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

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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Бакалавр
(назва освітнього ступеня)

на тему: Проєктування та розробка системи тайм-менеджменту з використанням фреймворку Vue

Виконав(ла): студент(ка) 4 курсу, групи СП-41
спеціальності 121 «Інженерія програмного Забезпечення»
(шифр і назва спеціальності)

(підпис) Грицишин П. А. (прізвище та ініціали)

Керівник (підпис) Мудрик І. Я. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю. М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2024

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

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

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М. Р.
(підпис) (прізвище та ініціали)
« » 2024 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Грицишину Павлу Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Проєктування та розробка системи тайм-менеджменту з використанням фреймворку Vue

Керівник роботи Мудрик Іван Ярославович, доктор філософії, старший викладач кафедри ПІ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи Методичні вказівки; Технічна документація для: JavaScript, Vue, Python, Django, Vue-router, Pinia, Celery.

4. Зміст роботи (перелік питань, які потрібно розробити)
Огляд конкурентів і порівняльний аналіз, обґрунтування вибору напрямку дослід ження, вибір методології розробки, формування вимог до системи, проєктування відношень між акторами та прецедентами, визначення архітектури системи, проєктування головних класів системи, проєктування структури веб-застосунку, конструювання серверної частини, конструювання клієнтської частини, тестування та верифікація вимог, долікарська допомога при уражен ні струмом, питання безпеки при експлуатації електроустановок високих, надвисоких та ультрависоких частот

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Діаграма варіантів використання, діаграма класів, діаграма структури веб-застосунку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності,	Лазарюк В. В., к.т.н., доц., доц. каф. МТ		
основи охорони праці			
Нормоконтроль	Стоянов Ю. М., к.т.н.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	**Примітка**
1.	Вибір та затвердження теми		**Виконано**
2.	Аналіз предметної області та технічного завдання		**Виконано**
3.	Вибір методології розробки		**Виконано**
4.	Формування вимог до системи та проєктування		**Виконано**
	відношень.		
5.	Визначення архітектури системи, проєктування		**Виконано**
	головних класів та структури веб-застосунку		
6.	Конструювання серверної частини		**Виконано**
7.	Конструювання клієнтської частини		**Виконано**
8.	Тестування та верифікація вимог		**Виконано**
9.	Безпека життєдіяльності, основи охорони праці		**Виконано**
10.	Підготовка презентації та доповіді		**Виконано**
11.	Нормоконтроль		**Виконано**
12.	Перевірка на плагіат		**Виконано**
13.	Попередній захист кваліфікаційної роботи		**Виконано**
14.	Захист кваліфікаційної роботи		**Виконано**

Студент _____ *Грицишин П. А.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Мудрик І. Я.*
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Кваліфікаційна робота бакалавра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення». ТНТУ, 2024, Сторінок 86, таблиць 1, рисунків 48, презентація.

Тема: Проєктування та розробка системи тайм-менеджменту з використанням фреймворку Vue.

У даній кваліфікаційні роботі бакалавра продемонстровано процес проєктування та розробки системи для тайм-менеджменту. Систему реалізовано у якості веб-сайту, що обґрунтовується доступністю та зручністю для всіх користувачів, відносною простотою реалізації та гнучкістю при майбутньому масштабуванні системи. Було проведено аналіз конкурентів, які надають схожі послуги, виділені переваги даних систем, а також взяті до уваги недоліки користування цими платформами. У результаті проробленої роботи було спроєктовано базу даних для цього проєкту, спроєктовано архітектуру всього веб-сайту. За допомогою фреймворку Vue було реалізовано клієнтську частину сайту. Завдяки використання цього фреймворку було впроваджено реактивність для веб-сайту, що в результаті вплинуло на досвід користування сайтом та його показники швидкості. Також в даній роботі описано процес створення серверної частини. Для цього використовувалась мова програмування Python та фреймворк Django.

Ключові слова: веб-сайт, тайм-менеджмент, проєктування бази даних, архітектура веб-сайту, JavaScript, Vue3, Python, Django.

ABSTRACT

Bachelor's qualification work. Ternopil National Technical University named after Ivan Puluj, Department of Software Engineering, specialty 121 "Software Engineering". TNTU, 2024, Pages 86, tables 1, images 48, presentation.

Topic: Design and development of a time management system using the Vue framework.

This bachelor's thesis demonstrates the process of designing and developing a time management system. The system is implemented as a website, which is justified by accessibility and convenience for all users, relative ease of implementation, and flexibility for future scaling of the system. An analysis of competitors providing similar services was conducted, the advantages of these systems were highlighted, and the disadvantages of using these platforms were taken into account. As a result of the work done, a database was designed for this project, and the architecture of the entire website was designed. The client side of the website was implemented using the Vue framework. Thanks to the use of this framework, responsiveness was implemented for the website, which ultimately affected the user experience and speed of the website. This paper also describes the process of creating the server side. The Python programming language and Django framework were used for this purpose.

Keywords: website, time management, database design, website architecture, JavaScript, Vue3, Python, Django.

ЗМІСТ

ВСТУП

Тайм-менеджмент - це набагато більше, ніж просто індивідуальна навичка або питання особистої дисципліни. Це фундаментальна основа для забезпечення безперебійної роботи та загального успіху бізнесу. У світі, де час є дорогоцінним і обмеженим ресурсом, вміння ефективно ним управляти стало імперативом для організацій, які прагнуть процвітати в постійно мінливому конкурентному середовищі.

Важливість часу в бізнесі не можна недооцінювати. Поганий тайм-менеджмент може призвести до затримок, помилок, втрачених можливостей і зниження продуктивності. З іншого боку, ефективний тайм-менеджмент допомагає оптимізувати повсякденну діяльність, максимізувати потенціал співробітників і підвищити загальну ефективність бізнесу.

Ефективний тайм-менеджмент базується на вмінні планувати, визначати пріоритети, організовувати та розумно розподіляти часові ресурси. Це означає прийняття обґрунтованих рішень щодо того, як витрачати час та енергію на завдання та проекти, які мають найбільший вплив та цінність для бізнесу. Оптимізуючи використання часу, компанії можуть підвищити продуктивність, швидше досягати своїх цілей і залишатися конкурентоспроможними на ринку.

Ефективний тайм-менеджмент також має відчутні переваги для працівників. Він дозволяє їм краще збалансувати своє професійне та особисте життя, зменшуючи стрес і сприяючи здоровому балансу. Коли працівники відчувають себе менш перевантаженими завданнями і дедлайнами, вони більш мотивовані, віддані і задоволені своєю роботою, що призводить до підвищення продуктивності та ефективності.

Коли мова йде про індивідуальну продуктивність, ефективний тайм-менеджмент відіграє ключову роль. Зменшуючи відволікання та прокрастинацію, ми можемо зосередитися на важливих завданнях і не витрачати дорогоцінний час на несуттєві види діяльності. Прокрастинація, яку часто вважають ворогом тайм-

менеджменту номер один, може мати значний вплив на продуктивність та ефективність компанії. Прокрастинація характеризується відкладанням або затримкою виконання завдань, що призводить до втрати цінного часу та накопичення не виконаної роботи.

Причиною може бути відсутність мотивації, страх невдачі, труднощі в управлінні пріоритетами або просто брак дисципліни. Якою б не була причина, наслідки часто бувають шкідливими. Затримки у виконанні завдань можуть призвести до затримки проекту, зриву дедлайнів і зниження якості роботи.

Подолати дані трудності і є покликанням створеної платформи. У неї закладено ідею самоконтролю та самоорганізованості. На відміну від корпоративних інструментів, вона надає приватні та націлені саме на вас інструменти та можливості. Використовуючи дану платформу, ви можете спокійно поєднувати різноманітні задачі, відстежувати хід проєктів, над якими ви працюєте.

За допомогою гнучкої та універсальної структури, ви можете створювати окремі завдання або ж поєднувати їх у списки. Дана платформа передбачає усі можливості для використання під ваші різноманітні потреби. Її можна використовувати як для побутових справ, так і для цілих робочих проєктів.

У процесі створення цієї платформи було задіяно сучасний стек технологій, який гарантує високу продуктивність, надійність та зручність користування.

Для клієнтської частини, тобто тієї, з якої користувач буде безпоседеньо взаємодіяти під час користування платформою, було обрано прогресивний JavaScript фреймворк Vue.js. Vue.js дозволив створити динамічний, інтерактивний та зручний інтерфейс, який миттєво реагує на будь-які дії користувача. Серверну частину платформи було реалізовано за допомогою Django - потужного та популярного Python-фреймворку. Django забезпечує високу швидкість роботи, надійність та безпеку, а також дозволяє легко масштабувати платформу в міру її зростання.

Саме вирішення вище описаних проблем та результати проведених досліджень та робіт з використанням таких передових технологій я б і хотів висвітили у даній роботі.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд конкурентів

Доволі популярним рішенням на сучасному ринку сервісів для тайм-менеджменту є продукт від світового гіганта Microsoft – Microsoft To-do. Корпорація Microsoft створила зручний та потужний інструмент для контролю завдань користувачів. Даний сервіс покликаний допомогти людям організувати їхні повсякденні справи та збільшити продуктивність.

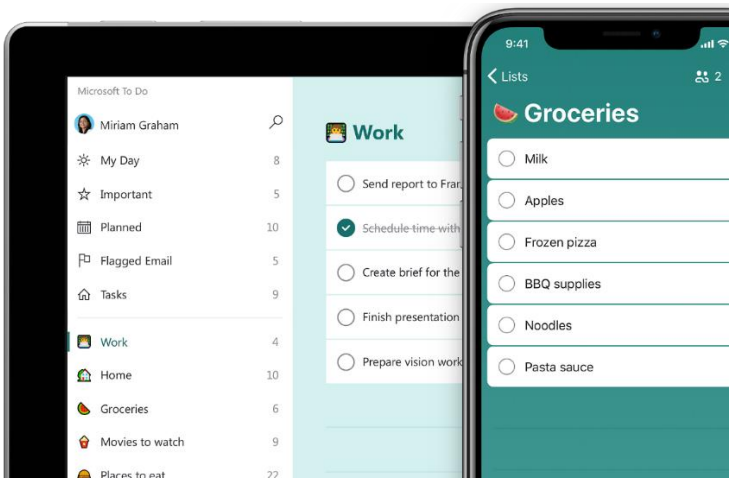

Рис. 1.1 – Вигляд сервісу Microsoft To-do [1].

Платформа пропонує широкий спектр можливостей і послуг з простим та інтуїтивним інтерфейсом. Користувачі можуть створювати завдання, групувати їх у списки для організації робочих процесів або домашніх справ. Сервіс доступний на смартфонах, персональних комп'ютерах і в браузері, з синхронізацією даних через обліковий запис.

Додатково, можна встановлювати терміни виконання завдань, що допомагає у плануванні та контролі. Списки можна ділити з іншими користувачами, що зручно для командної роботи, а функція нагадувань запобігає забуттю важливих подій. Є можливість кастомізації з вибором кольорів, тем і фонів.Отже, можна зробити висновок, що продукт від корпорації Microsoft є зручним рішенням на

кожен день та гнучкий у своєму використанні, тому при створенні аналогу, варто орієнтуватися на переваги цього конкуренту та старатися нівелювати його недоліки.

Іншим конкурентом можна розглядати платформу Trello. Trello — це візуальний інструмент, що дає змогу команді керувати різноманітними проєктами й робочими процесами та відстежувати виконання завдань [2].

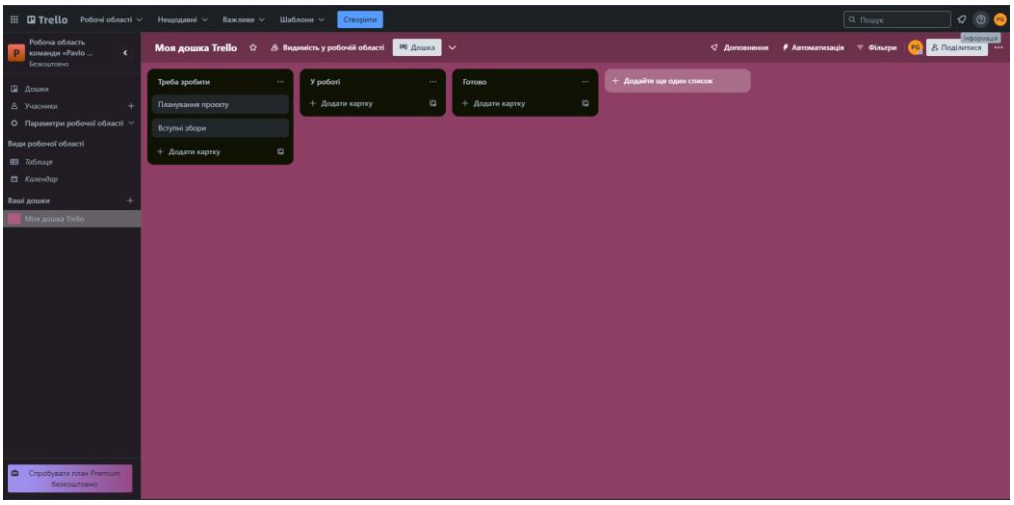

Рис. 1.2 – Вигляд платформи Trello.

Як і попередня платформа, Trello має у собі широкий функціонал, який здатен задовольнити велику масу потреб користувачів. Організація роботи тут базується на використанні дошок, усередині яких користувач можеш створювати списки та, власне, самі завдання. Якщо ж копнути глибше, можна зрозуміти наступний принцип роботи та використання вищезгаданих можливостей: дошки являють собою проєкт, над яким працює користувач, або ж просто комплексне і велике завдання. Уже всередині дошки можна посортувати завдання, розбивши їх на списки, що допомагає в організації та стуктуризації завдань. А, власне, усередину списку можна додати картку, яка і буде нашим завданням. Завдання можна розбити на підзавдання.

На перший погляд структура має дещо складний вигляд, проте добре підходить для організації великої кількості роботи та людей. Гадаю, це й можна виділити як одну з переваг даної платформи – вона гарно підходить для організації

роботи. Крім того, тут користувач має можливість запрошувати інших користувачів до своєї дошки для спільної праці над чимось єдиним. Система приватності у Trello дозволяє гнучко налаштовувати доступи як до дошок, так і до окремих списків. Це буде корисно для компаній з багатьма співробітниками, яка дбає про конфіденційність, тому це дозволить запобігти витокам інформації, якщо певні етапи чи функції продукту тримають у таємниці, а також і для самих працівників це буде плюсом.

Платформа має привабливий дизайн, вирізняється зручністю та інтуїтивним інтерфейсом у стилі мінімалізму. Основний акцент зроблено на картах, що робить взаємодію зручнішою для користувача.

Можна змінювати порядок списків та завдань за допомогою перетягування, а також переносити завдання між списками, змінюючи їхню структуру. Для швидкого ідентифікування на картки можна додавати кольорові теги та мітки, що спрощує пошук конкретних завдань. Це особливо корисно для команд, які працюють над великими проєктами з великою кількістю завдань.

Додатково, Trello надає широкий перелік фільтрів для сортування та фільтрації завдань, що дозволяє користувачеві зосередитися на найважливішому.Платформа Trello також доступна на багатьох пристроях. На відміну від Microsoft To-do вона не має програми для персонального комп’ютера, а є веб-сервісом. Проте також існує й мобільний застосунок для IOS та Android. На мою думку, найзручнішим все ж користуватися Trello саме на персональному комп’ютері, адже на маленькому екрані телефону не надто зручно переглядати велику кількість інформації, яку зосереджено в дошках. Однак, незважаючи на це, розробники Trello зробили свій сайт адаптивним і для мобільних пристроїв, якщо людина не хоче або не може зараз скористатися мобільним застосунком.

