terminals: a survey of optometrists*. Journal of the American Optometric Association. 56–57. (дата звернення: 15.06.2025).

74 Smith, M. J., & Carayon, P. (1995). *Work organization, stress, and cumulative trauma disorders*. Human Factors and Ergonomics in Manufacturing. 104–105. (дата звернення: 16.06.2025).

75 Цілі, завдання та вимоги до робочого простору та ергономічних меблів. [Електронний ресурс]. Режим доступу: https://ergo.place/shcho-take-erhonomichnist-tsili-zavdannia-ta-vymohy-do-robochoho-prostoru-ta-erhonomichnykh-mebliv/ (дата звернення: 16.06.2025).

76 Sanders, M. S., & McCormick, E. J. (1993). *Human Factors in Engineering and Design*. McGraw-Hill. 38–39. (дата звернення: 16.06.2025).

77 Sheedy, J. E., Hayes, J. R., & Engle, J. (2003). *Is all computer work bad for the eyes?*. Optometry Journal. 25–26. (дата звернення: 16.06.2025).

78 Robertson, M. M., Ciriello, V. M., & Garabet, A. M. (2007). *Office ergonomics training and a sit-stand workstation: effects on musculoskeletal and visual symptoms and performance*. Work, 28(1), 9–24. (дата звернення: 16.06.2025).

79 The 20-20-20 rule. [Електронний ресурс]. Режим доступу: https://www.cbhs.com.au/mind-and-body/blog/it-s-time-to-make-time-for-a-back-to-school-eye-test 12–14. (дата звернення: 16.06.2025).

80 Харченко, О. Г., Галай, І. О., Бондарчук, І. О., & Яцишин, В. В. (2010). Проектування архітектури WEB-застосування на основі моделі якості проектування. Інженерія Програмного Забезпечення, 4(4), 26.

81 Duda, O., Pasichnyk, V., Kunanets, N., Antonii, R., & Matsiuk, O. (2020). Multidimensional representation of covid-19 data using olap information technology.

82 Duda, O., Kunanets, N., Martsenko, S., Nykytyuk, V., & Pasichnyk, V. (2021). Information technology platform for the selection and analytical processing of information on COVID-19.

83 Nebesnyi, R., Pasichnyk, V., Kunanets, N., Veretennikova, N., & Kunanets, O. (2020). Formation of IT project implementation team [Доповідь на

конференції]. 2020 IEEE 15th International Conference on Computer Sciences and Information Technologies (CSIT), Збараж, Україна.

84 Nebesnyi, R., Kunanets, N., Vaskiv, R., & Veretennikova, N. (2021). Formation of an IT project team in the context of PMBOK requirements [Доповідь на конференції]. 2021 IEEE 16th International Conference on Computer Sciences and Information Technologies (CSIT), Львів, Україна.

ДОДАТКИ

Додаток А

Лістинг 1 —tutorspace_project/tutorspace/asgi.py

```
"""
ASGI config for tutorspace project.

It exposes the ASGI callable as a module-level variable
named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/howto/deployment/a
sgi/
"""

import os

from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'tutorspace.settings')

application = get_asgi_application()
```

Лістинг 2 —tutorspace_project/tutorspace/error_handler_middleware.py

```
import logging
from django.conf import settings
from django.http import JsonResponse

logger = logging.getLogger(__name__)

class ErrorHandlingMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        response = self.get_response(request)

        if 300 <= response.status_code < 500:
            return self.handle_error(response)
```

```
        return response

    def handle_error(self, response):
        """
        Handles different types of errors (300-500 status codes).
        """
        if settings.DEBUG:
            logger.error(f"Error occurred: {response.status_code}
{response.content}", exc_info=True)
            return JsonResponse({
                'error': f'Error {response.status_code}: Something went
wrong.',
                'details': response.content.decode()
            }, status=response.status_code)
        else:
            logger.error(f"Error occurred: {response.status_code}",
exc_info=True)
            return JsonResponse({
                'error': 'Something went wrong. Please try again later.'
            }, status=response.status_code)
```

Лістинг 3 —tutorspace_project/tutorspace/settings.py

```
"""
Django settings for tutorspace project.

Generated by 'django-admin startproject' using Django 4.2.16.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.2/ref/settings/
"""
import os

from os import getenv, path
from pathlib import Path
from django.core.management.utils import get_random_secret_key
import dotenv
from django_ses.settings import AWS_SES_REGION_NAME
from datetime import timedelta

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

dotenv_file = BASE_DIR / ".env.local"

if path.isfile(dotenv_file):
    dotenv.load_dotenv(dotenv_file)

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = getenv('DJANGO_SECRET_KEY', get_random_secret_key())

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = getenv('DJANGO_DEBUG', 'False') == 'True'
```

```
ALLOWED_HOSTS = ['*']

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django_bigbluebutton',
    'corsheaders',
    'rest_framework',
    'drf_yasg',
    'djoser',
    'social_django',
    'users',
    'bank',
    'commands',
    'languages',
    'specializations',
    'usersconfig',
    'review',
    'course',
    'certificate'
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'tutorspace.error_handler_middleware.ErrorHandlingMiddleware',
]

ROOT_URLCONF = 'tutorspace.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'tutorspace.wsgi.application'
```

```
# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'postgres',
        'USER': 'test',
        'PASSWORD': 'qa123123',
        'HOST': 'localhost',
        'PORT': '5432',
        'ATOMIC_REQUESTS': True
    }
}

# Email settings

EMAIL_BACKEND = 'django_ses.SESBackend'
DEFAULT_FROM_EMAIL = getenv('DJANGO_AWS_SES_FROM_EMAIL')

AWS_SES_ACCESS_KEY_ID = getenv('DJANGO_AWS_SES_ACCESS_KEY_ID')
AWS_SES_SECRET_ACCESS_KEY = getenv('DJANGO_AWS_SES_SECRET_ACCESS_KEY')
AWS_SES_REGION_NAME = getenv('DJANGO_AWS_SES_REGION_NAME')
AWS_SES_REGION_ENDPOINT = f'email.{AWS_SES_REGION_NAME}.amazonaws.com'
AWS_SES_FROM_EMAIL = getenv('DJANGO_AWS_SES_FROM_EMAIL')
USE_SES_V2 = True


# Password validation
# https://docs.djangoproject.com/en/4.2/ref/settings/#auth-password-
validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator
',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/4.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True
```

```
USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/

STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

AUTHENTICATION_BACKENDS = [
    'social_core.backends.google.GoogleOAuth2',
    'social_core.backends.facebook.FacebookOAuth2',
    'django.contrib.auth.backends.ModelBackend',
]

REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'users.authentication.CustomJWTAuthentication',
    ],
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
    ]
}

AUTH_COOKIE = 'access'
AUTH_COOKIE_MAX_AGE = 60 * 60 * 24
AUTH_COOKIE_SECURE = getenv('AUTH_COOKIE_SECURE', 'False') == 'True'
AUTH_COOKIE_HTTP_ONLY = True
AUTH_COOKIE_PATH = '/'
AUTH_COOKIE_SAMESITE = 'Lax'

DJOSER = {
    'PASSWORD_RESET_CONFIRM_URL': 'password-reset/{uid}/{token}',
    'SEND_ACTIVATION_EMAIL': False,
    'ACTIVATION_URL': 'activation/{uid}/{token}',
    'USER_CREATE_PASSWORD_RETYPE': True,
    'PASSWORD_RESET_CONFIRM_RETYPE': True,
    'TOKEN_MODEL': None,
    'SOCIAL_AUTH_ALLOWED_REDIRECT_URIS': None,
    'SERIALIZERS': {
        'user_create_password_retype':
'users.serializers.userCreateSerializer.UserAccountCreateSerializer',
        'user_create': 'users.serializers.UserAccountCreateSerializer',
        'current_user': 'users.serializers.UserAccountSerializer',
    }
}

SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=15),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=7),
    'ROTATE_REFRESH_TOKENS': False,
    'BLACKLIST_AFTER_ROTATION': True,
    'AUTH_HEADER_TYPES': ('Bearer',),
}

SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = getenv('GOOGLE_AUTH_KEY')
SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET = getenv('GOOGLE_AUTH_SECRET_KEY')
```

```
SOCIAL_AUTH_GOOGLE_OAUTH2_SCOPE = [
    'https://www.googleapis.com/auth/userinfo.email',
    'https://www.googleapis.com/auth/userinfo.profile',
    'openid'
]
SOCIAL_AUTH_GOOGLE_OAUTH2_EXTRA_DATA = ['first_name', 'last_name']

SOCIAL_AUTH_FACEBOOK_KEY = getenv('FACEBOOK_AUTH_KEY')
SOCIAL_AUTH_FACEBOOK_SECRET = getenv('FACEBOOK_AUTH_SECRET_KEY')
SOCIAL_AUTH_FACEBOOK_SCOPE = ['email']
SOCIAL_AUTH_FACEBOOK_PROFILE_EXTRA_PARAMS = {
    'fields': 'email, first_name, last_name'
}

CORS_ALLOWED_ORIGINS = getenv('DJANGO_CORS_ALLOWED_ORIGINS',
'http://localhost:3000,http://127.0.0.1:3000').split(',')
CORS_ALLOW_CREDENTIALS = True

# Default primary key field type
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

AUTH_USER_MODEL = 'users.UserAccount'

BBB_API_URL = 'https://test.com/bigbluebutton/api/'
BBB_SECRET_KEY = 'abcdefgabcdefgabcdefgabcdefgabcdefg'
```

Лістинг 4 —tutorspace_project/tutorspace/urls.py

```
from django.contrib import admin
from django.urls import path, re_path, include
from rest_framework import permissions
from drf_yasg.views import get_schema_view
from drf_yasg import openapi
from django.conf import settings
from django.conf.urls.static import static

schema_view = get_schema_view(
    openapi.Info(
        title="TutorSpace API",
        default_version="v1",
        description="TutorSpace API documentation",
        terms_of_service="https://www.test.com/terms/",
        contact=openapi.Contact(email="test@test.com"),
        license=openapi.License(name="MIT License"),
    ),
    public=True,
    permission_classes=[permissions.AllowAny],
)

urlpatterns = [
    path('admin/', admin.site.urls),
    path('tutor/', include('django_bigbluebutton.urls')),
    path('api/', include('djoser.urls')),
    path('api/', include('users.urls')),
    path('swagger/', schema_view.with_ui('swagger', cache_timeout=0),
name='schema-swagger-ui'),
    path('redoc/', schema_view.with_ui('redoc', cache_timeout=0),
name='schema-redoc'),
    re_path(r'^swagger(?P<format>\.json|\.yaml)$',
```

```
schema_view.without_ui(cache_timeout=0), name='schema-json'),
    path('api/languages/', include('languages.urls')),
    path('api/specializations/', include('specializations.urls')),
    path('api/bank-cards/', include('bank.urls')),
    path('api/review/', include('review.urls')),
    path('api/course/', include('course.urls')),
    path('api/certificate/', include('certificate.urls')),
]

urlpatterns += static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)
urlpatterns += static(settings.STATIC_URL,
document_root=settings.STATIC_ROOT)
```

Лістинг 4 —tutorspace_project/tutorspace/wsgi.py

```
"""
WSGI config for tutorspace project.

It exposes the WSGI callable as a module-level variable named
``application``.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'tutorspace.settings')

application = get_wsgi_application()
```