1.2 Порівняльний аналіз конкурентів

У попередньому розділі ми розглянули два основних конкуренти для мого проєкту: Microsoft To-do та Trello. Для розуміння, в якому напрямку рухатися, у цьому розділі ми детально розглянемо переваги та недоліки цих платформ, як вони відповідають сучасним вимогам та відгуки користувачів. Порівняльний аналіз цих двох платформ дасть чітке розуміння, чого варто уникати при розробці платформи, яка буде їхнім конкурентом, та на які рішення варто рівнятися.

- Що ж, для початку розглянемо переваги сервісу Microsoft To-do. Серед них можна виділити наступні:
- Інтеграція з Microsoft: сервіс легко інтегрується з Outlook, OneNote, Teams та іншими інструментами Microsoft.
- Інтуїтивний інтерфейс: простий і зрозумілий, не потребує попереднього досвіду.
- Функція "Мій день": допомагає зосередитися на завданнях поточного дня, підвищуючи продуктивність.
- Персоналізація: можливість змінювати колір і фон інтерфейсу, полегшуючи пошук необхідного.
- Кросплатформеність: доступний на Windows, iOS, Android і в браузері; синхронізація змін на всіх пристроях.
- Організація завдань: створення списків для структуризації завдань.

Далі варто виділити недоліки та спірні рішення, які є присутніми в даному сервісі. Серед таких можна навести кілька:

- Обмежена функціональність: базовий набір функцій, бракує глибшої аналітики, фільтрування та сортування.
- Мало інтеграцій: інтеграція тільки з сервісами Microsoft, без підтримки інших сервісів.

- Обмежена спільна робота: недостатньо функцій для ефективної командної роботи над проєктами.
- Скорочений функціонал у браузері: у браузерній версії менше функцій порівняно з нативними додатками.

Отже, Microsoft To-Do – це зручний інструмент для керування завданнями, який особливо добре підійде для користувачів, що вже використовують інші продукти Microsoft. Його простота, інтуїтивний інтерфейс та інтеграція з екосистемою Microsoft роблять його ефективним для організації та управління простими завданнями. Однак для дещо складніших завдань та проєктів або командної роботи цього рішення буде недостатньо.

Тепер на черзі платформа Trello. Серед переваг даного сервісу можна виділити наступні:

- Організація інтерфейсу: платформа використовує карти, які можна перетягувати та розміщувати у будь-якому порядку.
- Інтуїтивність: інтерфейс спрощує роботу завдяки найкращим практикам UI/UX дизайну.
- Персоналізація: можливість змінювати колір фону, тему і додавати кольорові мітки.
- Спільне користування: можна запрошувати інших користувачів для спільного менеджменту завдань і коментування.
- Безпека: гнучкі налаштування доступу до дошки, списків або карток.
- Інтеграції: підтримка інтеграцій з Slack, Google Drive, Jira тощо.
- Розширення: "Power-Ups" для розширення функціоналу та відкрите API для створення власних розширень.
- Автоматизація: інструмент "Butler" для автоматизації рутинних процесів.
- Кросплатформеність: доступний у браузерах та на смартфонах (iOS, Android).

- Зміни в реальному часі: користувачі бачать зміни одне одного без перезавантаження сторінки.
- Сповіщення: отримання сповіщень про коментарі, виконані завдання, і пуш-сповіщення на мобільному додатку.

Попри вражаючу кількість позитивних сторін, Trello також має і недоліки. Серед недоліків даної платформи можна виділити наступні:

- Виклики масштабування: структура Trello може бути непридатною для великих команд і проєктів.
- Залежність від інтеграцій: додаткові можливості часто потребують сторонніх інтеграцій, що може ускладнити робочий процес.
- Обмежена налаштування: бракує гнучкості для специфічних робочих процесів.
- Складність для новачків: інтерфейс може бути складним для нетехнічних користувачів.
- Перевантаження інформацією: багато інформації на картках може захаращувати інтерфейс.
- Умовна безкоштовність: базова версія обмежена, багато корисних функцій доступні лише у платній версії

Trello — це ефективний інструмент для організації роботи та проектів, особливо зручний для візуального планування та командної співпраці. Він вирізняється простотою використання, гнучкістю налаштування, можливостями інтеграції та автоматизації.

Однак для великих, багатогранних проектів, які потребують детальної звітності, можуть знадобитися спеціалізовані інструменти управління проектами.

1.3 Обґрунтування вибору напрямку дослідження

Перш за все важливо визначити, що буде з себе являти майбутня платформа, адже від цього значною мірою буде залежати як вибір архітектурного рішення, так і охоплення цільової авдиторії. Беручи до уваги потенційних конкурентів, зазвичай в них було кілька продуктів, які синхронізувалися з єдиною базою даних. Таке рішення, можливо, підходить великим компаніям, проте в даній ситуації воно буде невиправдано трудомістким.

Розробка нативного застосунку для персональних комп'ютерів чи телефонів було б непоганим рішенням, якби можна було зі всією впевненістю заявити, що цільова авдиторія переважно користується тим чи іншим пристроєм для відстежування завдань та тайм-менеджменту. Однак тут ситуація відрізняється. Майбутньою системою можна буде користуватися як для менеджменту робочих справ, так і хатніх. І, думаю, легко погодитися, що для роботи було б зручніше мати усе на ноутбуці чи ПК, але тоді відстеження хатніх справ стає незручним. Золотою серединою в цьому питання може стати веб-застосунок.

Веб-застосунок - це програмне забезпечення, що працює на принципах клієнт-серверної взаємодії, де клієнтська частина представлена браузером, а серверна - веб-сервером. Логіка веб-застосунку розподілена між цими двома компонентами, обмін інформацією відбувається через API.

Веб-застосунки виникли як наступний етап у розвитку веб-сайтів, відрізняючись вищим рівнем інтерактивності, можливістю інтеграції та системами автентифікації. Кожен веб-застосунок складається з клієнтської частини (Front-end), серверної частини (Back-end) та бази даних.

Клієнтська частина – це інтерфейс, з яким взаємодіє користувач. Вона отримує дані від серверної частини через API, відображає їх і реагує на дії користувачів. Серверна частина обробляє запити від клієнта, взаємодіє з базою даних і надсилає відповіді. База даних зберігає всю необхідну інформацію для роботи веб-застосунку.

Front-End – один з найпопулярніших термінів у веб-розробці. Усе, що можна побачити в Інтернеті, включно з текстом, зображеннями, відео, аудіо, графікою тощо, називається Front-End'ом. Front-End – це та частина, з якою користувачі взаємодіють безпосередньо. Іншими слова це можна назвати «клієнтською частиною» [3].

У той час як Front-end розробка зосереджена на проектуванні та створенні користувацького інтерфейсу, з яким взаємодіють користувачі, Back-end розробка працює за лаштунками, обробляючи логіку на стороні сервера та управління базою даних, які забезпечують роботу всієї системи. По суті, Back-end розробка передбачає написання коду, який виконується на сервері і взаємодіє з інтерфейсом, забезпечуючи динамічний та інтерактивний веб-досвід [4].

Для динамічного відображення сторінок браузери використовують HTML, CSS та JavaScript. HTML структурує сторінку за допомогою тегів, таких як «div», «header», і «footer». CSS додає стилі, змінюючи кольори, шрифти, відступи та розміри елементів, роблячи сторінку привабливою та зручною.

JavaScript забезпечує інтерактивність і динамічність, маніпулюючи елементами DOM, надсилаючи запити до сервера та виконуючи обчислення. JavaScript є ключовою частиною сучасних веб-сайтів і застосунків, додаючи функціональність та інтерактивність до статичних сторінок.

Однак попри свої переваги та функціонал, який надає мова JavaScript, часто на вирішення деяких рутинних завдань йде надто багато часу або ж вони не надто оптимізовані. Як відомо, лінь – двигун прогресу, тому як правило, розробники створюють фреймоврки, які допомагають мінімалізувати рутинні завдання при розробці. Раніше клієнтська частина веб-сайтів була доволі простою. Від розробників не вимагалося багато знань, аби зверстати сторінку, вставити туди текст та картинку. Проте з часом користувацькі інтерфейси ставали все складнішими і складнішими. Результатом цього став поділ веб-розробників на дві течії: Front-end та Back-end розробників, де розробник займається своєю частиною веб-застосунку. Згодом ці течії почали розвиватися кожна у своєму напрямку, набуваючи все нові і нові технології, тому що для клієнтської частини, що для

серверної частини з’явилося безліч фреймворків, які допомагають та пришвидшують роботу розробника. Фреймворк, або програмний каркас, - це платформа, яка забезпечує основу для розробки програмних застосунків. Його можна уявити як шаблон робочої програми, який можна вибірково змінювати, додаючи код. Він використовує спільні ресурси – такі як бібліотеки, файли зображень і довідкові документи, і об'єднує їх в один пакет. Цей пакет можна змінювати відповідно до конкретних потреб проєкту. За допомогою фреймворку розробник може додавати або замінювати функції, щоб надати застосунку нову функціональність [5].

Від правильно вибраного фреймворку залежить чимало речей у реалізації того чи іншого проєкту. Цьому варто приділити багато уваги, адже якщо зробити хибний вибір, час реалізації проєкту може збільшитися, підтримка проєкту в майбутньому муже бути складною або можливість масштабування проєкту на обраних технологіях буде надто обмеженою, що не дасть продукту розвиватися.

На даному етапі розвитку клієнтська частина у своїй складності не поступається серверній. Перед Front-end розробником постає чимало викликів при розробці сучасних клієнтських інтерфейсів: красиві анімації, продуктивність, адаптивність, гнучкість. У залежності від проєкту на дотриманні усіх цих метрик, розробнику йде дедалі більше часу та зусиль, тому для клієнтської частини почали з’являтися фреймворки, з якими розробникам буде зручніше та швидше створювати нові інтерфейси. Згідно дослідження платформи Stack Overflow [6] станом на 2024 рік виділяють три найпопулярніших серед розробників фреймворки – це React, Angular та Vue.

Vue.js — це JavaScript-фреймворк, призначений для створення користувацьких інтерфейсів (UI) та односторінкових додатків (SPA). Як і більшість найкращих фреймворків, Vue.js має відкритий вихідний код. Його архітектура базується на патерні Model-view-viewmodel (MVVM).

Рис. 1.3.2 – Логотип фреймворку VueJS.

Vue.js, розроблений Еваном Ю і випущений у 2014 році, створений як легкий фреймворк, який зберігає найкращі риси AngularJS. Патерн MVVM у Vue.js розділяє бізнес-логіку (модель) і графічний інтерфейс (представлення), полегшуючи співпрацю між front-end і back-end розробниками. Основна бібліотека Vue.js орієнтована на рівень представлення, а додаткова функціональність реалізується за допомогою допоміжних бібліотек, що робить його прогресивним фреймворком.

Однією з ключових особливостей Vue.js є директиви, які розширюють можливості HTML і спрощують створення динамічних веб-застосунків. Vue.js відомий своєю простотою використання, поступовою адаптацією та легким вивченням навіть для початківців. Ядро бібліотеки базується на CSS, HTML та JavaScript.

Vue.js має чудові офіційні бібліотеки, такі як Vue Router для маршрутизації та Vuex для управління станами. Завдяки невеликому розміру (21 КБ) і віртуальному DOM, Vue.js забезпечує швидкий рендеринг, ефективно синхронізуючи зміни.

Вивчення Vue.js не вимагає глибоких знань інших бібліотек або специфічних варіацій JavaScript. Використання популярних редакторів коду, таких як Visual Studio або Webstorm, робить розробку ще простішою. Активна та віддана спільнота Vue.js підтримує його розвиток через форуми, чат Discord та понад 72 000 запитань на Stack Overflow, а також понад 132 000 користувачів на GitHub.

Попри численні переваги, Vue.js має свої недоліки, які слід враховувати при виборі фреймворку для проєкту.

Vue.js відомий своєю гнучкістю, але це може бути як перевагою, так і недоліком. Ця гнучкість дозволяє починати проєкти з нуля та додавати нові функції, але також може призвести до ускладнень у великих проєктах через потенційні помилки та невідповідності.

Іншим недоліком Vue.js є обмеженість його екосистеми порівняно з React чи AngularJS, особливо у кількості доступних плагінів та компонентів.

Vue.js також є відносно новою технологією з малою спільнотою, що може ускладнювати пошук відповідей на питання та отримання необхідної підтримки.

Для невеликого, але масштабованого проекту, який вимагає швидкості та простоти використання, фреймворк Vue.js є найкращим вибором. Він ефективний, швидкий і підходить як для простих, так і для масштабованих проектів. Мій досвід роботи з Vue.js підтверджує, що це найкращий вибір для клієнтської частини платформи управління часом.

Використання серверних фреймворків має безліч переваг для розробки веб-застосунків. Вони забезпечують безпеку застосунку, маючи захист від зловмисних дій, а також спрощують підтримку та розширення застосунку завдяки грамотній архітектурі. Використання фреймворків також дозволяє швидше розробляти, оскільки вони надають готовий функціонал для базових речей у веб-розробці, що сприяє високій швидкості розробки.

Царина фреймворків для Back-end’у значно обширніша, ніж для Front-end’у, і на відміну від нього, фаворитів там справді багато. Згідно дослідження платформи «Statistics & Data» на тему найпопулярніших фреймворків для Back-end розробки на 2023 рік [7] (див. рис. 1.3.1), можна виділити 3 найпопулярніші фрейморки. Це Laravel, Django та Spring.

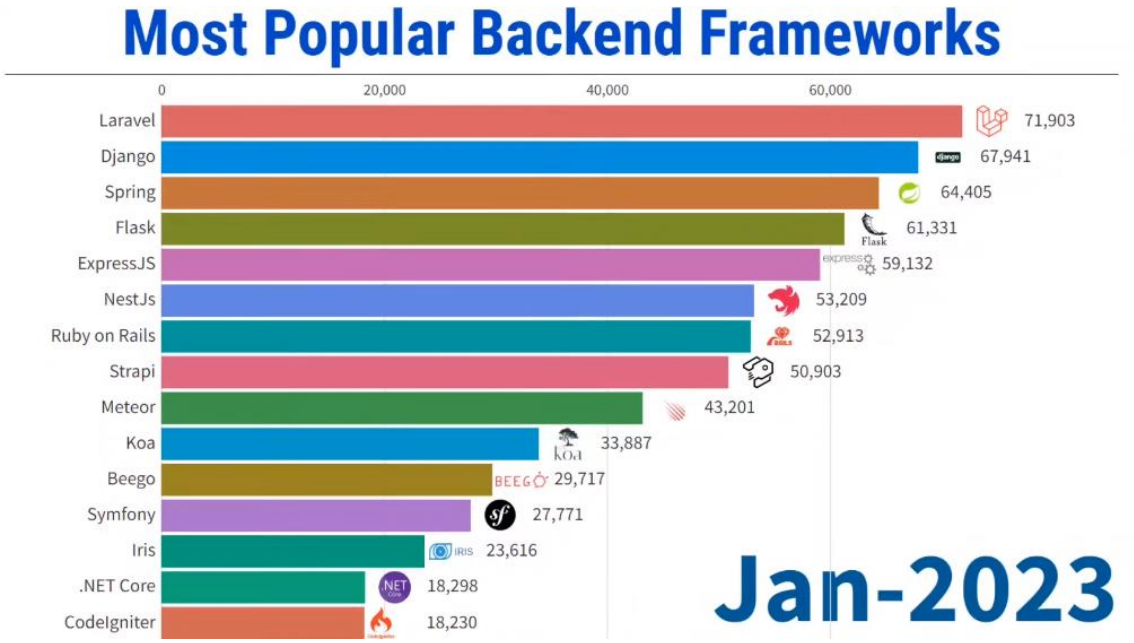

Рис. 1.3.4 – Найпопулярніші Back-end фреймворки станом на січень 2023 року.