Лістинг 5 —tutorspace_project/users/views.py

```
from django.conf import settings
from rest_framework import status, generics
from djoser.social.views import ProviderAuthView
from rest_framework_simplejwt.views import (
    TokenObtainPairView,
    TokenRefreshView,
    TokenVerifyView
)
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework.permissions import IsAuthenticated
from .serializers import UserAccountSerializer
from rest_framework.generics import UpdateAPIView
from users.serializers.userDetailSerializer import
FullUserAccountSerializer
from users.serializers.userUpdateSerializer import
FullUserAccountUpdateSerializer
from users.models.socialMedia import SocialMediaLink
from users.serializers.socialMediaSerializer import
SocialMediaLinkSerializer
from users.pagination import UsersResultsSetPagination
from users.serializers.userChangePasswordSerializer import
ChangePasswordSerializer
from users.models.userAccount import UserAccount
```

```
class CustomProviderAuthView(ProviderAuthView):
    def post(self, request, *args, **kwargs):
        response = super().post(request, *args, **kwargs)

        if response.status_code == 201:
            access_token = response.data.get('access')
            refresh_token = response.data.get('refresh')

            response.set_cookie(
                'access',
                access_token,
                max_age=settings.AUTH_COOKIE_MAX_AGE,
                path=settings.AUTH_COOKIE_PATH,
                secure=settings.AUTH_COOKIE_SECURE,
                httponly=settings.AUTH_COOKIE_HTTP_ONLY,
                samesite=settings.AUTH_COOKIE_SAMESITE
            )
            response.set_cookie(
                'refresh',
                refresh_token,
                max_age=settings.AUTH_COOKIE_MAX_AGE,
                path=settings.AUTH_COOKIE_PATH,
                secure=settings.AUTH_COOKIE_SECURE,
                httponly=settings.AUTH_COOKIE_HTTP_ONLY,
                samesite=settings.AUTH_COOKIE_SAMESITE
            )

        return response


class CustomTokenObtainPairView(TokenObtainPairView):
    def post(self, request, *args, **kwargs):
        response = super().post(request, *args, **kwargs)

        if response.status_code == 200:
            access_token = response.data.get('access')
            refresh_token = response.data.get('refresh')

            response.set_cookie(
                'access',
                access_token,
                max_age=settings.AUTH_COOKIE_MAX_AGE,
                path=settings.AUTH_COOKIE_PATH,
                secure=settings.AUTH_COOKIE_SECURE,
                httponly=settings.AUTH_COOKIE_HTTP_ONLY,
                samesite=settings.AUTH_COOKIE_SAMESITE
            )
            response.set_cookie(
                'refresh',
                refresh_token,
                max_age=settings.AUTH_COOKIE_MAX_AGE,
                path=settings.AUTH_COOKIE_PATH,
                secure=settings.AUTH_COOKIE_SECURE,
                httponly=settings.AUTH_COOKIE_HTTP_ONLY,
                samesite=settings.AUTH_COOKIE_SAMESITE
            )

        return response


class CustomTokenRefreshView(TokenRefreshView):
```

```
    def post(self, request, *args, **kwargs):
        refresh_token = request.COOKIES.get('refresh')

        if refresh_token:
            request.data['refresh'] = refresh_token

        response = super().post(request, *args, **kwargs)

        if response.status_code == 200:
            access_token = response.data.get('access')

            response.set_cookie(
                'access',
                access_token,
                max_age=settings.AUTH_COOKIE_MAX_AGE,
                path=settings.AUTH_COOKIE_PATH,
                secure=settings.AUTH_COOKIE_SECURE,
                httponly=settings.AUTH_COOKIE_HTTP_ONLY,
                samesite=settings.AUTH_COOKIE_SAMESITE
            )

        return response


class CustomTokenVerifyView(TokenVerifyView):
    def post(self, request, *args, **kwargs):
        access_token = request.COOKIES.get('access')

        if access_token:
            request.data['token'] = access_token

        return super().post(request, *args, **kwargs)


class LogoutView(APIView):
    def post(self, request, *args, **kwargs):
        response = Response(status=status.HTTP_204_NO_CONTENT)
        response.delete_cookie('access')
        response.delete_cookie('refresh')

        return response


class MeView(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request):
        serializer = UserAccountSerializer(request.user)
        return Response(serializer.data)


class FullUserDetailsView(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request):
        serializer = FullUserAccountSerializer(request.user)
        return Response(serializer.data)


class UpdateUserProfileView(UpdateAPIView):
    permission_classes = [IsAuthenticated]
```

```
    serializer_class = FullUserAccountUpdateSerializer

    def get_object(self):
        return self.request.user


class DeleteUserAccountView(APIView):
    permission_classes = [IsAuthenticated]

    def delete(self, request):
        user = request.user
        user.delete()
        return Response({"detail": "User account deleted successfully."},
status=status.HTTP_204_NO_CONTENT)


class SocialMediaLinkListView(generics.ListAPIView):
    queryset = SocialMediaLink.objects.all()
    serializer_class = SocialMediaLinkSerializer
    pagination_class = UsersResultsSetPagination


class SocialMediaLinkRetrieveView(generics.RetrieveAPIView):
    permission_classes = [IsAuthenticated]
    queryset = SocialMediaLink.objects.all()
    serializer_class = SocialMediaLinkSerializer


class ChangePasswordView(APIView):
    permission_classes = [IsAuthenticated]

    def post(self, request):
        serializer = ChangePasswordSerializer(data=request.data,
context={'request': request})
        if serializer.is_valid():
            serializer.save()
            return Response({"detail": "Password changed successfully."},
status=status.HTTP_200_OK)
        return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)


class SocialMediaLinkBulkView(APIView):
    permission_classes = [IsAuthenticated]

    def post(self, request):
        serializer = SocialMediaLinkSerializer(data=request.data,
many=True)
        serializer.is_valid(raise_exception=True)
        serializer.save()
        return Response(serializer.data, status=status.HTTP_201_CREATED)

    def put(self, request):
        data = request.data

        if not isinstance(data, list):
            return Response({"error": "Expected a list of items."},
status=400)

        updated_links = []
```

```
        for item in data:
            try:
                obj = SocialMediaLink.objects.get(id=item.get('id'))
            except SocialMediaLink.DoesNotExist:
                continue

            serializer = SocialMediaLinkSerializer(obj, data=item,
partial=True)
            serializer.is_valid(raise_exception=True)
            serializer.save()
            updated_links.append(serializer.data)

        return Response(updated_links, status=200)

    def delete(self, request):
        ids = request.data.get('ids', [])
        if not isinstance(ids, list):
            return Response({"error": "Expected 'ids' to be a list."},
status=400)

        SocialMediaLink.objects.filter(id__in=ids).delete()
        return Response({"deleted": ids},
status=status.HTTP_204_NO_CONTENT)


class ToggleUserActivationView(APIView):
    permission_classes = [IsAuthenticated]

    def put(self, request, user_id):
        try:
            user = UserAccount.objects.get(id=user_id)
        except UserAccount.DoesNotExist:
            return Response({"error": "User not found."},
status=status.HTTP_404_NOT_FOUND)

        user.is_active = not user.is_active
        user.save()

        return Response({
            "user_id": user.id,
            "is_active": user.is_active,
            "message": f"User {'activated' if user.is_active else
'deactivated'} successfully."
        }, status=status.HTTP_200_OK)
```

Лістинг 6 —tutorspace_project/users/urls.py

```
from django.urls import path, re_path
from .views import (
    CustomProviderAuthView,
    CustomTokenObtainPairView,
    CustomTokenRefreshView,
    CustomTokenVerifyView,
    LogoutView,
    MeView,
    FullUserDetailsView,
    UpdateUserProfileView,
    DeleteUserAccountView,
    SocialMediaLinkListView,
    SocialMediaLinkRetrieveView,
    SocialMediaLinkBulkView,
```

```
    ChangePasswordView,
    ToggleUserActivationView
)

urlpatterns = [
    re_path(
        r'^o/(?P<provider>\S+)/$',
        CustomProviderAuthView.as_view(),
        name='provider-auth'
    ),
    path('jwt/create/', CustomTokenObtainPairView.as_view()),
    path('jwt/refresh/', CustomTokenRefreshView.as_view()),
    path('jwt/verify/', CustomTokenVerifyView.as_view()),
    path('logout/', LogoutView.as_view()),
    path('me/', MeView.as_view(), name='me'),
    path('profile/', FullUserDetailsView.as_view(), name='profile'),
    path('profile/update/', UpdateUserProfileView.as_view(),
name='update-user'),
    path('profile/delete/', DeleteUserAccountView.as_view(),
name='delete-user'),
    path('social-links/', SocialMediaLinkListView.as_view(),
name='social-link-list'),
    path('social-links/create/', SocialMediaLinkBulkView.as_view(),
name='social-link-create'),
    path('social-links/<int:pk>/', SocialMediaLinkRetrieveView.as_view(),
name='social-link-detail'),
    path('social-links/update/', SocialMediaLinkBulkView.as_view(),
name='social-link-update'),
    path('social-links/delete/', SocialMediaLinkBulkView.as_view(),
name='social-link-delete'),
    path('change-password/', ChangePasswordView.as_view(), name='change-
password'),
    path('<int:user_id>/toggle-activation/',
ToggleUserActivationView.as_view(), name='toggle-user-activation'),
]
```

Лістинг 7 —tutorspace_project/users/pagination.py

```
from rest_framework.pagination import PageNumberPagination

class UsersResultsSetPagination(PageNumberPagination):
    page_size = 10
    page_size_query_param = 'page_size'
    max_page_size = 100
```

Лістинг 8 —tutorspace_project/users/authentication.py

```
from django.conf import settings
from rest_framework_simplejwt.authentication import JWTAuthentication


class CustomJWTAuthentication(JWTAuthentication):
    def authenticate(self, request):
        try:
            header = self.get_header(request)

            if header is None:
                raw_token = request.COOKIES.get(settings.AUTH_COOKIE)
            else:
                raw_token = self.get_raw_token(header)
```

```
            if raw_token is None:
                return None

            validated_token = self.get_validated_token(raw_token)

            return self.get_user(validated_token), validated_token
        except:
            return None
```

Лістинг 9 —tutorspace_project/users/apps.py

```
from django.apps import AppConfig


class UsersConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'users'
```

Лістинг 10 —
tutorspace_project/users/serializers/socialMediaSerializer.py

```
from rest_framework import serializers
from users.models import SocialMediaLink


class SocialMediaLinkSerializer(serializers.ModelSerializer):
    class Meta:
        model = SocialMediaLink
        fields = '__all__'
```