Django є одним з найпопулярніших веб-фреймворків у світі. Підтверджуючи це звання на його офіційному сайті можна знайти перелік всесвітньо відомих партнерів, таких як Instagram, YouTube, Google та NASA використовують Django для своїх веб-сайтів. Це підкреслює надійність та ефективність фреймворку як у роботі з простими проєктами, так і у величезних мультинаціональних проєктах.

Рис. 1.3.6 – Логотип фреймворку Django.

Django відомий як фреймворк з «батарейками в комплекті», що означає наявність великої кількості вбудованих інструментів, які можна використовувати за бажанням. Замість написання власного коду, розробники можуть просто імпортувати необхідні пакети. Це частина підходу "конвенції над конфігурацією", що дозволяє використовувати рішення, розроблені професіоналами світового рівня. Django надає широкий спектр вбудованих модулів, таких як автентифікація

через пакет auth, адміністрування за допомогою admin, керування сеансами через Sessions, обробка повідомлень через Messages, створення XML-карти сайту Google за допомогою Sitemaps, підтримка специфічних можливостей Postgres за допомогою пакету Postgres та підключення до "типів" контенту через фреймворк типів контенту.

Використання Python як основної мови програмування дає Django додаткові переваги, оскільки Python є однією з найпростіших мов для початківців і дуже популярною в освітніх установах. Згідно з опитуванням розробників Stackoverflow 2017 року, Python вже обігнав PHP за поширеністю, а робота на Python оплачується краще, ніж на C# та C++.

Один з ключових аспектів Django – це його спільнота. Вона активно допомагає розробникам і працює над тим, щоб зробити фреймворк більш зручним для новачків та стабільним, постійно додаючи нові функції. Документація Django є вичерпною та корисною, що робить її цінним ресурсом для навчання і розуміння можливостей фреймворку.

Масштабованість – ще одна важлива характеристика Django. Фреймворк дозволяє реалізовувати різні підходи до масштабування, такі як запуск окремих серверів для бази даних, медіафайлів та самого застосунку, а також використання кластеризації та балансування навантаження для розподілу застосунку між кількома серверами.

Команда Django також подбала про зручність користувачів, включивши вбудований адміністративний інтерфейс. Це дозволяє розробникам уникати створення власного бекенд-інтерфейсу для базових CRUD-операцій. Адміністративний інтерфейс Django є професійним та універсальним, забезпечуючи зручне управління даними прямо з коробки.

Хоча Django є потужним фреймворком, у нього є кілька недоліків, які можуть виявитися проблемними залежно від потреб. Наприклад, визначення URL за допомогою регулярних виразів може бути складним завданням, особливо для новачків. Крім того, для невеликих проектів Django може здаватися занадто громіздким, а в контексті великих проектів деякі вважають його перевантаженим,

оскільки всі моделі зазвичай зберігаються в одному файлі. Помилки в шаблонах за замовчуванням не відображаються явно, тому можна витратити багато часу на пошуки проблеми, навіть не знаючи, що вона існує. Також фреймворк дотримується суворої парадигми, яка може створювати відчуття монолітності, змушуючи вас дотримуватися одного рекомендованого підходу до реалізації.

Фреймворк Django ідеально підходить для реалізації цього проєкту з кількох причин. По-перше, він простий у використанні, особливо якщо маєш певний досвід з ним. Це дозволяє легко розширювати застосунки, додаючи новий функціонал. Крім того, Python - це дуже приваблива мова програмування з потужними можливостями та читабельним синтаксисом, що полегшує розробку.

Окрім цього, Django має зручний ORM для роботи з базою даних, що спрощує взаємодію та дозволяє створювати складні SQL-запити з використанням Python. Також, Django REST Framework значно спрощує взаємодію з клієнтською частиною через HTTP запити. Не менш важливою перевагою є вбудована безпека, яку надає Django, що є великим плюсом.

1.4 Вибір методології розробки

Розробка програмного забезпечення – це складний процес, який потребує чіткого плану та структури. Методології розробки – це, по суті, набори правил і принципів, що допомагають впорядкувати цей процес. Вони поєднують у собі філософські підходи до створення програмного забезпечення з практичними інструментами, які виникли ще в ранні дні розвитку комп’ютерної техніки.

Головна мета методологій – надати систематичний підхід до розробки програмного забезпечення. Це дозволяє зробити процес більш ефективним та передбачуваним, особливо для команд, які працюють над великими проєктами. Методології також надають чіткий план роботи, що допомагає командам краще

комунікувати між собою, розподіляти завдання та ефективніше обмінюватися інформацією.

Однак, хоча багато IT-компаній визнають важливість використання методологій, не існує єдиного "найкращого" методу. Кожна методологія має свої переваги та недоліки, і вибір оптимальної залежить від специфіки команди, її потреб та цілей. Немає нічого поганого в тому, щоб використовувати різні методи для різних проектів, залежно від їхньої складності та специфіки.

Чому важливо дотримуватися методології розробки програмного забезпечення? Насамперед, важливо обрати методологію розробки ПЗ та послідовно її застосовувати протягом усього проєкту. Ігнорування методології може призвести до серйозних ризиків.

Без чітких вказівок команда розробників може зіткнутися з такими проблемами, як постійно змінювані запити клієнтів, непорозуміння, невиправдані очікування, низька продуктивність, проблеми з бюджетом та погане управління проєктом. Це може призвести до частих змін у програмному забезпеченні без урахування загальних наслідків для проєкту.

Якщо команда не дотримується методології, можуть бути ризики трати часу, грошей і зусиль, а також ризик створення неякісного продукту, який не приносить користі.

Методології розробки ПЗ створені для того, щоб приносити користь як команді розробників, так і клієнтам. Правильний вибір методології забезпечує ведення дискусій у відповідних каналах і прийняття рішень після оцінки всіх факторів.

Застосування методології розробки ПЗ дозволяє команді зменшити неефективність, забезпечити точніші терміни виконання, бути більш організованою та структурованою при роботі зі спонтанними змінами.

Однією з найкращих методологій є методологія Scrum. Scrum – це гнучка система управління проектами, яка допомагає командам структурувати та керувати своєю роботою за допомогою набору цінностей, принципів та практик [8].

Це не просто методологія, а філософія розробки, яка віддає перевагу гнучкості та співпраці над жорстким плануванням. Його головні принципи – командна робота, прозорість та адаптивність. На відміну від традиційних підходів, які вимагають детального попереднього планування, Scrum заснований на ітеративному процесі та постійному вдосконаленні.

Робота в Scrum організована в коротких циклах – спринтах, тривалістю від одного до чотирьох тижнів. Кожен спринт – це фокус на виконанні невеликої, але відчутної частини проекту, що називається інкрементом. Регулярні зустрічі, такі як щоденні планерки та огляди спринтів, забезпечують злагоджену роботу команди та моніторинг прогресу. Розбиваючи роботу на керовані частини, Scrum дозволяє командам швидко реагувати на зміни, адаптуватися до мінливих вимог та створювати якісний продукт.

Основна ціль Scrum – задоволення потреб клієнтів через прозору комунікацію, спільну відповідальність і прагнення до вдосконалення. Починаючи з загальної концепції проекту, Scrum перетворює її в список пріоритетних функцій, що відповідають цілям власника продукту. Така гнучкість стала ключовим елементом в розробці програмного забезпечення, сприяючи інноваціям та задоволенню клієнтів.

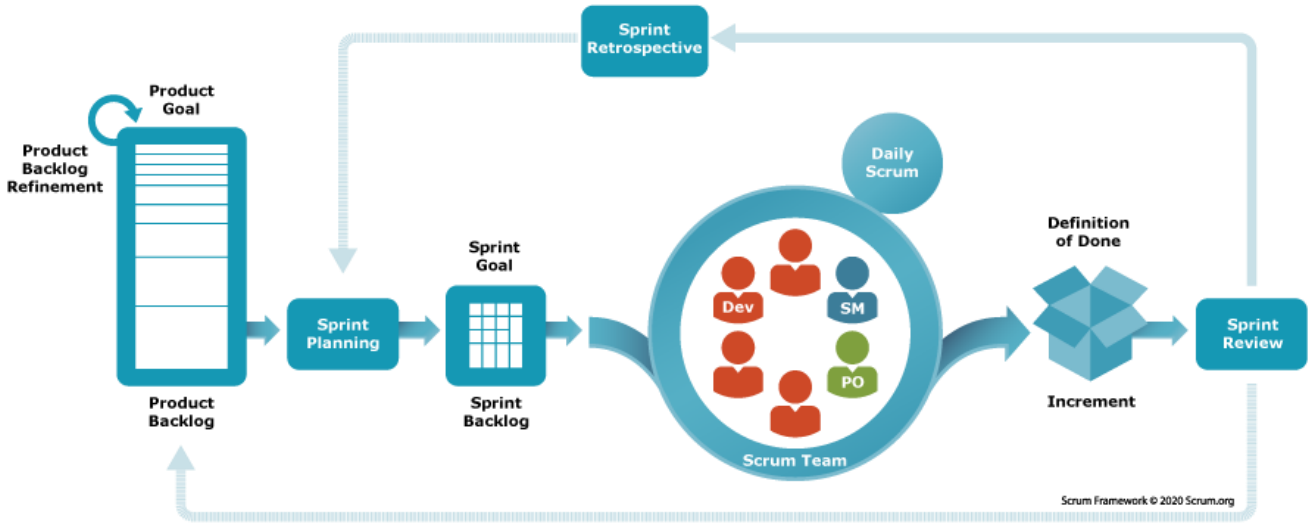

Рис. 1.4.1 – Візуалізація методології Scrum [8].

Scrum має численні переваги перед іншими гнучкими методологіями розробки і є найпоширенішою та найнадійнішою методологією в індустрії програмного забезпечення. Ось деякі з відомих переваг Scrum:

- Легкість масштабування: Ітеративні процеси Scrum, що виконуються протягом фіксованих робочих періодів, дозволяють команді зосередитися на конкретних функціях для кожного періоду. Це не тільки покращує відповідність потребам користувача, але й дає можливість масштабувати функціональність, дизайн та обсяг у впорядкований і прозорий спосіб.
- Відповідність очікуванням: Клієнт визначає цінність кожної вимоги або історії проекту, команда оцінює їх, а Власник продукту пріоритезує. Під час спринт-демонстрацій Власник продукту перевіряє виконання вимог і надає зворотній зв'язок команді.
- Гнучкість до змін: Scrum швидко реагує на зміни у вимогах, спричинені потребами клієнтів або розвитком ринку. Методологія дозволяє легко адаптуватися до нових вимог, навіть для складних проектів.
- Скорочення часу виходу на ринок: Клієнт може почати використовувати найважливіші функції проекту ще до його повного завершення.
- Вища якість програмного забезпечення: Ітеративний підхід і необхідність отримання функціональної версії після кожної ітерації сприяють підвищенню якості програмного забезпечення.
- Своєчасне прогнозування: Знаючи середню швидкість команди за спринт (story points), можна оцінити, коли буде доступний певний функціонал, який все ще знаходиться в беклозі.
- Зниження ризиків: Виконання найцінніших функцій у першу чергу та знання швидкості команди дозволяють ефективно мінімізувати ризики заздалегідь.

Отже, Scrum – це потужна гнучка методологія розробки програмного забезпечення, яка наголошує на співпраці, ітеративному прогресі та безперервній адаптації. Орієнтація на цінність для клієнта, чітка комунікація та гнучке реагування на зміни роблять його високоефективною основою для вирішення складних проєктів у сучасному динамічному середовищі. Використовуючи Scrum, можна швидше створювати високоякісні програмні продукти, ефективно управляти ризиками та послідовно задовольняти мінливі потреби клієнтів.

1.5 Формування вимог до системи

У минулому розділі ми дослідили потенційних конкурентів нашої платформи та визначили двох основних: Microsoft To-do та Trello. Після цього ми провели детальний аналіз цих платформ та виділили їхні сильні та слабкі сторони. На основі отриманих даних, а також розуміння сучасних потреб та тенденцій, в цьому розділі будуть сформовані вимоги до створюваної платформи.

Функціональні вимоги.

Цей сайт має містити наступний функціонал:

- Реєстрація;
- Авторизація;
- Вихід з обліково запису;
- Створення завдання;
- Редагування завдання;
- Видалення завдання;
- Створення списків;
- Редагування списків;
- Видалення списків;
- Створення проєктів;

- Редагування проєктів;
- Видалення проєктів;
- Додавання завдання до проєкту;
- Додавання списку до проєкту;
- Створення тематики;
- Редагування тематики;
- Видалення тематики;
- Фільтрування завдань;
- Відстежування;
- Аналітика;
- Адаптивність;
- Кросплатформеність;

Крім того, сайт має відповідати певному переліку нефункціональних вимог. Наведемо основні з них:

Нефункціональні вимоги.

- Стабільність та надійність: платформа має працювати безперебійно та стабільно, бути доступною 99% часу.
- Безпека: платформа має бути безпечною, надійно зберігати дані користувачів, мати механізми захисту від різноманітних атак.
- Зручність: платформа має мати інтуїтивний дизайн, зручні функції та зрозумілу навігацію.
- Продуктивність: сторінки мають швидко завантажуватися, не більше 3 секунд, а відгук на дії користувача становити не більше 200 мілісекунд.
- Масштабованість: платформа має підтримувати одночасну роботу багатьох користувачів без втрати продуктивності.

1.6 Проєктування відношень між акторами та прецедентами

На момент створення система передбачає у собі 2 актори:

- Користувач:
    - Може реєструватися на сайті, входити на сайт та виходити з сайту;
    - Створювати нові завдання, редагувати та видаляти існуючі;
    - Додавати, редагувати та видаляти власні тематики;
    - Створювати проєкти, редагувати та видаляти існуючі проєкти;
    - Створювати списки, редагувати та видаляти існуючі списки;
    - Переглядати аналітику своєї роботи;
    - Отримувати сповіщення про завдання.

- Адміністратор:
    - Доступні усі ті ж сам функції, що і користувачу;
    - Вхід в адміністративну панель сайту;
    - Додавання, редагування та видалення завдань від усіх користувачів;
    - Додавання, редагування та видалення списків від усіх користувачів;
    - Додавання, редагування та видалення проєктів від усіх користувачів;
    - Додавання, редагування та видалення тематик від усіх користувачів;
    - Створення, редагування та видалення користувачів;

Отже, було наведено зв’язком між двома акторами, користувачем та адміністратором, з їхніми прецедентами. Тепер необхідно структурувати ці дані у вигляді таблиці варіантів використання системи.