Лістинг 11 —
tutorspace_project/users/serializers/userChangePasswordSerializer.py

```
from rest_framework import serializers


class ChangePasswordSerializer(serializers.Serializer):
    current_password = serializers.CharField(write_only=True)
    new_password = serializers.CharField(write_only=True)
    re_new_password = serializers.CharField(write_only=True)

    def validate(self, attrs):
        user = self.context['request'].user
        current_password = attrs.get('current_password')
        new_password = attrs.get('new_password')
        re_new_password = attrs.get('re_new_password')

        if not user.check_password(current_password):
            raise serializers.ValidationError({"error": "Current password
is incorrect."})

        if new_password != re_new_password:
            raise serializers.ValidationError({"error": "New passwords do
not match."})

        if current_password == new_password:
            raise serializers.ValidationError({"error": "New password
must be different from the current one."})

        return attrs

    def save(self, **kwargs):
```

```
        user = self.context['request'].user
        user.set_password(self.validated_data['new_password'])
        user.save()
        return user
```

Лістинг 11 —tutorspace_project/users/serializers/userCreateSerializer.py

```
from djoser.serializers import UserCreatePasswordRetypeSerializer
from rest_framework import serializers
from django.contrib.auth import get_user_model

User = get_user_model()

class UserAccountCreateSerializer(UserCreatePasswordRetypeSerializer):
    class Meta(UserCreatePasswordRetypeSerializer.Meta):
        model = User
        fields = (*UserCreatePasswordRetypeSerializer.Meta.fields,
'first_name', 'last_name', 'role')

        def validate_role(self, value):
            allowed_roles = ['tutor', 'student', 'admin']
            if value not in allowed_roles:
                raise serializers.ValidationError("Invalid role.")
            return value
```

Лістинг 12 —tutorspace_project/users/serializers/userDetailSerializer.py

```
from rest_framework import serializers
from users.models import UserAccount
from users.serializers.socialMediaSerializer import
SocialMediaLinkSerializer
from bank.serializers.bankSerializer import BankCardSerializer


class FullUserAccountSerializer(serializers.ModelSerializer):
    social_links = SocialMediaLinkSerializer(many=True, read_only=True)
    bank_cards = BankCardSerializer(many=True, read_only=True)
    role = serializers.SerializerMethodField()
    sex = serializers.SerializerMethodField()

    class Meta:
        model = UserAccount
        fields = [
            'id',
            'first_name',
            'last_name',
            'email',
            'phone',
            'specialization',
            'language',
            'rating',
            'birthday',
            'sex',
            'about_me',
            'avatar',
            'is_active',
            'is_staff',
            'is_superuser',
            'role',
            'social_links',
            'bank_cards'
        ]
```

```
    def get_role(self, obj):
        return [group.name for group in obj.groups.all()]

    def get_sex(self, obj):
        return obj.get_sex_display() if obj.sex else None
```

Лістинг 13 —tutorspace_project/users/serializers/userMeSerializer.py

```
from rest_framework import serializers
from users.models.userAccount import UserAccount
from usersconfig.models.userConfig import UserConfig

class UserAccountSerializer(serializers.ModelSerializer):
    phone_number = serializers.CharField(source='phone', required=False)
    image = serializers.CharField(source='avatar', required=False)
    role = serializers.SerializerMethodField()
    config = serializers.SerializerMethodField()

    class Meta:
        model = UserAccount
        fields = [
            'id',
            'image',
            'first_name',
            'last_name',
            'email',
            'phone_number',
            'role',
            'config',
            'is_active'
        ]

    def get_role(self, obj):
        return [group.name for group in obj.groups.all()]

    def get_config(self, obj):
        try:
            return obj.config.config
        except UserConfig.DoesNotExist:
            return {}
```

Лістинг 14 —tutorspace_project/users/serializers/userUpdateSerializer.py

```
from rest_framework import serializers
from users.models.userAccount import UserAccount
from users.serializers.socialMediaSerializer import SocialMediaLink,
SocialMediaLinkSerializer
from bank.serializers.bankSerializer import BankCard
from specializations.models.specializations import Specialization
from languages.models.languages import Languages
from usersconfig.models.userConfig import UserConfig


class FullUserAccountUpdateSerializer(serializers.ModelSerializer):
    social_links = SocialMediaLinkSerializer(many=True, required=False)
    bank_cards = serializers.PrimaryKeyRelatedField(
        queryset=BankCard.objects.all(),
        many=True,
        required=False
```

```
    )
    specialization = serializers.PrimaryKeyRelatedField(
        queryset=Specialization.objects.all(),
        many=True,
        required=False
    )
    language = serializers.PrimaryKeyRelatedField(
        queryset=Languages.objects.all(),
        many=True,
        required=False
    )
    sex = serializers.ChoiceField(
        choices=['Male', 'Female', 'Other'],
        required=False,
        allow_blank=True
    )
    config_data = serializers.SerializerMethodField()

    class Meta:
        model = UserAccount
        fields = [
            'first_name', 'last_name', 'phone', 'specialization',
            'language', 'rating', 'birthday', 'sex', 'about_me',
'avatar',
            'social_links', 'bank_cards', 'config_data'
        ]

    def update(self, instance, validated_data):
        specialization_ids = self.initial_data.get('specialization', [])
        language_ids = self.initial_data.get('language', [])
        social_links_data = self.initial_data.get('social_links', [])
        bank_card_ids = self.initial_data.get('bank_cards', [])
        config_data = self.initial_data.get('config_data', {})

        validated_data.pop('specialization', None)
        validated_data.pop('language', None)
        validated_data.pop('social_links', None)
        validated_data.pop('bank_cards', None)
        validated_data.pop('config_data', None)

        self._update_user_fields(instance, validated_data)


instance.specialization.set(Specialization.objects.filter(id__in=speciali
zation_ids))

instance.language.set(Languages.objects.filter(id__in=language_ids))

        self._sync_social_links(instance, social_links_data)
        instance.bank_cards.exclude(id__in=bank_card_ids).delete()

BankCard.objects.filter(id__in=bank_card_ids).update(user=instance)

        user_config, exists =
UserConfig.objects.get_or_create(user=instance)
        if not exists:
            user_config.config.update(config_data)
        user_config.save()

        return instance
```

```
    def _update_user_fields(self, instance, fields):
        for attr, value in fields.items():
            setattr(instance, attr, value)
        instance.save()

    @staticmethod
    def _sync_specializations(user, specialization_ids):
        if isinstance(specialization_ids, list):
            user.specialization.set(specialization_ids)

    @staticmethod
    def _sync_languages(user, language_ids):
        if isinstance(language_ids, list):
            user.language.set(language_ids)

    def get_config_data(self, obj):
        try:
            return obj.config.config
        except UserConfig.DoesNotExist:
            return {}

    def _sync_social_links(self, user, links_data):
        ids_to_keep = []

        for link_data in links_data:
            if isinstance(link_data, dict):
                link_id = link_data.get("id")
                if link_id:
                    try:
                        link = user.social_links.get(id=link_id)
                        for key, value in link_data.items():
                            if key == "user":
                                continue
                            setattr(link, key, value)
                        link.save()
                        ids_to_keep.append(link.id)
                    except SocialMediaLink.DoesNotExist:
                        continue
                else:
                    new_link = SocialMediaLink.objects.create(user=user,
**link_data)
                    ids_to_keep.append(new_link.id)
            elif isinstance(link_data, int):
                try:
                    link = SocialMediaLink.objects.get(id=link_data)
                    link.user = user
                    link.save()
                    ids_to_keep.append(link.id)
                except SocialMediaLink.DoesNotExist:
                    continue


SocialMediaLink.objects.filter(user=user).exclude(id__in=ids_to_keep).del
ete()
```

Додаток Б

Лістинг 1 – src/shared/config/routes.ts

```
import type { TRoutesType } from '@/shared/types/routes';

import {
  Home,
  AutoStories,
  PeopleAlt,
  School,
  Forum,
  Settings
} from '@mui/icons-material';

const routes: TRoutesType = {
  home: {
    key: 'home',
    label: 'Home',
    path: '/',
    viewBy: 'all',
    protected: true,
    navigationRoute: true,
    icon: Home
  },
  courses: {
    key: 'courses',
    label: 'Courses',
    path: '/courses',
    viewBy: 'all',
    protected: true,
    navigationRoute: true,
    icon: AutoStories
  },
  students: {
    key: 'students',
    label: 'Students',
    path: '/students',
    viewBy: 'teacher',
    protected: true,
    navigationRoute: true,
    icon: PeopleAlt
  },
  teachers: {
    key: 'teachers',
    label: 'Teachers',
    path: '/teachers',
    viewBy: 'student',
    protected: true,
    navigationRoute: true,
    icon: School
  },
  chats: {
    key: 'chats',
    label: 'Chats',
    path: '/chats',
    viewBy: 'all',
    protected: true,
    navigationRoute: true,
    icon: Forum
  },
```

```
  settings: {
    key: 'settings',
    label: 'Settings',
    path: '/settings',
    viewBy: 'all',
    protected: true,
    navigationRoute: true,
    icon: Settings
  },
  login: {
    key: 'login',
    label: 'Login',
    path: '/login',
    viewBy: 'all',
    protected: false
  },
  register: {
    key: 'register',
    label: 'Register',
    path: '/register',
    viewBy: 'all',
    protected: false
  },
  passwordReset: {
    key: 'passwordReset',
    label: 'Password Reset',
    path: '/password-reset',
    viewBy: 'all',
    protected: false
  }
};

export default routes;
```

Лістинг 2 – src/shared/config/store/index.ts

```
import type { ThunkAction, UnknownAction } from '@reduxjs/toolkit';

import authSlice from '@/store/slices/auth';
import { baseApi } from '@/shared/api';
import { configureStore } from '@reduxjs/toolkit';
import { UseExtraArguments } from '@/shared/hooks/store/use-extra-
arguments';

export const makeStore = <T>(extraArgument: T) => configureStore({
  reducer: {
    [baseApi.reducerPath]: baseApi.reducer,
    [authSlice.name]: authSlice.reducer,
  },
  devTools: process.env.NODE_ENV !== 'production',
  middleware: (getDefaultMiddleware) => {
    return getDefaultMiddleware({
      thunk: { extraArgument }
    }).concat(baseApi.middleware);
  }
});
type TAppStore = ReturnType<typeof makeStore>;
export type TExtraArguments = ReturnType<typeof UseExtraArguments>;
export type TAppStoreState = ReturnType<TAppStore['getState']>;
export type TAppStoreDispatch = TAppStore['dispatch'];
export type TAppThunk<R = void> = ThunkAction<R, TAppStoreState, any,
```

```
UnknownAction>;
```

Лістинг 3 – src/shared/api/index.ts

```
import type {
  BaseQueryFn,
  FetchArgs,
  FetchBaseQueryError
} from '@reduxjs/toolkit/query/react';

import authSlice from '@/store/slices/auth';
import getCookieOnTheClient from '@/shared/utils/get-cookie-on-the-
client';
import { API_POST_AUTH_REFRESH_TOKEN, BASE_URL } from
'@/shared/constants';
import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react';
import { Mutex } from 'async-mutex';

const mutex = new Mutex();

const baseQuery = fetchBaseQuery({
  baseUrl: BASE_URL,
  credentials: 'include'
});

const baseQueryWithReAuth: BaseQueryFn<
  string | FetchArgs,
  unknown,
  FetchBaseQueryError
> = async (args, api, extraArguments) => {
  await mutex.waitForUnlock();

  let result = await baseQuery(args, api, extraArguments);

  if (result.error && result.error.status === 401) {
    if (!mutex.isLocked()) {
      const release = await mutex.acquire();

      try {
        const refreshResult = await baseQuery(
          {
            url: API_POST_AUTH_REFRESH_TOKEN,
            method: 'POST',
            body: JSON.stringify({
              refresh: getCookieOnTheClient('refresh')
            })
          },
          api,
          extraArguments
        );

        if (refreshResult.data) {
          api.dispatch(authSlice.actions.setAuth);
          result = await baseQuery(args, api, extraArguments);
        } else {
          api.dispatch(authSlice.actions.logout);
        }
      } finally {
        release();
      }
    } else {
      await mutex.waitForUnlock();
```

```
      result = await baseQuery(args, api, extraArguments);
    }
  }
  return result;
};

export const baseApi = createApi({
  reducerPath: 'api',
  baseQuery: baseQueryWithReAuth,
  tagTypes: ['User'],
  endpoints: () => ({})
});
```

Лістинг 4 – src/shared/api/auth/index.ts

```
import {
  TCurrentUser,
  IAuthActivationPayload,
  IAuthLoginPayload,
  IAuthRegisterPayload,
  IAuthResetPasswordPayload,
  IAuthSocialAuthenticatePayload,
  IAuthResetPasswordConfirmationPayload,
  TLoginResponse,
  TRegistrationResponse
} from '@/shared/api/auth/types';

import { baseApi } from '@/shared/api';
import {
  currentUserSchema,
  loginResponseSchema,
  registerResponseSchema
} from '@/shared/api/auth/models';

import {
  API_GET_AUTH_ME,
  API_POST_AUTH_LOGIN,
  API_POST_AUTH_REGISTER,
  API_POST_AUTH_VERIFY,
  API_POST_AUTH_LOGOUT,
  API_POST_RESET_PASSWORD,
  API_POST_AUTH_ACTIVATION,
  API_POST_RESET_PASSWORD_CONFIRMATION
} from '@/shared/constants';

export const authApi = baseApi.injectEndpoints({
  endpoints: create => ({
    // GET
    getMe: create.query<TCurrentUser, void>({
      query: () => ({ url: API_GET_AUTH_ME }),
      transformResponse: (res: unknown) => currentUserSchema.parse(res),
      providesTags: ['User']
    }),

    // POST
    postLogin: create.mutation<TLoginResponse, IAuthLoginPayload>({
      query: payload => ({
        url: API_POST_AUTH_LOGIN,
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(payload)
      }),
```

```
      transformResponse: (res: unknown) =>
loginResponseSchema.parse(res),
      invalidatesTags: ['User']
    }),
    postRegister: create.mutation<TRegistrationResponse,
IAuthRegisterPayload>({
      query: payload => ({
        url: API_POST_AUTH_REGISTER,
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(payload)
      }),
      transformResponse: (res: unknown) =>
registerResponseSchema.parse(res),
      invalidatesTags: ['User']
    }),
    postSocialAuthenticate: create.mutation<
      void,
      IAuthSocialAuthenticatePayload
    >({
      //TODO: need to add response type and response object validation
      query: ({ provider, state, code }) => ({
        url:
`/o/${provider}/?state=${encodeURIComponent(state)}&code=${encodeURICompo
nent(code)}`,
        method: 'POST',
        headers: {
          Accept: 'application/json',
          'Content-Type': 'application/x-www-form-urlencoded'
        }
      }),
      invalidatesTags: ['User']
    }),
    postVerify: create.mutation<void, void>({
      query: () => ({
        url: API_POST_AUTH_VERIFY,
        method: 'POST'
      }),
      invalidatesTags: ['User']
    }),
    postActivation: create.mutation<void, IAuthActivationPayload>({
      query: payload => ({
        url: API_POST_AUTH_ACTIVATION,
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(payload)
      }),
      invalidatesTags: ['User']
    }),
    postLogout: create.mutation<void, void>({
      query: () => ({
        url: API_POST_AUTH_LOGOUT,
        method: 'POST'
      }),
      invalidatesTags: ['User']
    }),
    postResetPassword: create.mutation<any, IAuthResetPasswordPayload>({
      query: payload => ({
        url: API_POST_RESET_PASSWORD,
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
```

```
        body: JSON.stringify(payload)
      }),
      invalidatesTags: ['User']
    }),
    postResetPasswordConfirm: create.mutation<
      void,
      IAuthResetPasswordConfirmationPayload
    >({
      query: payload => ({
        url: API_POST_RESET_PASSWORD_CONFIRMATION,
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(payload)
      }),
      invalidatesTags: ['User']
    })
  }),
  overrideExisting: true
});
```