Таблиця 1.5.1 – Варіанти використання системи

Код	Основний актор	Найменування	Опис
К1	Користувач	Вхід на сайт	Користувач може увійти на сайт у раніше зареєстрований обліковий запис, для доступу до власних списків, завдань та проєктів.
К2	Користувач	Реєстрація на сайті	Користувач може зареєструватися на сайті.
К3	Користувач	Вихід з сайту	Користувач може вийти зі свого облікового запису на сайті.
К4	Користувач	Створення завдання	Користувач може створювати нові завдання.
К5	Користувач	Редагування завдання	Користувач може редагувати існуючі завдання.
К6	Користувач	Видалення завдання	Користувач може видаляти існуючі завдання.
К7	Користувач	Додавання тематики	Користувач може додавати нові тематики.
К8	Користувач	Редагування тематики	Користувач може редагувати існуючі тематики.
К9	Користувач	Видалення тематики	Користувач може видаляти існуючі тематики.
К10	Користувач	Створення проєкту	Користувач може створювати нові проєкти.
К11	Користувач	Редагування проєкту	Користувач може редагувати існуючі проєкти.
К12	Користувач	Видалення проєкту	Користувач може видаляти існуючі проєкти.
К13	Користувач	Створення списку	Користувач може створювати персоналізовані списки.
К14	Користувач	Редагування списку	Користувач може редагувати існуючі списки.
К15	Користувач	Видалення списку	Користувач може видаляти існуючі списки.
К16	Користувач	Перегляд аналітики	Користувач може переглядати аналітику своєї роботи.
К17	Користувач	Отримання сповіщень	Користувач може отримувати сповіщення про завдання.
А1	Адміністратор	Вхід в адмінпанель	Адміністратор може увійти в адміністративну панель сайту.

Продовження таблиці 1.5.1

Код	Основний актор	Найменування	Опис
A2	Адміністратор	Керування завданнями	Адміністратор може додавати, редагувати та видаляти завдання від усіх користувачів.
A3	Адміністратор	Керування списками	Адміністратор може додавати, редагувати та видаляти списки від усіх користувачів.
A4	Адміністратор	Керування проєктами	Адміністратор може додавати, редагувати та видаляти проєкти від усіх користувачів.
A5	Адміністратор	Керування тематиками	Адміністратор може додавати, редагувати та видаляти тематики від усіх користувачів.
A6	Адміністратор	Керування користувачами	Адміністратор може створювати, редагувати та видаляти користувачів.

Для кращої візуалізації зв’язків між акторами і прецедентами на основі даних з Таблиці 1.5.1 – Варіанти використання системи варто також створити діаграму варіантів використання. Діаграма варіантів використання UML є основною формою визначення вимог до нової системи або програмного забезпечення. Вона описує очікувану поведінку системи, але не вказує конкретні методи реалізації. Варіанти використання можуть бути представлені, наприклад, за допомогою діаграм варіантів використання. Основна ідея такого моделювання полягає в тому, щоб спроєктувати систему з точки зору кінцевого користувача. Це ефективний спосіб передати поведінку системи мовою користувача, описуючи всю її зовнішню функціональність.

На рисунку 1.6.1 представлено скорочену версію діаграми варіантів використання. Повну діаграму можна переглянути в Додатку Б, рисунок Б.1.

Рисунок 1.6.1 – Базова діаграма варіантів використання.

Отже, в цьому розділі було проведено комплексний за змістовний огляд предметно область тайм-менеджменту та необхідних рішень. Було проведено огляд потенційних конкурентів, які будуть у системи та проведено їхній детальний аналіз, де було визначено їхні переваги та недоліки. Також було обрано спосіб реалізації даної системи, а саме за допомогою веб-застосунку, адже це найоптимальніше рішення. Крім того, обґрунтовано вибір технологічних рішень, таких як фреймворки для клієнтської та серверної частини. Беручи до уваги переваги та недоліки конкурентів, а також сучасні потреби в користувачів у тайм-менеджменті, було сформовано функціональні та нефункціональні вимоги до системи. Також було виявлено акторів системи: користувач та адміністратор. Було спроєктовано відношення між акторами та прецедентами, на основі яких створено діаграму варіантів використання.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА КОНСТРУЮВАННЯ СИСТЕМИ

2.1 Проєктування архітектури

У розробці програмного забезпечення вибір правильної архітектури є ключовим для створення надійних, масштабованих та підтримуваних застосунків. Ранні архітектурні рішення значно впливають на довгостроковий успіх проєкту, зокрема на його продуктивність, гнучкість і можливість розширення.

Власне, що ж таке архітектура? Архітектура розробки програмного забезпечення — це розроблена структура додатка, яка включає визначення взаємодії компонентів інтерфейсу з внутрішніми процесами програми. Простіше кажучи, це своєрідний підхід, який визначає які функції за що відповідають та як вони взаємодіють між собою [9].

Добре спроектована архітектура програмного забезпечення визначає його продуктивність, якість, масштабованість, підтримуваність та зручність використання. Основна мета полягає в тому, щоб програмне забезпечення було гнучким, розширюваним і здатним адаптуватися до нових вимог з часом. Для ефективної розробки важливо ретельно проробити архітектуру програмного забезпечення. Це зменшить ризик виникнення помилок, дефектів та інших проблем протягом всього життєвого циклу розробки.

2.1.1 Визначення архітектури системи

Переважна кількість сучасних веб-застосунків використовуються клієнт-серверну архітектуру. У клієнт-серверній архітектурі клієнт - це пристрій або програма, яка запитує послуги або ресурси у сервера. Сервер, з свого боку, є потужним комп'ютером або програмним додатком, який виконує ці запити,

надаючи послуги, дані або ресурси клієнтам. Ця модель дозволяє ефективно розподіляти завдання і ресурси, а також централізовано керувати ними.

У структурі клієнт-сервер, існують три основні складові, які грають ключову роль у забезпеченні взаємодії між користувачами та сервером. Перш за все, є клієнт, який може бути яким-небудь пристроєм або програмою, що виконує запити до сервера. Наприклад, це може бути веб-браузер, який користувач використовує для отримання потрібної інформації або послуг.

На іншому боці знаходиться сервер – централізована система, де зберігаються необхідні ресурси, дані або послуги, які потрібні клієнтам. Сервер обробляє отримані від клієнтів запити, виконує необхідні обчислення, взаємодіє з базами даних і повертає відповіді назад клієнтам.

Мережа виступає як сполучний ланцюг, який забезпечує комунікацію між клієнтами та серверами. Це може бути локальна мережа (LAN), глобальна мережа (WAN) або навіть Інтернет, що дозволяє здійснювати взаємодію з великою кількістю серверів та користувачів по всьому світу.

Переваги клієнт-серверної архітектури відображаються в кількох аспектах. По-перше, ця архітектура дозволяє легко розширювати систему, додаючи нові сервери для відповіді на зростаючий обсяг запитів від клієнтів.

Крім того, централізоване управління ресурсами та даними на сервері спрощує процес обслуговування та проведення оновлень. Такий підхід сприяє підвищенню рівня безпеки, оскільки дозволяє легко впроваджувати заходи захисту даних і ресурсів.

Надалі, можливість одночасного доступу до сервера декількох клієнтів сприяє ефективному використанню ресурсів, що забезпечує швидке обслуговування запитів і задоволення потреб користувачів.

Для створення веб-застосунку для тайм-менеджменту завдань та проєктів буде доцільно обрати трьохрівневу клієнт-серверну архітектуру. Як можна зрозуміти ця архітектура складається з трьох рівнів, кожен з яких відповідає за певні аспекти роботи програми. Клієнтський рівень має справу з інтерфейсом користувача. Він відповідає за те, щоб дані представлялися користувачеві зручно і

прийнятно, а також за отримання від користувача необхідної інформації. Сюди входять усі елементи, які можна побачити на екрані, такі як веб-сторінки та форми.

Серверний рівень - це місце, де відбувається основна обробка даних. Тут знаходиться бізнес-логіка програми, що відповідає за обробку запитів користувачів, маніпулювання даними та виконання складних операцій. Модулі, що працюють на цьому рівні, включають контролери, сервіси та інтерфейси програмування застосунків (API).

Останній рівень це рівень даних – місце, де зберігаються у системі. Це може бути база даних або файлова система, яка забезпечує місце для зберігання інформації.

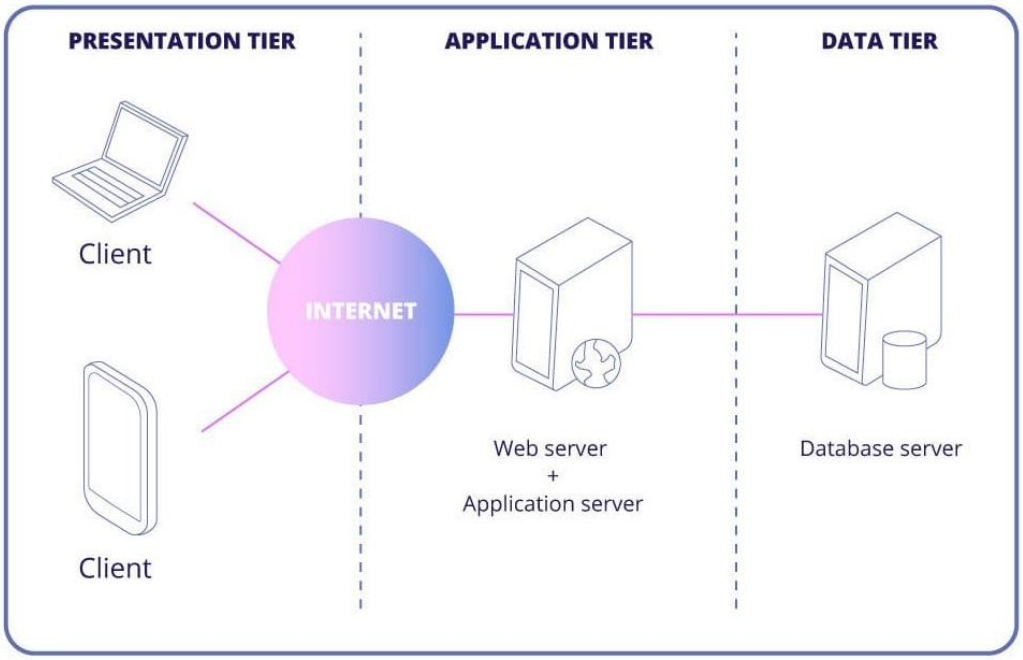

Рис. 2.1.1.1 – Візуалізація трьохрівневої архітектури веб-застосунку [10].

Крім загальної архітектури застосунку існують також різні варіації відображення даних та взаємодія з ними на клієнтські частині. Традиційні веб-сайти та застосунки зберігаються усі свої дані на сервері, включно з HTML-сторінками, які надсилаються на запит користувача. Цей підхід перевірений та надійно працює. Однак в нього є свої недоліки. Перш за все при кожному переході на іншу сторінку з клієнту повинен слатися запит на сервер на отримання нової

сторінки, навіть якщо, взяти до прикладу, що це пагінація в каталозі, де змінюється лише список товарів, а все інше залишається незмінним. Проте при цьому підході буде надсилатися усе, що призводить до збільшення часу на відображення даних і досвід користувачів сайту чи застосунку може знизитися, особливо якщо в них повільний інтернет.

Проте з розвитком технологій та появою JavaScript-фреймворків, з'явився новий тип веб-застосунків, який називають Single Page Application (SPA), що буквально перекладається як односторінковий застосунок. Single page application – це тип веб-застосунків, що працюють без перезавантаження сторінки у браузері, навіть під час відображення нових даних. Цей тип веб-застосунків широко застосовується в реальному житті: такі популярні сервіси, як Facebook, Gmail, Google Maps, GitHub і Twitter, є прикладами односторінкових застосунків.

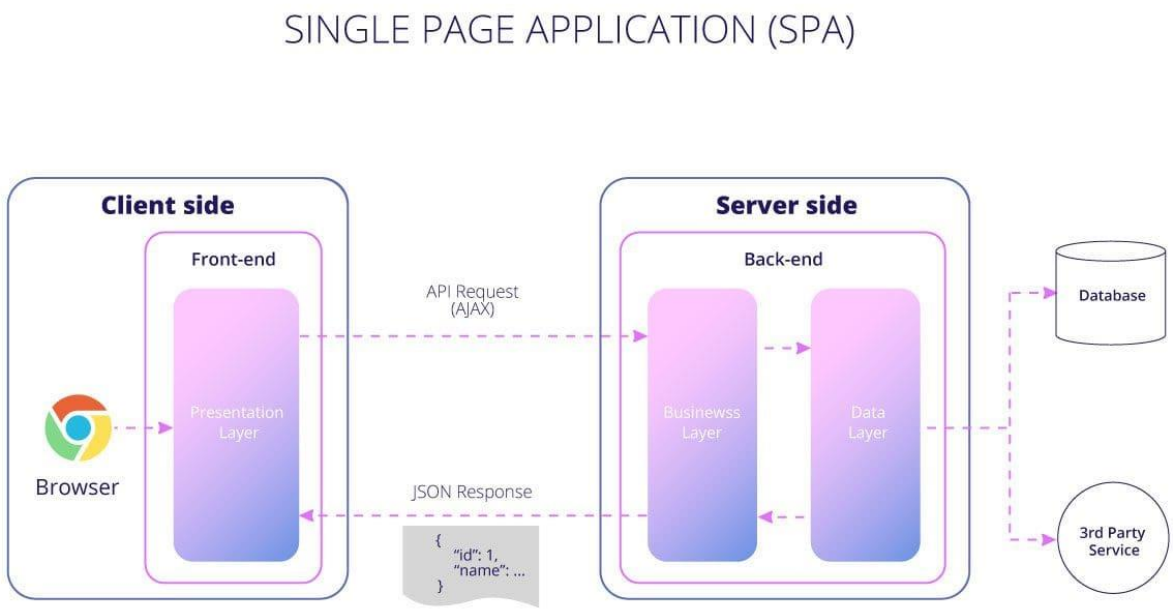

Рис. 2.1.1.2 – Візуалізація Single page application [10].

На відміну від традиційних веб-застосунків та веб-сайтів, як використовують Server side rendering та Static site generation, Single page application дозволяють створювати інтерактивні веб-застосунки, які взаємодіють з сервером через API.

Односторінкові застосунки (SPA) покращують користувацький досвід, забезпечуючи безперервну та плавну взаємодію. Користувачам більше не доводиться зіштовхуватися з перезавантаженням сторінок, як це часто трапляється на традиційних веб-сайтах, що підвищує їхню залученість та задоволеність.

Крім того, SPA зменшують навантаження на сервер, оскільки більша частина обробки переноситься на клієнтську сторону. Це може призвести до зниження витрат, оскільки для обробки трафіку застосунку можна використовувати менш потужні сервери.

Завдяки своїй крос-платформній природі, SPA забезпечують широку сумісність з різними платформами. Використовуючи фреймворки, наприклад, Vue.js, можна створити єдину кодову базу, яка працюватиме у веб-браузерах, на мобільних пристроях та навіть у десктопних застосунках.

Реалізація методів кешування та використання сервісних працівників у SPA дозволяє забезпечити офлайн-функціональність, що дозволяє користувачам користуватися застосунком навіть без стабільного Інтернет-з'єднання. Це особливо корисно для мобільних користувачів.

2.1.2 Проєктування головних класів системи

ООП – це методологія програмування, заснована на представленні програми у вигляді сукупності об’єктів, кожен з яких є екземпляром певного класу, а класи утворюють ієрархію наслідування [11]. Поділ програми на класи, виділення певних сутностей значно спрощує розробку, адже тоді програма набуває зрозумілої структури і розробнику легше в ній орієнтуватися, адже при такому підході легко осягнути кожну сутність, її ролі та властивості. Саме тому й було обрано цю парадигму при розробці серверної частини.

Основні сутності в даній платформі будуть представлені у вигляді класів. Воно будуть об’єднувати в собі певні властивості та функціонал, а також являти

собою представлення даних, які зберігатимуться в базі даних. Перейдімо ж до проєктування основних з них.