Лістинг 5 – src/shared/hooks/use-locations.ts

```
'use client';
import type { TLocalesList } from '@/shared/types/i18n';
import type { IUseLocations } from '@/shared/types/hooks/use-locations';

import * as constants from '@/shared/constants';

import { useRouter } from 'next/navigation';
import { useCallback, useEffect, useState } from 'react';
import getCookieOnTheClient from '@/shared/utils/get-cookie-on-the-
client';

export const useLocations = (): IUseLocations => {
  const router = useRouter();

  const [locale, setLocale] = useState<TLocalesList>('en');

  const changeLocale = useCallback(
    (locale: TLocalesList) => {
      setLocale(locale);

      document.cookie = `${constants.COOKIE_LOCALE}=${locale}`;
      router.refresh();
    },
    [router]
  );

  useEffect(() => {
    const cookieLocale = getCookieOnTheClient<TLocalesList>(
      constants.COOKIE_LOCALE
    );

    if (cookieLocale) {
      setLocale(cookieLocale);
      return;
    }

    const browserLocale = navigator.language
      .slice(3, 5)
      .toLowerCase() as TLocalesList;
    setLocale(browserLocale);
```

```
    document.cookie = `${constants.COOKIE_LOCALE}=${browserLocale}`;
    router.refresh();
  }, [router]);

  return {
    locale,
    changeLocale
  };
};
```

Лістинг 6 – src/shared/hooks/use-theme-switcher.ts

```
'use client';
import type { IUseThemeSwitcher } from '@/shared/types/hooks/use-theme-
switcher';
import type { ThemeModeType } from '@/shared/types/theme';

import * as constants from '@/shared/constants';
import getCookieOnTheClient from '@/shared/utils/get-cookie-on-the-
client';
import { useRouter } from 'next/navigation';
import { useCallback, useEffect, useState } from 'react';

export const useThemeSwitcher = (): IUseThemeSwitcher => {
  const router = useRouter();

  const [theme, setTheme] = useState<ThemeModeType>('light');

  const changeTheme = useCallback(
    (theme: ThemeModeType) => {
      setTheme(theme);

      document.cookie = `${constants.COOKIE_THEME}=${theme}`;
      router.refresh();
    },
    [router]
  );

  useEffect(() => {
    const cookieTheme = getCookieOnTheClient<ThemeModeType>(
      constants.COOKIE_THEME
    );

    if (cookieTheme) {
      setTheme(cookieTheme);
      return;
    }
    setTheme('light');

    document.cookie = `${constants.COOKIE_THEME}=light`;
    router.refresh();
  }, [router]);

  return {
    theme,
    changeTheme
  };
};
```

Лістинг 7 – src/widgets/header/index.tsx

```
'use client';
import useMediaQuery from '@mui/material/useMediaQuery';
import { useTheme } from '@mui/material/styles';

import styles from '@/widgets/header/header.module.scss';

import LogoIcon from '@/shared/ui/icons/logo/logo';
import LogoMobileIcon from '@/shared/ui/icons/logo/logo-mobile';
import LogoMin from '@/shared/ui/icons/logo/logo-min';
import { Navbar } from '@/widgets/header/ui/navbar';
import { Options } from '@/widgets/header/ui/options';
import { Container, Stack } from '@mui/material';

export default function Header() {
  const theme = useTheme();
  const downMdL = useMediaQuery(theme.breakpoints.down('md_l'));
  const downMd = useMediaQuery(theme.breakpoints.down('md'));
  const upSmL = useMediaQuery(theme.breakpoints.up('sm_l'));

  return (
    <header className={styles.root}>
      <Stack
        alignItems="center"
        justifyContent="center"
        sx={theme => ({
          background: theme.palette.primary.main,
          height: upSmL ? 80 : 50,
          boxShadow: theme.shadows[24]
        })}
      >
        <Container
          maxWidth="xl"
          sx={() => ({
            height: '100%'
          })}
        >
          <Stack
            direction="row"
            alignItems="center"
            justifyContent="space-between"
            height="100%"
          >
            {downMd ? (
              <LogoMin />
            ) : (
              <>
                {downMdL ? (
                  <LogoMobileIcon
                    sx={theme => ({
                      width: 98,
                      height: 'auto'
                    })}
                  />
                ) : (
                  <LogoIcon
                    sx={{
                      width: 192
                    }}
                  />
                )}
              </>
```

```
            )}
            <Navbar />
            <Options />
          </Stack>
        </Container>
      </Stack>
    </header>
  );
}
```

Лістинг 8 – src/widgets/header/ui/navbar.tsx

```
'use client';
import type { TAppRoutes, TRouteTypes } from '@/shared/types/routes';

import routes from '@/shared/config/routes';
import useMediaQuery from '@mui/material/useMediaQuery';
import { useCallback, useState } from 'react';
import { useTheme } from '@mui/material/styles';
import { usePathname } from 'next/navigation';

import { Close, Menu } from '@mui/icons-material';
import { Drawer, IconButton, paperClasses, Stack } from '@mui/material';
import { NavbarItem } from '@/shared/ui/navbar/navbar-item';
import { MobileMenu } from '@/widgets/header/ui/mobile-menu';

export const Navbar = () => {
  const pathname = usePathname();
  const userType: TRouteTypes = 'student'; //TODO: get data from the user
config

  const theme = useTheme();
  const upSmL = useMediaQuery(theme.breakpoints.up('sm_l'));

  const [burgerMenuState, setBurgerMenuState] = useState<boolean>(false);

  const toggleBurgerMenuClick = useCallback(
    (state: boolean) => {
      setBurgerMenuState(state);
    },
    [setBurgerMenuState]
  );

  return (
    <>
      {upSmL ? (
        <nav style={{ height: '100%' }}>
          <Stack direction="row" spacing={6} height="100%">
            {Object.keys(routes).map((key: string) => {
              const route = routes[key as TAppRoutes];

              if (
                route.navigationRoute &&
                (route.viewBy === 'all' || route.viewBy === userType)
              ) {
                return (
                  <NavbarItem
                    key={key}
                    route={route}
                    active={pathname === route.path}
                  />
                );
```

```
              }
            })}
          </Stack>
        </nav>
      ) : (
        <Stack width="100%" alignItems="flex-end" marginRight={4}>
          <IconButton
            aria-label="menu"
            onClick={() => toggleBurgerMenuClick(!burgerMenuState)}
            sx={theme => ({
              m: theme.spacing(-2),
              color: theme.palette.base.white
            })}
          >
            {burgerMenuState ? <Close /> : <Menu />}
          </IconButton>
          <Drawer
            anchor={'top'}
            open={burgerMenuState}
            hideBackdrop
            sx={() => ({
              [`.${paperClasses.root}`]: {
                top: 50,
                borderBottomLeftRadius: theme.spacing(2.5),
                borderBottomRightRadius: theme.spacing(2.5),
                boxShadow: theme.shadows[24]
              }
            })}
          >
            <MobileMenu
              routes={routes}
              userType={userType}
              pathname={pathname}
            />
          </Drawer>
        </Stack>
      )}
    </>
  );
};
```

Лістинг 9 – src/widgets/header/ui/options.tsx

```
'use client';
import { LocaleSwitcher } from '@/shared/ui/options/locale-switcher';
import { ThemeSwitcher } from '@/shared/ui/options/theme-switcher';
import { NotificationMenu } from '@/shared/ui/options/notification-menu';
import { AccountMenu } from '@/shared/ui/options/account-menu';
import { Stack } from '@mui/material';
import { useTheme } from '@mui/material/styles';
import useMediaQuery from '@mui/material/useMediaQuery';

export const Options = () => {
  const theme = useTheme();
  const upSmL = useMediaQuery(theme.breakpoints.up('sm_l'));

  return (
    <Stack
      direction="row"
      alignItems="center"
      spacing={{ lg: 6, md: 6, sm_l: 6, sm: 4, xs: 4 }}
      sx={theme => ({
```

```
        [theme.breakpoints.down('xs')]: {
          gap: theme.spacing(4)
        }
      })}
    >
      {upSmL && <LocaleSwitcher />}
      <NotificationMenu />
      {upSmL && <ThemeSwitcher />}
      <AccountMenu />
    </Stack>
  );
};
```

Лістинг 10 – src/widgets/header/ui/mobile-menu.tsx

```
'use client';
import type { MobileMenuProps } from '@/widgets/header/types/mobile-
menu';
import type { TAppRoutes } from '@/shared/types/routes';

import { Stack } from '@mui/material';
import { NavbarItem } from '@/shared/ui/navbar/navbar-item';

export const MobileMenu = (props: Readonly<MobileMenuProps>) => {
  const { routes, userType, pathname } = props;

  return (
    <Stack padding={4} alignItems="flex-start" spacing={3}>
      {Object.keys(routes).map((key: string) => {
        const route = routes[key as TAppRoutes];

        if (
          route.navigationRoute &&
          (route.viewBy === 'all' || route.viewBy === userType)
        ) {
          return (
            <NavbarItem
              key={key}
              route={route}
              active={pathname === route.path}
              mobile={true}
            />
          );
        }
      })}
    </Stack>
  );
};
```

Лістинг 11 – src/features/auth/ui/auth-form/index.tsx

```
'use client';
import type {
  TAuthFormSchema,
  IAuthFormProps
} from '@/features/auth/model/auth-form';
import type { IShowPassword } from '@/features/auth/types';

import styles from '@/features/auth/ui/auth-form/auth-form.module.scss';

import routes from '@/shared/config/routes';
import { useForm } from 'react-hook-form';
import { useState, useTransition } from 'react';
```

```
import { useTranslations } from 'next-intl';
import { zodResolver } from '@hookform/resolvers/zod';
import {
  authFormDefaultValues,
  loginFormSchema,
  registrationFormSchema
} from '@/features/auth/model/auth-form';

import Link from 'next/link';
import {
  Button,
  IconButton,
  InputAdornment,
  Stack,
  TextField,
  Typography
} from '@mui/material';
import { Visibility, VisibilityOff } from '@mui/icons-material';

export const AuthForm = (props: Readonly<IAuthFormProps>) => {
  const { type: formType, onSubmit } = props;

  const t = useTranslations('AuthForm');
  const [isAuthorizationTransition, startAuthorizationTransition] =
    useTransition();
  const [showPassword, setShowPassword] = useState<IShowPassword>({
    password: false,
    re_password: false
  });

  const handleClickShowPassword = (key: keyof IShowPassword) => {
    setShowPassword(prev => ({
      ...prev,
      [key]: !prev[key]
    }));
  };

  const {
    register,
    formState: { errors },
    handleSubmit
  } = useForm<TAuthFormSchema>({
    mode: 'onSubmit',
    resolver: zodResolver(
      formType === 'login' ? loginFormSchema : registrationFormSchema
    ),
    defaultValues: authFormDefaultValues
  });

  return (
    <form
      onSubmit={handleSubmit(data => {
        startAuthorizationTransition(async () => {
          // @ts-ignore TODO: need to update TS property
          await onSubmit(data);
        });
      })}
    >
      <Stack spacing={1}>
        <fieldset
          className={
```

```
            formType === 'login'
              ? styles.fieldset_login
              : styles.fieldset_register
          }
        >
          <label htmlFor="email" className={styles.field__root_full}>
            <Typography variant="label">{t('Email')}</Typography>
            <TextField
              {...register('email')}
              id="email"
              size="medium"
              type="email"
              variant="filled"
              helperText={errors.email?.message &&
t(errors.email?.message)}
              error={!!errors.email}
              placeholder={'your@email.com'}
              fullWidth
            />
          </label>
          {formType === 'register' && (
            <>
              <label htmlFor="first_name" className={styles.field__root}>
                <Typography variant="label">{t('First
name')}</Typography>
                <TextField
                  {...register('first_name')}
                  id="first_name"
                  size="medium"
                  type="text"
                  variant="filled"
                  helperText={
                    'first_name' in errors &&
                    errors.first_name?.message &&
                    t(errors.first_name?.message)
                  }
                  error={!!('first_name' in errors && errors.first_name)}
                  placeholder={t('First name')}
                  fullWidth
                />
              </label>
              <label htmlFor="last_name" className={styles.field__root}>
                <Typography variant="label">{t('Last name')}</Typography>
                <TextField
                  {...register('last_name')}
                  id="last_name"
                  size="medium"
                  type="text"
                  variant="filled"
                  helperText={
                    'last_name' in errors &&
                    errors.last_name?.message &&
                    t(errors.last_name?.message)
                  }
                  error={!!('last_name' in errors && errors.last_name)}
                  placeholder={t('Last name')}
                  fullWidth
                />
              </label>
            </>
          )}
```

```
            <label htmlFor="password" className={styles.field__root}>
              <Typography variant="label">{t('Password')}</Typography>
              <TextField
                {...register('password')}
                id="password"
                size="medium"
                type={showPassword.password ? 'text' : 'password'}
                variant="filled"
                helperText={
                  errors.password?.message && t(errors.password?.message)
                }
                error={!!errors.password}
                placeholder={t('Password')}
                fullWidth
                slotProps={{
                  input: {
                    endAdornment: (
                      <InputAdornment position="end">
                        <IconButton
                          aria-label={
                            showPassword.password
                              ? 'Hide the password'
                              : 'Display the password'
                          }
                          onClick={() =>
handleClickShowPassword('password')}
                          edge="end"
                        >
                          {!showPassword.password ? (
                            <VisibilityOff />
                          ) : (
                            <Visibility />
                          )}
                        </IconButton>
                      </InputAdornment>
                    )
                  }
                }}
              />
            </label>
            {formType === 'register' && (
              <label htmlFor="re_password" className={styles.field__root}>
                <Typography variant="label">{t('Repeat
password')}</Typography>
                <TextField
                  {...register('re_password')}
                  id="re_password"
                  size="medium"
                  type={showPassword.re_password ? 'text' : 'password'}
                  variant="filled"
                  helperText={
                    're_password' in errors &&
                    errors.re_password?.message &&
                    t(errors.re_password?.message)
                  }
                  error={!!('re_password' in errors && errors.re_password)}
                  placeholder={t('Repeat password')}
                  fullWidth
                  slotProps={{
                    input: {
                      endAdornment: (
```

```
                        <InputAdornment position="end">
                          <IconButton
                            aria-label={
                              showPassword.re_password
                                ? 'Hide the password'
                                : 'Display the password'
                            }
                            onClick={() =>
handleClickShowPassword('re_password')}
                            edge="end"
                          >
                            {!showPassword.re_password ? (
                              <VisibilityOff />
                            ) : (
                              <Visibility />
                            )}
                          </IconButton>
                        </InputAdornment>
                      )
                    }
                  }}
                />
              </label>
            )}
          </fieldset>
          {formType === 'login' && (
            <Typography
              textAlign="end"
              variant="label"
              sx={theme => ({
                ['a']: {
                  color: theme.palette.text.primary
                }
              })}
            >
              <Link href={routes.passwordReset.path} prefetch={true}>
                {t('Forgot your password?')}
              </Link>
            </Typography>
          )}
        </Stack>
        <Stack marginTop={6} spacing={1}>
          <Button
            type="submit"
            variant="contained"
            size="large"
            fullWidth
            loading={isAuthorizationTransition}
            loadingPosition="end"
          >
            {formType === 'login' ? t('Log in') : t('Sign up')}
          </Button>
          <Typography
            variant="label"
            sx={theme => ({
              ['a']: {
                color: theme.palette.text.primary
              }
            })}
          >
            {formType === 'register' ? (
```

```
            <>
              {t('Already have an account?')}{' '}
              <Link href={routes.login.path} prefetch>
                {t('Log in')}
              </Link>
            </>
          ) : (
            <>
              {t("Don't have account?")}{' '}
              <Link href={routes.register.path} prefetch>
                {t('Register now')}
              </Link>
            </>
          )}
        </Typography>
      </Stack>
    </form>
  );
};
```