Одною з найважливіших сутностей буде реалізація класу Task, який буде являти собою завдання користувачів, тобто основний компонент системи. В даному класі будуть наступні поля:

- Дата кінцевого виконання (deadline) буде являти собою поле типу DateTimeField. Це поле буде зберігати дату та час, до якого має бути виконане завдання.
- Опис (description) являє собою поле типу TextField, в якому буде зберігатися текстовий опис завдання.
- Посилання (links) являє собою поле типу QuerySet, в ньому буде зберігатися запит на пов’язані з цим завданням посилання.
- Назва (name) являє собою поле типу CharField, в ньому буде зберігатися текстова назва завдання.
- Primary key (pk) являє собі поле типу Property, повертає первинний ключ на об’єкт завдання.
- Проєкт (project) являє собою поле типу ForeignKey, яке використовується для зв’язку з моделлю проєкту багато-до-одного.
- Дата початку завдання (start_date) являє собою поле типу DateTimeField, в якому зберігається час та дата початку виконання завдання.
- Статус (status) являє собою поле типу BooleanField. Позначатиме, чи завершене завдання.
- Теми (topics) являє собою поле типу багато-до-багатьох для зв’язку з об’єктами моделі «Тема».

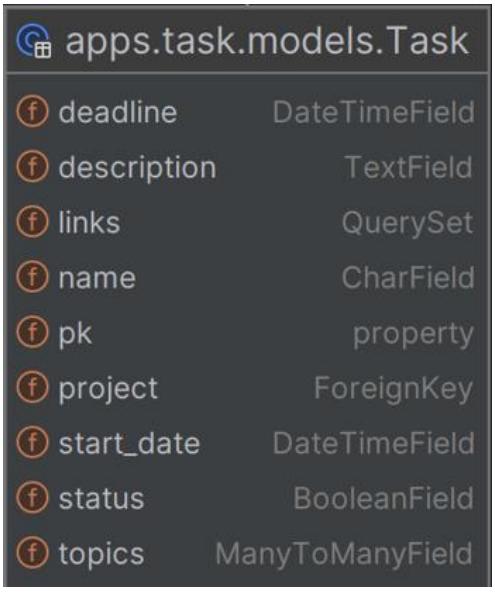

Рис. 2.1.2.1 – Клас Task.

Іншим важливим класом буде клас Project, який буде являти собою проєкт користувача. Даний клас буде налічувати наступні поля:

- Дата кінцевого виконання (deadline) буде являти собою поле типу DateTimeField. Це поле буде зберігати дату та час, до якого має бути виконане проєкту.
- Опис (description) являє собою поле типу TextField, в якому буде зберігатися текстовий опис проєкту.
- Назва (name) являє собою поле типу CharField, в ньому буде зберігатися текстова назва проєкту.
- Primary key (pk) являє собі поле типу Property, повертає первинний ключ на об’єкт проєкту.
- Дата початку завдання (start_date) являє собою поле типу DateTimeField, в якому зберігається час та дата початку виконання проєкту.
- Статус (status) являє собою поле типу BooleanField. Позначатиме, чи завершене проєкту.
- Завдання (tasks) являє собою поле типу QuerySet, в якому зберігається запит на пов’язані з цим проєктом завдання.

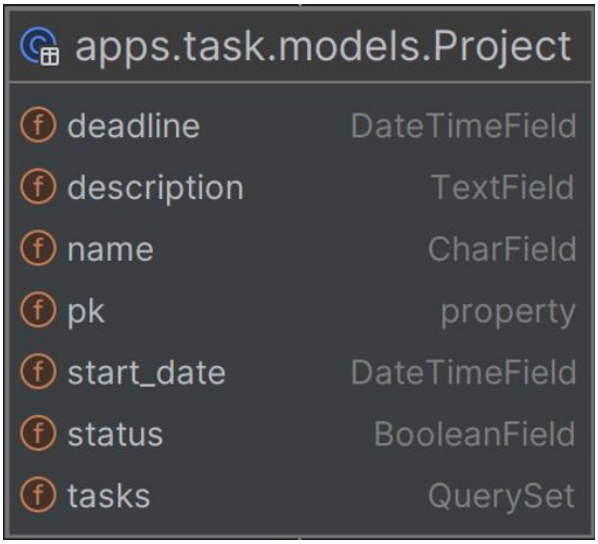

Рис. 2.1.2.2 – Клас Project.

Наступний клас буде представленням прикріплених до завдання посилань. Цей клас буде містити собі наступні поля:

- Назва (name) являє собою поле типу CharField, яке зберігатиме назву посилання.
- Primary key (pk) являє собі поле типу Property, повертає первинний ключ на об’єкт посилання.
- Завдання (task) являє собою поле типу ForeignKey, яке використовується для зв’язку багато-до-одного з об’єктом завдання.
- URL являє собою поле типу URLField, яке буде зберігати в собі саме посилання.

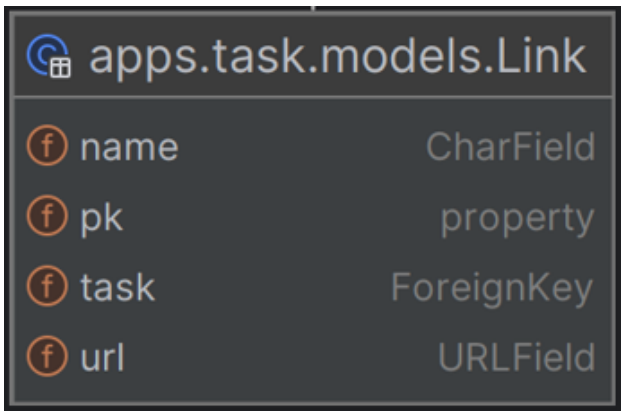

Рис. 2.1.2.3 – Клас Link.

Наступний клас буде представлення тем для завдань. Цей клас буде містити наступні поля:

- Назва (name) являє собою поле типу CharField, буде зберігати в собі назву теми.
- Primary key (pk) являє собі поле типу Property, повертає первинний ключ на об’єкт теми.
- Завдання (tasks) являє собою поле типу QuerySet, в якому зберігається запит на пов’язані з цією темою завдання.

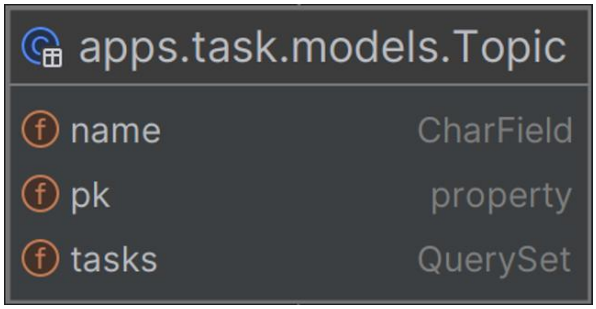

Рис. 2.1.2.4 – Клас Topic.

Якщо ж описати їхні зв’язки, то проєкти можуть складатися з багатьох завдань. Кожне завдання належить лише одному проєкту. Завдання ж, у свою чергу, можу бути пов’язаним з багатьма темами, і кожна тема може бути пов’язаною з багатьма завданнями. Також в завдання може бути багато пов’язаних посилань, однак посилання може відноситися лише до одного завдання. Детальніше на ці зв’язки можна глянути на рисунку (див. рис. 2.1.2.5).

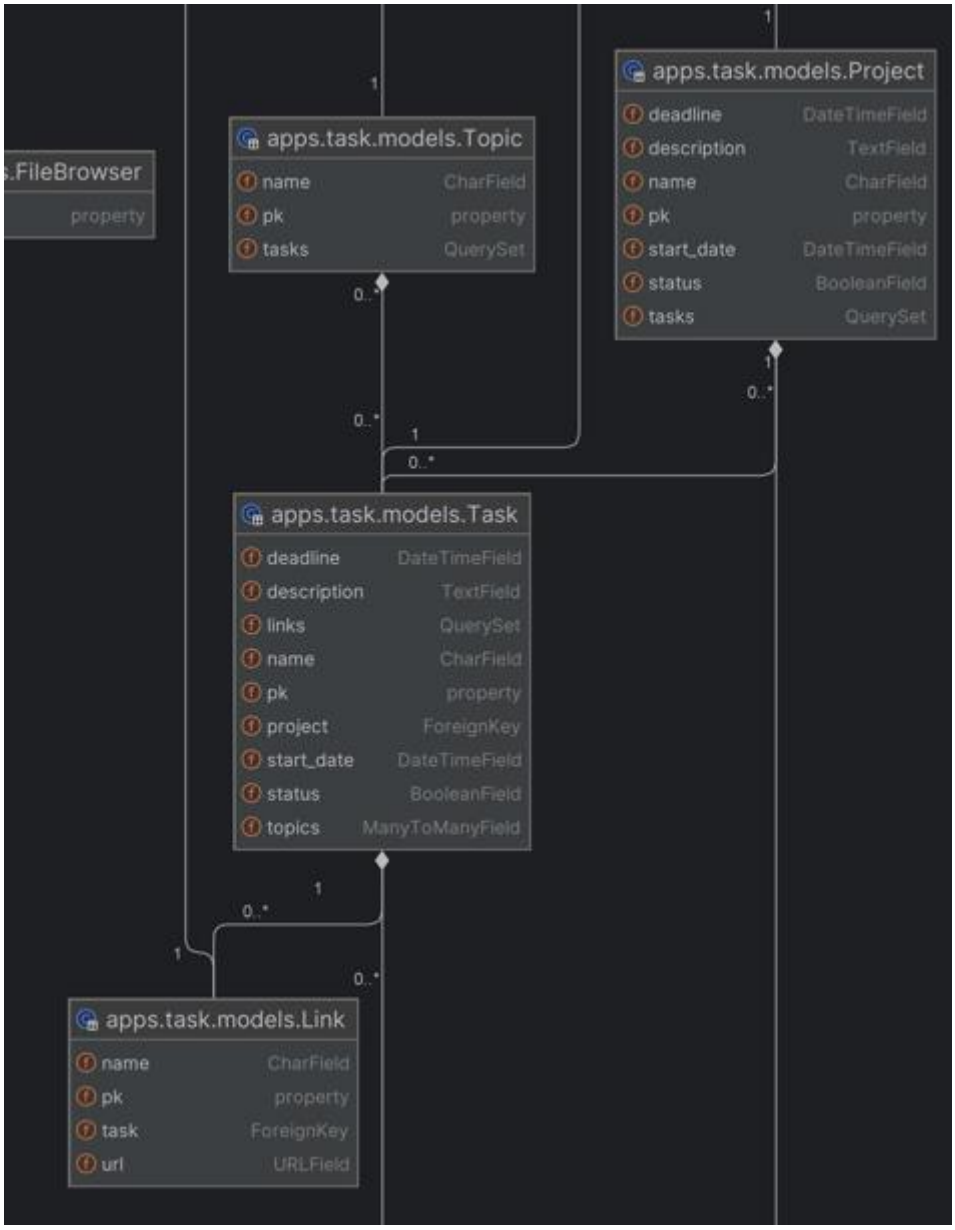

Рисунок 2.1.2.5 – Зв’язки між наведеними класами.

Крім того, кожне завдання, кожна тема і решта описаних сутностей, будуть відноситися до класу користувача. Тому варто його також винести окремо. Клас Profile буде втілювати в собі сутність користувача. У ньому будуть наступні поля:

- date являє собою поле типу DateTimeField, де зберігається дата та час створення профілю.
- date_joined являє собою поле типу DateTimeField, де зберігається дата приєднання до системи.

- email являє собою поле типу EmailField, де зберігається електронна адреса користувача.
- first_name являє собою поле типу CharField, де зберігається ім’я користувача.
- is_active являє собою поле типу BooleanField, яке вказує, чи активний користувач, чи ні.
- is_staff являє собою поле типу BooleanField, яке вказує, чи є користувач адміністратором.
- last_name являє собою поле типу CharField, де зберігається ім’я прізвище.
- links являє собою поле ManyToManyField і використовується для зберігання зв’язку між користувачем та посиланнями виду багато-до-багатьох.
- notification_type являє собою поле типу CharField, де зберігаєть обраний варіант надсилання сповіщень.
- objects поле типу UserManager - менеджер для керування об'єктами профілів.
- picture являє собою поле типу FileBrowseField, використовується для зберігання аватару користувача.
- projects являє собою поле ManyToManyField і використовується для зберігання зв’язку між користувачем та проєктом виду багато-до-багатьох.
- tasks являє собою поле ManyToManyField і використовується для зберігання зв’язку між користувачем та завдання виду багато-до-багатьох.
- topics являє собою поле ManyToManyField і використовується для зберігання зв’язку між користувачем та завдання виду багато-до-багатьох.

- type_profile_fk являє собою поле типу ForeignKey і використовується для зв’язку з іншою моделлю, яка визначає тип профілю.
- username являє собою поле типу CharField і використовується для зберігання ім’я користувача в системі.
- username_validator – це UnicodeUsernameValidator, який використовується для перевірки правильності введеного імені користувача.

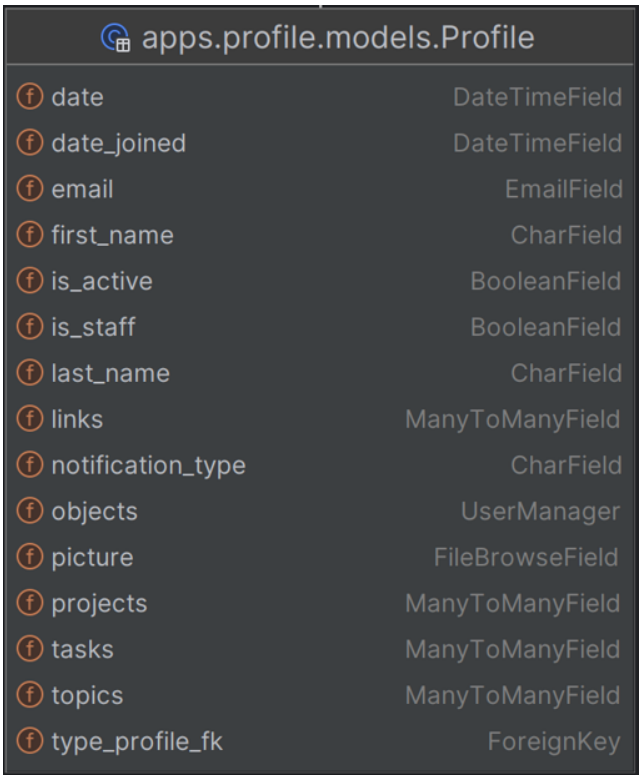

Рис. 2.1.2.6 – Клас Profile.

Було створено діаграму класів, де наведено класи системи та зображено зв’язки між ними. Переглянути повну діаграму класів можна в додатку Б.

2.1.3 Проєктування структури веб-застосунку

Зручність користування сайтом має завжди бути пріоритетним завданням при його проєктуванні. Грамотне її планування забезпечує легке та приємне користування, логічну навігацію по сторінках та в цілому роботу із сервісом. У цей процес можна включити створення переліку сторінок, які будуть на сайті, планування переходів між ними та відповідні зв’язки, і саме проєктування взаємодії з функціоналом, який надає веб-застосунок.

Почнімо з визначення того, який перелік сторінок буде в цьому веб-застосунку.