Лістинг 12 – src/features/auth/ui/password-reset-form/index.tsx

```
"use client"
import type { IPasswordResetFormProps, TPasswordResetFormSchema } from
'@/features/auth/model/password-reset-form';

import styles from '@/features/auth/ui/password-reset-form/password-
reset-form.module.scss';

import { useTranslations } from 'next-intl';
import { useTransition } from 'react';
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import { passwordResetFormSchema, passwordResetFormDefaultValues } from
'@/features/auth/model/password-reset-form';

import { Button, Stack, TextField, Typography } from '@mui/material';

export const PasswordResetForm = (props:
Readonly<IPasswordResetFormProps>) => {
  const {
    onSubmit
  } = props;

  const t = useTranslations('PasswordResetForm');
  const [isActionTransition, startActionTransition] = useTransition();

  const {
    register,
    formState: { errors },
    handleSubmit
  } = useForm<TPasswordResetFormSchema>({
    mode: 'onSubmit',
    resolver: zodResolver(passwordResetFormSchema),
    defaultValues: passwordResetFormDefaultValues
  });

  return (
    <form
      onSubmit={handleSubmit((data) => {
        startActionTransition(async () => {
          await onSubmit(data);
```

```
        })
      })}
    >
      <Stack
        spacing={{ lg: 8, md: 8, sm: 6, xs: 4 }}
        sx={(theme) => ({
          [theme.breakpoints.down('xs')]: {
            gap: theme.spacing(4),
          }
        })}
      >
        <fieldset>
          <label htmlFor="email" className={styles.field__root}>
            <Typography
              variant='label'
            >
              {t('Email')}
            </Typography>
            <TextField
              {...register('email')}
              id='email'
              size='medium'
              type='email'
              variant='filled'
              helperText={errors.email?.message &&
t(errors.email?.message)}
              error={!!errors.email}
              placeholder={'your@email.com'}
              fullWidth
            />
          </label>
        </fieldset>
        <Button
          type='submit'
          variant='contained'
          size='large'
          fullWidth
          loading={isActionTransition}
          loadingPosition='end'
        >
          { t('Request password reset') }
        </Button>
      </Stack>
    </form>
  )
}
```

Лістинг 13 – src/features/auth/ui/social-buttons.tsx

```
"use client"
import type { ISocialType, TSocialTypes } from '@/features/auth/types';

import { useState, useTransition } from 'react';
import { useTranslations } from 'next-intl';
import { continueWithFacebook, continueWithGoogle } from
'@/shared/utils/continue-wth-social-login';

import GoogleIcon from '@/shared/ui/icons/socials/google';
import FacebookIcon from '@/shared/ui/icons/socials/facebook';
import { Button, Stack, SvgIcon, buttonClasses } from '@mui/material';

export const SocialButtons = (props: Readonly<{ type: 'login' |
```

```
'register' }>) => {
  const { type: formType } = props;

  const t = useTranslations('SocialButtons');
  const [activeAction, setActiveAction] =
useState<TSocialTypes>('google');
  const [isAuthorizationTransition, startAuthorizationTransition] =
useTransition();

  const handleClickButton = async (data: ISocialType) => {
    setActiveAction(data.type);

    startAuthorizationTransition(async () => {
      switch (data.type) {
        case 'google': {
          await continueWithGoogle();
          break;
        }
        case 'facebook': {
          await continueWithFacebook();
          break;
        }
      }
    })
  }

  return (
    <Stack
      direction='row'
      sx={(theme) => ({
        rowGap: theme.spacing(6),
        columnGap: theme.spacing(11),
        [theme.breakpoints.down('md')]: {
          columnGap: theme.spacing(6)
        },
        [theme.breakpoints.down('sm_l')]: {
          flexDirection: 'column'
        }
      })}
    >
      <Button
        type='button'
        variant='contained'
        startIcon={<SvgIcon inheritViewBox><GoogleIcon/></SvgIcon>}
        fullWidth
        loading={isAuthorizationTransition && activeAction === 'google' }
        loadingPosition='end'
        onClick={() => handleClickButton({
          type: 'google'
        })}
        sx={(theme) => ({
          color: theme.palette.text.primary,
          backgroundColor: theme.palette.background.default,
          [`& .${buttonClasses.startIcon}`]: {
            ['>*:nth-of-type(1)']: {
              fontSize: theme.typography.pxToRem(28)
            }
          },
          [theme.breakpoints.down('sm')]: {
            fontSize: theme.typography.pxToRem(14)
          }
```

```
        })}
      >
        {formType === 'login' ? t('Log in with Google') : t('Sign up with
Google')}
      </Button>
      <Button
        type='button'
        variant='contained'
        startIcon={<SvgIcon inheritViewBox><FacebookIcon/></SvgIcon>}
        fullWidth
        loading={isAuthorizationTransition && activeAction ===
'facebook'}
        loadingPosition='end'
        onClick={() => handleClickButton({
          type: 'facebook'
        })}
        sx={(theme) => ({
          color: theme.palette.text.primary,
          backgroundColor: theme.palette.background.default,
          [`& .${buttonClasses.startIcon}`]: {
            ['>*:nth-of-type(1)']: {
              fontSize: theme.typography.pxToRem(28)
            }
          },
          [theme.breakpoints.down('sm')]: {
            fontSize: theme.typography.pxToRem(14)
          }
        })}
      >
        {formType === 'login' ? t('Log in with Facebook') : t('Sign up
with Facebook')}
      </Button>
    </Stack>
  )
}
```