- Вітальна сторінка – сторінка буде першою, на яку попадатиме користувач, де знаходитиметься короткий опис, банер, посилання на вхід у систему.
- Сторінка логіну – на сторінці користувачі зможуть авторизуватися за допомогою свого логіну та паролю.
- Сторінка реєстрації – на цій сторінці користувачі зможуть зареєструватися, вказавши логін та проль.
- Сторінка скиду паролю – на цій сторінці користувачі зможуть скинути пароль, у разі, як вони забули його.
- Головна сторінка системи – буде першою сторінкою роботи з системою. На ній будуть відображатися завдання на сьогодні, можливість створити завдання, створити проєкт, та невеличка аналітика.
- Сторінка проєктів – на цій сторінці можна буде створити новий проєкт, переглянути список існуючих проєктів та відповідно перейти на них.
- Сторінка проєкта – на цій сторінці будуть зображені всі задачі та списки проєкту користувача. Можна буде створити нову задачу, новий список та відфільтрувати за статусом, терміновістю і тематикою.

- Сторінка задач – ця сторінка буде ідентичною до сторінки проєкту, проте додається можливість фільтрації по проєктам.
- Сторінка календаря – на цій сторінці буде великий календар, на якому будуть показувати завдання по днях, тижнях, годинах. Також буде можливість додати нове завдання.

Загалом, веб-застосунок буде мати 9 варіантів сторінок. Заважаючи на їхню невелику кількість, навігація по ним не буде проблемою. Крім того, для зручності, на головній сторінці системи буде сайдбар з меню, в якому перелічені головні сторінки: домівка, проєкти, задачі і календар. Також будуть кнопки, які відповідатимуть за налаштування та вихід із системи.

Загальну структуру і взаємодію між сторінками сайту можна побачити на діаграмі на рисунку 2.1.3.1 – Структура сайту.

Рис. 2.1.3.1 – Структура сайту.

Наступним важливим етаном буде відображення станів основних подій у веб-застосунку. Що ж таке діаграма станів? Діаграма станів UML (також відома як діаграма станів) візуалізує стани скінченного автомата, тобто поведінкову модель, що складається з дій та станів або переходів між станами. Діаграма передбачає початковий і кінцевий стан та принаймні один проміжний стан для кожного об'єкта в моделі [12]. Використовуючи діаграми станів, доволі легко розуміти порядок дій в програмі, дають чітке розуміння її поведінки. Крім того, діаграми використання

можна використовувати для демонстрації логіки програми без занурення в технічні аспекти чи код. А також вона спрощує процес тестування системи, адже QA-спеціаліст розуміє, що і на якому етапі має відбуватися за певних умов.

Наступна діаграма показує процес авторизації в системі. Процес починається з початкового стану, позначеного чорною крапкою. Користувач переходить до стану введення логіну та пароля. Він вводить свої облікові дані (логін та пароль). Після введення даних, система переходить до стану "Validating", де відбувається перевірка введених логіну та пароля. Якщо введені дані правильні, система переходить до стану "Success" (Успіх). Якщо введені дані неправильні, система переходить до стану "Error" (Помилка). У разі правильних даних, система переходить до стану "Success". В цьому стані система здійснює запис сесії користувача. Після запису сесії користувач переходить до стану "LoggedIn". Стан "LoggedIn" означає, що користувач успішно увійшов в систему. У разі неправильних даних, система переходить до стану "Error". З цього стану система повертає користувача до стану "Введення логіну та пароля" для повторної спроби.

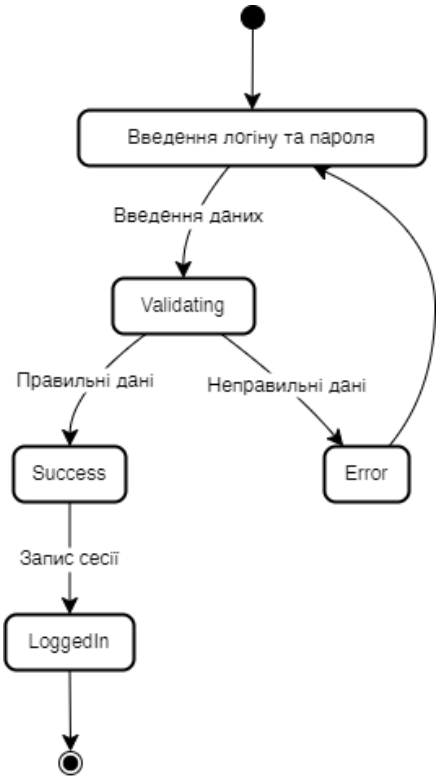

Рис. 2.1.3.2 – Діаграма станів входу в систему.

Діаграма описує життєвий цикл проєкту, починаючи з його створення, продовжуючи роботою над ним і завершуючись архівуванням. Проєкт створюється. В процесі до нього можуть додаватися задачі, списки і відповідно регадуватися. Після виконання всіх завдань проєкт завершується. Якщо проєкт не вдалося завершити вчасно, він переходить в стан "прострочено". З простроченого стану проєкт можна відкласти, тобто тимчасово не виконувати. Після завершення проєкт архівується.

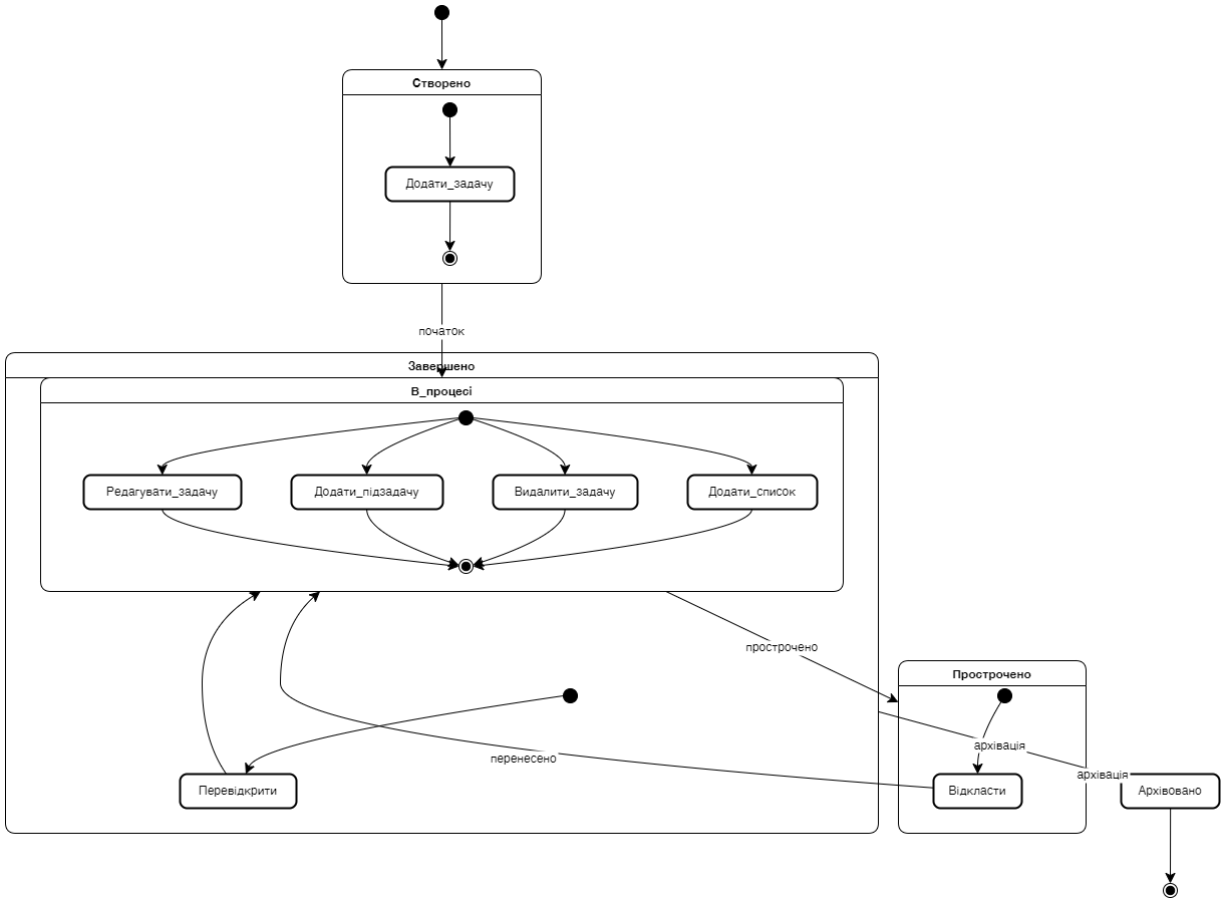

Рис. 2.1.3.3 – Діаграма стану проєкту.

Отже, в цьому пункті було описано структуру веб-застосунку, визначено перелік сторінок клієнтської частини, та спроєктовано зв’язки та переходи між ними. Крім того, було розглянути процеси авторизації та життєвий цикл проєкту і створено відповідні діаграми.

2.3 Конструювання системи

Під час огляду предметної області було сформовано перелік функціональних та нефункціональних вимог на основі чого було визначено архітектуру веб-застосунку та описано основні класи системи та її компоненти. Настав час реалізувати закладену архітектуру та функціонал.

2.3.1 Конструювання серверної частини

Оскільки для побудови серверної частини буде використовуватися фреймворк Django та мова програмування Python, то найкращим середовищем розробки на цих технологіях буде IDE Pycharm, яка широко підтримує програмування на мові Python, а також має спеціальні можливості для Django. Крім того, дане IDE має в собі безліч вбудованого функціоналу, як робота з GIT, безліч плагінів та зручну підсвітку синтаксису. Хоч вона і платна, однак компанія JetBrains надає безкоштовну ліцензію студентам.

Що ж, для початку оберемо версію мови програмування. На перевагу встановленню Python на робочу машину за допомогою інсталятора, існує спеціальний інструмент, який називається pyenv. Pyenv – це простий, потужний, безкоштовний, відкритий і крос-платформний інструмент, який орієнтований на управління кількома версіями Python на робочій машині [13]. Процес інсталяції доволі простий, тому описувати його додатково немає сенсу. Цей інструмент надає велику перевагу при підтримці великої кількості проєктів, які були написані на різних версіях цієї мови, тому дуже зручний у використанні.

Для вибору та встановлення версії мови Python слід відкрити термін та прописати наступну команду:

```
pyenv install 3.12.3
pyenv local 3.12.3
```

Тепер в нашому середовищі розробки встановлена дана версія Python. Наступним кроком створюємо віртуальне середовище (venv) та встановлюємо відпвідні модулі, які заготовлені у файлі r.txt.

```
pyenv -m venv venv
python venv/Scripts/activate
pip install -r r.txt
```

Тепер наше середовище повністю готове до старту проєкту. Щоб створити новий проєкт Django, потрібно в терміналі ввести наступну команду:

```
django-admin startproject unine_engine
```

Дана команда створює головний застосунок Django.

Наступним важливим етапом буде налаштування Django проєкту для бази даних. Ми будемо використовувати провірену часом СУБД MySQL. Для цього у файлі settings.py потрібно вказати наступне:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': os.environ.get('MYSQL_DATABASE', 'database'),
        'USER': os.environ.get('MYSQL_USER', 'root'),
        'PASSWORD': os.environ.get('MYSQL_PASSWORD', 'password'),
        'HOST': os.environ.get('MYSQL_HOST', 'localhost'),
        'OPTIONS': {
            'charset': 'utf8',
        },
    }
}
```

Рисунок 2.3.1.1 – Налаштування бази даних в проєкті Django.

Тепер щоб застосунок запрацював, необхідно провести міграції. Міграції – це спосіб Django поширювати зміни, які ви вносите у моделі (додавання поля, видалення моделі і т.д.) у схемі бази даних [14]. Проводяться вони за допомогою двох наступних команд в терміналі:

```
python manage.py makemigrations
python manege.py migrate
```

Тепер базові налаштування завершені і проєкт готовий до подальшої розробки.

Наступним етапом буде реалізація базових класів, які були спроєктовані в розділі 2.1.2.

Для створення даних класів спершу створимо окрему папку, де будуть знаходитися модулі нашого проєкту, або ж як вони там називаються «застосунки». Для цього в терміналі прописуємо команду:

```
mkdir apps
```

Дана команда створить папку з відповідним ім’ям. В цій папці створимо ще одну папку з назвою «tasks», яка слугуватиме для розробки всієї логіки, пов’язаної із завданнями. За допомогою наступної команди Django створить необхідну застосунку структуру всередині новоствореної папки:

python manage.py startapp tasks appa/tasks

Після цієї команди всередині папки tasks створиться файл migrations.py. Всередині нього ми пропишемо реалізацію класів.

```python
class Task(models.Model):
    name = models.CharField(max_length=255)
    description = models.TextField(blank=True, null=True)
    start_date = models.DateTimeField()
    deadline = models.DateTimeField()
    status = models.BooleanField(default=False)
    project = models.ForeignKey(Project, related_name='tasks', on_delete=models.CASCADE)
    topics = models.ManyToManyField(Topic, related_name='tasks')

    def __str__(self):
        return self.name
```

Рисунок 2.3.1.2 – Реалізація класу Task.

Клас Task представляє модель завдання з полями для назви, опису, дати початку, кінцевого терміну, статусу виконання, посиланням на проект та зв'язком з темами. Назва зберігається у вигляді рядка з обмеженням у 255 символів, опис може бути порожнім, дати представлені у форматі дати та часу, статус

відображається як булеве значення, проект пов'язаний за допомогою зовнішнього ключа, а теми зв'язуються за допомогою багато-багато відношення.

```
class Project(models.Model):
    name = models.CharField(max_length=255)
    description = models.TextField(blank=True, null=True)
    start_date = models.DateTimeField()
    deadline = models.DateTimeField()
    status = models.BooleanField(default=False)

    def __str__(self):
        return self.name
```

Рисунок 2.3.1.3 – Реалізація класу Project.

Клас Project представляє модель проекту з полями для назви, опису, дати початку, кінцевого терміну та статусу виконання. Назва зберігається у вигляді рядка з обмеженням у 255 символів, опис може бути порожнім, дати представлені у форматі дати та часу, а статус відображається як булеве значення. Метод str повертає назву проекту як його рядкове представлення.

```
class Topic(models.Model):
    name = models.CharField(max_length=255)

    def __str__(self):
        return self.name
```

Рисунок 2.3.1.4 – Реалізація класу Topic.

Клас Topic представляє модель теми з полем для назви. Назва зберігається у вигляді рядка з обмеженням у 255 символів. Метод str повертає назву теми як її рядкове представлення.

```python
class Link(models.Model):
    name = models.CharField(max_length=255)
    url = models.URLField()
    task = models.ForeignKey(Task, related_name='links', on_delete=models.CASCADE)

    def __str__(self):
        return self.name
```

Рисунок 2.3.1.5 – Реалізація класу Link.

Клас Link представляє модель посилання з полями для назви та URL-адреси. Назва зберігається у вигляді рядка з обмеженням у 255 символів, URL-адреса зберігається у форматі URL-поля, а посилання пов'язане із завданням за допомогою зовнішнього ключа. Метод str повертає назву посилання як його рядкове представлення.

Для того, аби ці дані відображалися в адмінпанелі, їх необхідно зареєструвати у файлі admin.py. Це робиться наступним чином:

```python
from django.contrib import admin

from .models import Task, Project, Topic, Link

admin.site.register(Task)
admin.site.register(Project)
admin.site.register(Topic)
admin.site.register(Link)
```

Рисунок 2.3.1.6 – Реєстрація створених класів в адмінпанелі.

За аналогією з минулим застосунком, тепер створимо застосунок під профіль користувача. В ньому створимо клас, який буде втілювати в собі користувача. Django має з коробки реалізацію користувача, тому ми наслідуємося від його стандартної реалізації, щоб розширити функціонал цього класу.

```python
class Profile(AbstractUser):
    type_profile_fk = models.ForeignKey(TypeProfile, on_delete=models.SET_NULL, null=True, blank=True)
    picture = FileBrowseField(verbose_name=_('admin__image'), max_length=255, directory="profile/", extensions=FILEBROWSER_EXTENSIONS['Image'], blank=True)
    date = models.DateTimeField(verbose_name=_("admin__date"), auto_now_add=True, auto_now=False)
    tasks = models.ManyToManyField( to: 'task.Task', related_name='profiles', blank=True)
    projects = models.ManyToManyField( to: 'task.Project', related_name='profiles', blank=True)
    notification_type = models.CharField(verbose_name=_('admin__notification_type'), max_length=255, default='email', choices=NOTIFICATION_TYPES)
    topics = models.ManyToManyField( to: 'task.Topic', related_name='profiles', blank=True)
    links = models.ManyToManyField( to: 'task.Link', related_name='profiles', blank=True)

    def __str__(self):
        try:
            return f'{self.first_name} {self.last_name}'
        except Exception:
            return str(self.id)
```

Рисунок 2.3.1.7 – Реалізація класу Profile.

Клас Profile представляє модель профілю користувача з різними полями для зберігання інформації про користувача. Він наслідується від AbstractUser, що надає стандартні поля користувача, такі як ім'я користувача, пароль, електронна пошта тощо. Додатково, у класі є поле type_profile_fk, яке є зовнішнім ключем до моделі TypeProfile, що дозволяє встановлювати тип профілю. У випадку видалення пов'язаного об'єкту TypeProfile, значення цього поля встановлюється в NULL.

Поле picture використовується для зберігання зображення профілю за допомогою FileBrowseField. Воно має обмеження на типи файлів і зберігає шляхи до файлів у директорії profile/. Поле date автоматично заповнюється під час створення об'єкту і містить дату та час створення профілю. Поле tasks визначає зв'язок багатьох-до-багатьох з моделлю Task, дозволяючи профілю мати кілька завдань. Аналогічно, поле projects визначає зв'язок багатьох-до-багатьох з моделлю Project, дозволяючи профілю мати кілька проектів.

Поле notification_type зберігає тип сповіщень, які користувач хоче отримувати (наприклад, електронна пошта). Воно має обмеження на довжину 255 символів і набір можливих значень, визначених у NOTIFICATION_TYPES. Поле topics визначає зв'язок багатьох-до-багатьох з моделлю Topic, дозволяючи профілю мати кілька тем. Поле links визначає зв'язок багатьох-до-багатьох з моделлю Link, дозволяючи профілю мати кілька посилань.

Метод str повертає рядкове представлення профілю, що складається з імені та прізвища користувача, якщо вони доступні. У випадку виникнення винятку повертається ідентифікатор профілю. Таким чином, клас Profile зберігає детальну

інформацію про користувача та дозволяє гнучко керувати взаємозв'язками з іншими моделями завдань, проєктів, тем та посилань.

Для взаємодії з клієнтською частиною ми реалізуємо RESTful API, до якого вона буде звертатися і отримувати чи надсилати певні дані.

Перш за все варто встановити офіційну бібліотеку, який значно спрощує реалізацію RESTful API. Для цього в терміналі необхідно ввести наступну команду:

pip install djangorestframework

Вона встановлює бібліотеку Django REST Framework. Далі ми реєструємо його в нашому проєкті для взаємодії з ним.

Далі ми реалізовуємо класи, які перетворюють наші моделі у формат JSON. У файлі serializer.py створюємо наступні класи:

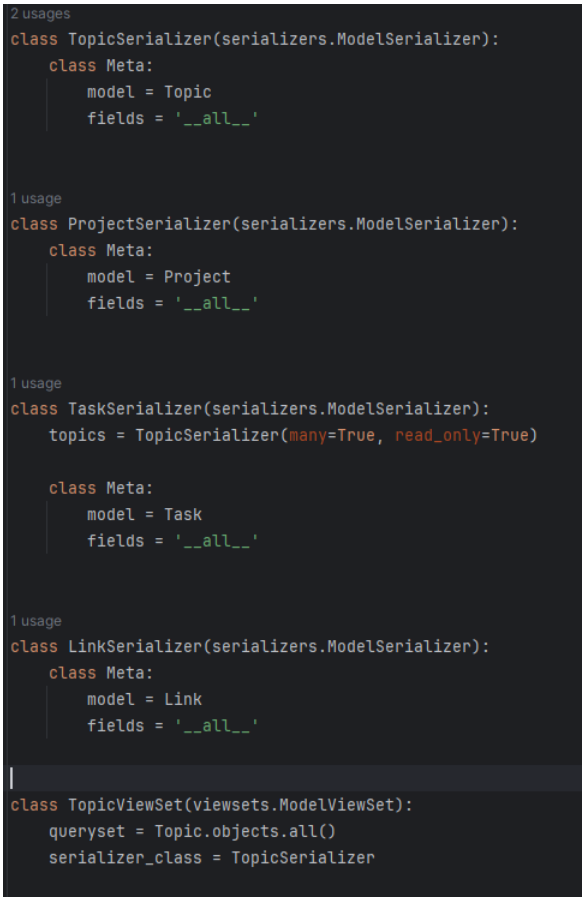

Рисунок 2.3.1.8 – Реалізація класів-серіліазаторів.

Класи TopicSerializer, ProjectSerializer, TaskSerializer і LinkSerializer відповідають за перетворення об'єктів моделей у дані JSON та навпаки.

Далі створимо реалізацію моделі CRUD (створення, читання, оновлення і видалення). Для цього реалізуємо наступні класи, які наслідуються від ModelViewSet.

```
class TopicViewSet(viewsets.ModelViewSet):
    queryset = Topic.objects.all()
    serializer_class = TopicSerializer


class ProjectViewSet(viewsets.ModelViewSet):
    queryset = Project.objects.all()
    serializer_class = ProjectSerializer


class TaskViewSet(viewsets.ModelViewSet):
    queryset = Task.objects.all()
    serializer_class = TaskSerializer


class LinkViewSet(viewsets.ModelViewSet):
    queryset = Link.objects.all()
    serializer_class = LinkSerializer
```

Рисунок 2.3.1.9 – Реалізація класів для забезпечення CRUD моделей.

Наступним кроком буде прописування маршрутів до наших ViewSets, аби можна було надсилати запити на відповідну адресу.

```
router = routers.DefaultRouter()
router.register( prefix: 'topics', TopicViewSet)
router.register( prefix: 'projects', ProjectViewSet)
router.register( prefix: 'tasks', TaskViewSet)
router.register( prefix: 'links', LinkViewSet)

urlpatterns = [
    path('', include(router.urls)),
]
```

Рисунок 2.3.1.10 – Маршрути до API.

Можна вважати, що налаштування API на стороні серверу завершено. Django REST Framework надає зручний інтерфейс для тестування API. Приклад успішного запиту на список задач наведений на рисунку нижче.

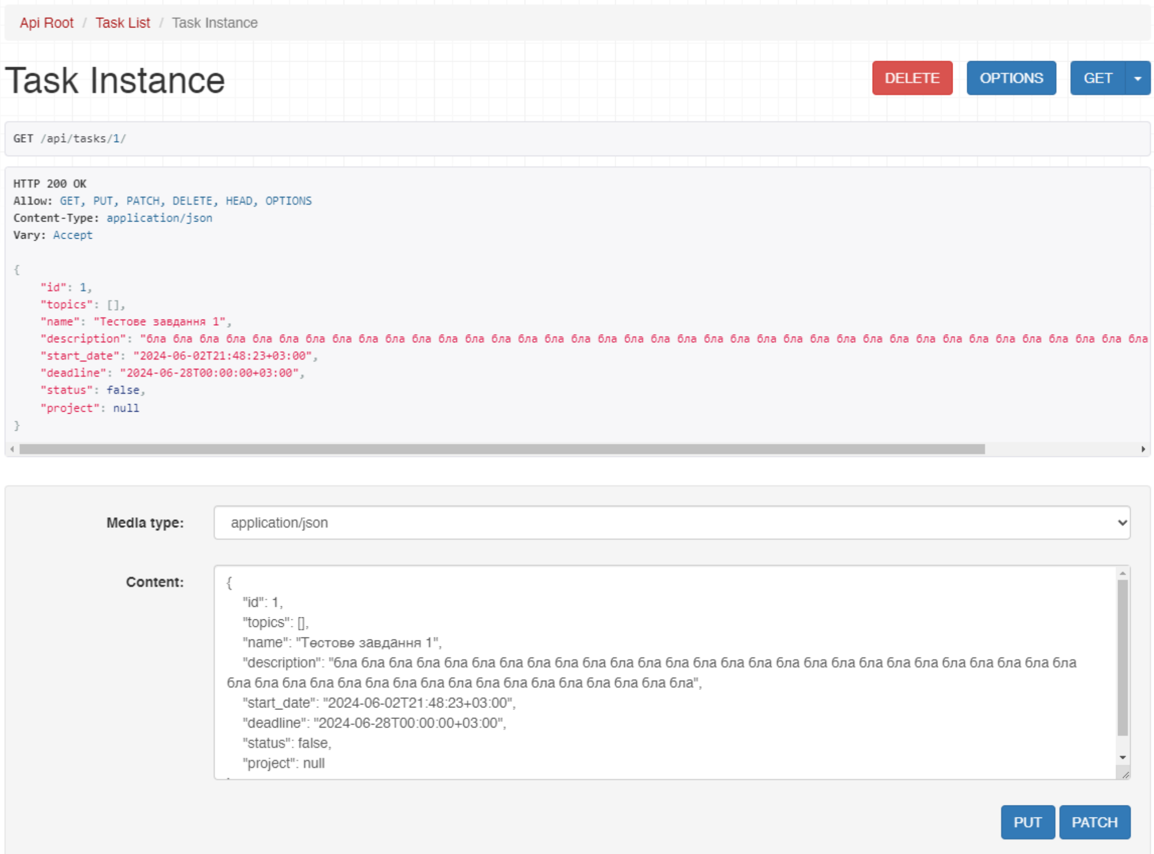

Рисунок 2.3.1.11 – Тестовий запит на API зі списком задач.

Наступним кроком буде реалізація функціоналу надсилання сповіщень про наближення дедлайну. На початку цей функціонал буде реалізований за допомогою електронного листа на пошту, проте закладено можливість для обрання інших варіантів: мессенджер-боти, смс, тощо.

Для початку потрібно встановити модуль Celery. Celery – це інструмент, який виконує завдання в асинхронному режимі в Python-додатках. Іншими словами, програма виступає в ролі посередника, забезпечуючи паралельне виконання операцій, не забираючи при цьому цінні ресурси основного потоку виконання [15]. Встановлення відбувається за допомогою команди в терміналі:

```
pip install celery redis
```

Далі створюємо файл celery.py, в якому будуть задані основні налаштування Celery.

```python
from __future__ import absolute_import, unicode_literals
import os
from celery import Celery

os.environ.setdefault( key: 'DJANGO_SETTINGS_MODULE',  value: 'unine_engine.settings')

app = Celery('unine_engine')

app.config_from_object( obj: 'django.conf:settings', namespace='CELERY')
app.autodiscover_tasks()
```

Рисунок 2.3.1.12 – Конфігурація Celery.

Далі створюємо файл tasks.py, в якому описуємо два завдання для celery – надсилання листів.

```python
@shared_task
def send_task_reminder(task_id):
    task = Task.objects.get(id=task_id)
    send_mail(
        subject: 'Нагадування про дедлайн завдання',
        message: f'Дедлайн Вашого завдання "{task.name}" минає через 12 годин.',
        settings.DEFAULT_FROM_EMAIL,
        recipient_list: [task.project.user.email],
    )

@shared_task
def send_project_reminder(project_id):
    project = Project.objects.get(id=project_id)
    send_mail(
        subject: 'Нагадування про дедлайн проєкту',
        message: f'Дедлайн Вашого проєкту "{project.name}" минає через 5 днів.',
        settings.DEFAULT_FROM_EMAIL,
        recipient_list: [project.user.email],
    )
```

Рисунок 2.3.1.13 – Завдання для Celery.

Далі створимо сигнал для автоматичного створення завдань Celery при створенні або оновленні завдань і проєктів. Для цього у файлі singals.py описуємо наступні сигнали:

```python
@receiver(post_save, sender=Task)
def schedule_task_reminder(sender, instance, created, **kwargs):
    if created:
        reminder_time = instance.deadline - datetime.timedelta(hours=12)
        send_task_reminder.apply_async((instance.id,), eta=reminder_time)

@receiver(post_save, sender=Project)
def schedule_project_reminder(sender, instance, created, **kwargs):
    if created:
        reminder_time = instance.deadline - datetime.timedelta(days=5)
        send_project_reminder.apply_async((instance.id,), eta=reminder_time)
```

Рисунок 2.3.1.14 – Сигнали для створення завдань Celery.

Отже, в цьому пункті було детально описано процес конструювання серверної частини системи тайм-менеджменту та реалізації функціональних вимог до цього проєкту. Було створено відповідні моделі в базі даних, реалізовано API для взаємодії з клієнтською частиною.

2.3.2 Конструювання клієнтської частини

Перейдемо тепер до фінального етапу конструювання. Як уже була раніше вказано, клієнтська частину буде реалізовано на фреймворку Vue. Реалізація серверної частини відбувалася у IDE Pycharm від компнаії Jetbrains. Дана компанія також має IDE для проєктів на мові програмування JavaScript – WebStrorm. У даної IDE такі ж переваги, як були раніше описані в Pycharm. А також одразу «з коробки» підтримується синтаксис Vue та його компоненти.

Робота Vue буде відбуватися у зв’язці з Vite – швидким та сучасним компілятором проєктів. Його перевагою є швидкість, що можна зрозуміти з його назви (vitesse – з французької перекладається як «швидкість»).

Для старту нового проєкту в терміналі прописуємо наступну команду:

npm init vite

```
PS E:\Projects\TaskManager> npm init vite
Need to install the following packages:
Ok to proceed? (y) y
√ Project name: ... TaskManager
√ Package name: ... taskmanager
√ Select a framework: » Vue
√ Select a variant: » JavaScript

Scaffolding project in E:\Projects\TaskManager\TaskManager...
```

Рисунок 2.3.2.1 – Створення нового проєкту Vue + Vite.

Дана команда створила новий проєкт Vue зі всіма необхідними файлами і тепер він готовий до розробки.

Як готову бібліотеку компонентів будемо використовувати Vuetify. Це бібліотека компонентів Vue, таких як кнопки, поля, слайдери тощо, яка пришвидшить процес розробки, адже більшість необхідних нам компонентів у ній вже реалізовано.

Крім того, давно вже з’явилися фреймворки для стилізації. Раніше найпопулярнішим серед таких був Bootstrap. Зараз він і надалі є одним з лідерів у цій сфері, проте при створенні сучасних проєктів, надають перевагу TailwindCSS. Він в дечому схожий з Bootstrap. Принаймні обоє використовують підхід написання, який називається атомічним. Тобто будується на використанні маленький класів css, які надають одну властивість. На приклад, «p-1» в Tailwind дасть внутрішній відступ елементу.

Для навігації між сторінками раніше використовувалися HTTP запити, де передавалася ціла сторінка. При підході з використанням SPA цього більше не потрібно робити, запит буде посилатися лише на необхідні дані. Для зміни сторінок, а у випадку Vue, правильніше буде назвати їх компонентами, буде використовуватися офіційний модуль Vue Router. Саме він буде заміняти компоненти в залежності від шляху, на якому перебуває чи переходить користувач.

Для конфігурації роутера, необхідно створити окремий файл. В ньому слід прописати наступне:

Рисунок 2.3.2.2 – Конфігурація роутера.

Тут ми вказуємо, які маршрути будуть в нашій клієнтській частині застосунку, по яким зможе переходити користувач.

Ще одним важливим елементом початкової конфігурації проєкту Vue буде підключення інструменту роботи зі станами. Pinia надає широкі можливості для збереження станів застосунку Vue. Це не офіційний модуль, яким є vue router або ж аналог Pinia – Vuex. Проте він має над своїм офіційним аналогом деякі переваги, а головною з них є можливість створення кількох сховищ для станів. Наприклад в нашому випадку буде зручно мати глобальне сховище для станів застосунку та сховище для завдань, адже це пришвидшує та робить зручнішую розробку, тому що часто буває необхідно винести певні дані в глобальну видимість, щоб не робити зайві запити чи робити велику вкладеність даних.

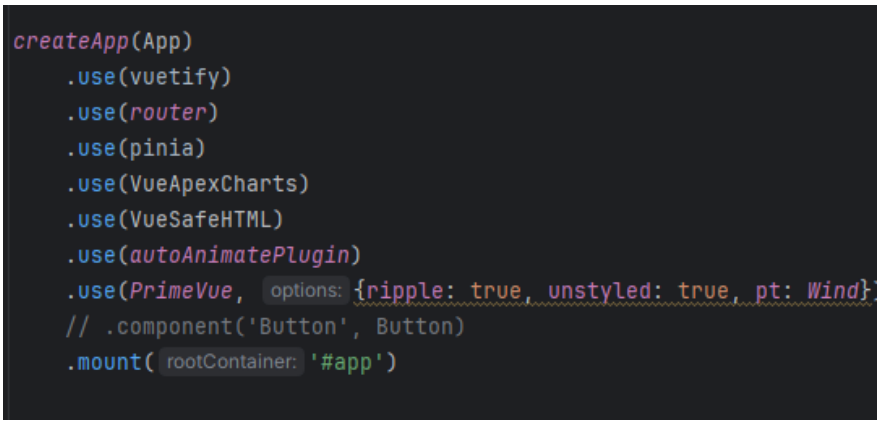

Рисунок 2.3.2.3 – Конфігурації проєкту Vue.

Далі опишемо процес створення кількох основних компонетів.

На мою думку, найважливішою буде сторінка із задачами. На ній користувач буде проводити найбільше часу, контролюючи свої задачі та проєкти. Слідуючи принципу DRY (don’t repeat yourself) даний компонент також буде використовуватися для відображення сторінки проєкту. Це буде відбуватися в залежності від того, чи приходить контекст від роутера, а саме ідентифікатор проєкту, чи ні.

Після імпорту необхідних функцій та даних, створимо функцію для запитів до API на отримання необхідних даних із серверу.

```
const fetchData = async (url) => {  Show usages  new *
  try {
    const response = await fetch( input: `${API_URL}${url}`);
    if (!response.ok) {
      throw new Error('API request failed');
    }
    const data = await response.json();
    return data;
  } catch (error) {
    console.error(error);
  }
};

// Використання функції fetchData
onMounted( hook: async () => {
  if (route.params.id === undefined || route.params.id === null) {
    console.log('Задачі')
    pageTitle = 'Задачі';
    tasks.value = await fetchData( url: '/tasks');
  } else {
    project = await fetchData( url: `/projects/${route.params.id}`);
    tasks = computed( getter: () => {
      return tasks.filter((task) => task.relatedProject == route.params.id);
    });
  }
});
```

Рисунок 2.3.2.4 – Функція для отримання даних через API.

Даний фрагмент коду реалізовує отримання завдання чи проєкту з API і зберігання їх у відповідні змінні. Далі з цих змінних дані виводяться в шаблон. Список завдань виводить через цикл, використовуючи компонент Task, який являє собою завдання. Дані в цей компонент передаються через пропси, і будуть доступні для використання всередині нього. Готова сторінка буде мати наступний вигляд:

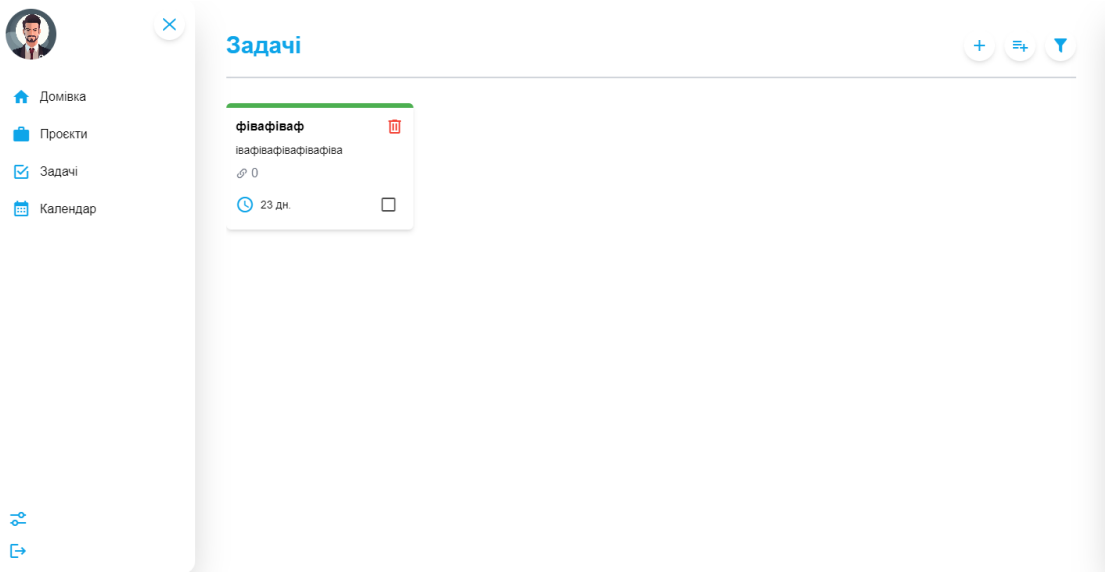

Рисунок 2.3.2.5 – Сторінка із задачами.

Окрему увагу хотів би також зосередити на ще на двох сторінках.

Перша з них це сторінка логіну. Ця сторінка Vue-застосунку являє собою сторінку, де користувачі входять у свої облікові записи. Спершу компонент включає імпорт необхідних модулів і компонентів, таких як ref і useRouter з Vue, а також useAppStore для доступу до стану додатку. Для авторизації використовується функція login, яка надсилає запит на сервер та в разі входу, отримує access_token та refresh_token. Далі ці дані записуються у сховище. Якщо користувач успішно авторизується, він перенаправляється на головну сторінку за допомогою маршрутизатора vue-router.

```
const login = async () => {  Show usages   zerrty
  try {
    const response = await axios.post('/api/token/', {
      username: username.value,
      password: password.value,
    });

    const { access, refresh } = response.data;
    localStorage.setItem('access_token', access);
    localStorage.setItem('refresh_token', refresh);
    axios.defaults.headers.common['Authorization'] = `Bearer ${access}`;
    appStore.isAuthenticated = true;
    router.push({ name: 'home' });
  } catch (error) {
    console.error('Login failed:', error);
  }
};
```

Рисунок 2.3.2.6 – Функція логіну.

Основний контент сторінки складається з форми входу, розташованої по центру сторінки. Ця форма має заголовок, текстові поля для введення імені та пароля, і кнопку для входу. Текстове поле для пароля оснащене іконкою, яка дозволяє користувачу показувати або приховувати пароль, натискаючи на іконку очей. Натискання на кнопку входу викликає функцію login, яка встановлює стан авторизації користувача і перенаправляє його на головну сторінку.

Форма для входу розміщена в центрі екрану і має стильний дизайн з використанням градієнтів та напівпрозорого фону. Три SVG-графіки розташовані по краях сторінки, додаючи декоративні елементи та створюючи сучасний вигляд.

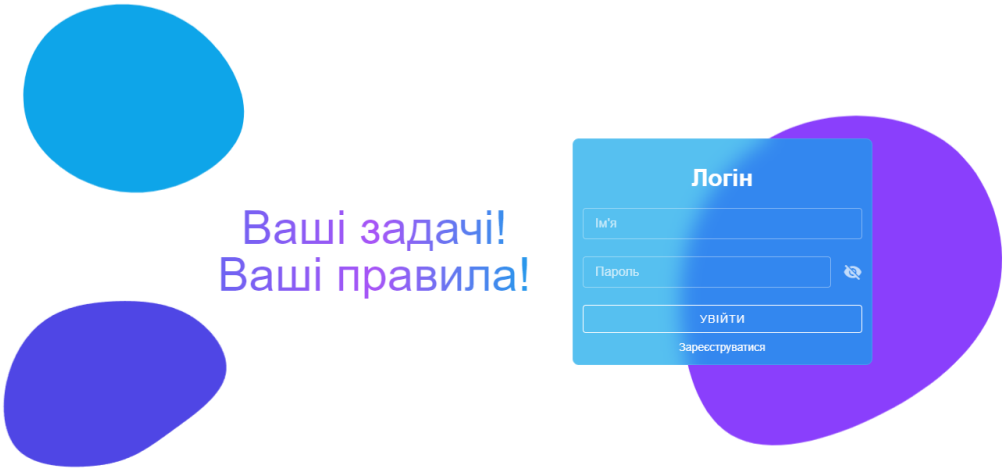

Рисунок 2.3.2.7 – Сторінка логіну.

Далі розглянемо сторінку з календарем. Ця сторінка є частиною системи управління завданнями, де користувачі можуть бачити свої задачі в календарному вигляді. Основна функціональність сторінки включає відображення завдань у різних режимах перегляду, таких як місячний, тижневий або денний.

Спочатку, коли сторінка завантажується, викликається функція fetchTasks, яка робить запит до сервера за списком завдань. Ці завдання зберігаються у змінній tasks. Після цього, у функції getEvents, завдання перетворюються на події календаря, які відображаються з використанням компоненту v-calendar.

Для кожного завдання визначається початкова та кінцева дата з використанням бібліотеки moment. Залежно від того, чи завершене завдання, йому присвоюється певний колір: зелений для завершених і червоний для незавершених завдань. Ці події зберігаються у змінній events, яка потім використовується для рендерингу календаря.

Користувач може змінювати режим перегляду календаря за допомогою селекторів, які дозволяють вибирати між місячним, тижневим та денним режимами, а також налаштовувати робочі дні. Все це забезпечує гнучкість і зручність в управлінні завданнями.

Додатково, на сторінці є кнопка для створення нових завдань, яка розташована внизу праворуч і відображається як плаваюча кнопка. Це дозволяє користувачам швидко додавати нові завдання, не покидаючи поточної сторінки.

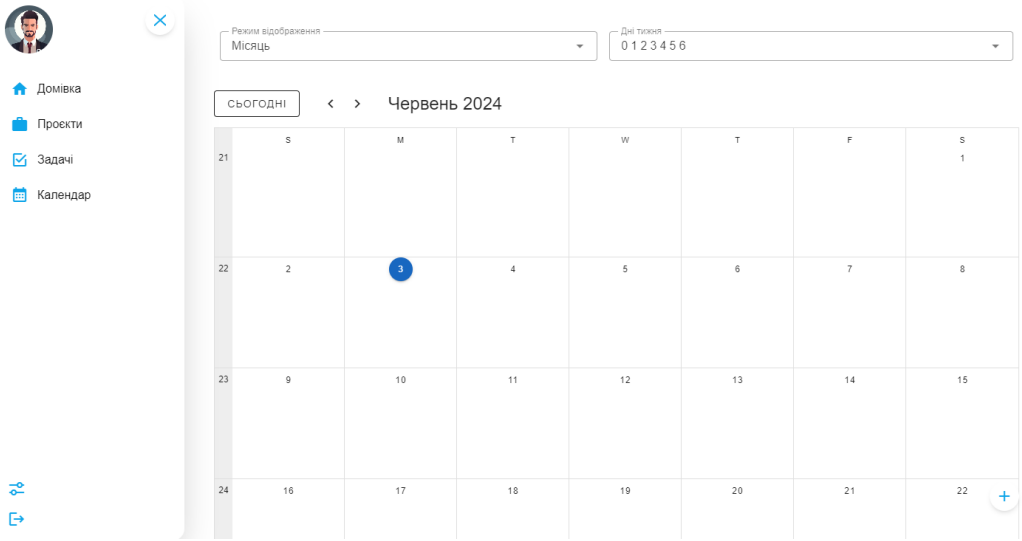

Рисунок 2.3.2.8 – Сторінка календаря.

Отже, в цьому пункті було детально описано процес конструювання основних компонентів клієнтської частини системи тайм-менеджменту. Під час конструювання було налаштовано проєкт на Vue та Vite, створено компоненти для відображення елементів інтерфейсу та цілих сторінок, прописані основні методи взаємодії із серверною частиною.

2.3 Тестування та верифікація вимог

Після успішної розробки системи надзвичайно важливо переконатися, що вона відповідає вимогам, які висвітлені при аналізі вимог. На цьому етапі ми зосередимося на функціональному тестуванні, яке дозволяє перевірити, чи система справді виконує поставлені перед нею задачі. Крім того, буде перевірено показники оптимізованості та швидкості веб-застосунку.

1. Створення завдання:

Назва: Створення нового завдання

Умова: Користувач авторизований в системі.

Кроки:

1) Відкрити сторінку "Задачі".
2) Натиснути кнопку "Створити завдання".
3) Ввести назву завдання.
4) Описати завдання.
5) Встановити термін виконання.
6) Натиснути кнопку "Зберегти".

Очікуваний результат: Нове завдання з заданими параметрами з'являється в списку завдань.

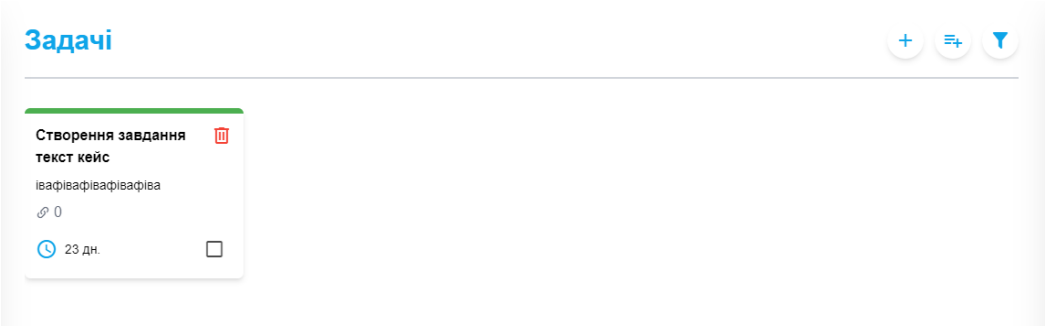

Рисунок 2.3.1 – Результат створення завдання.

2. Створення списку:

Назва: Створення нового списку завдань

Умова: Користувач авторизований в системі.

Кроки:

1) Відкрити сторінку "Задачі".
2) Натиснути кнопку "Створити список".
3) Ввести назву списку.
4) Натиснути кнопку "Зберегти".

Очікуваний результат: Новий список з заданими параметрами з'являється в списку списків.

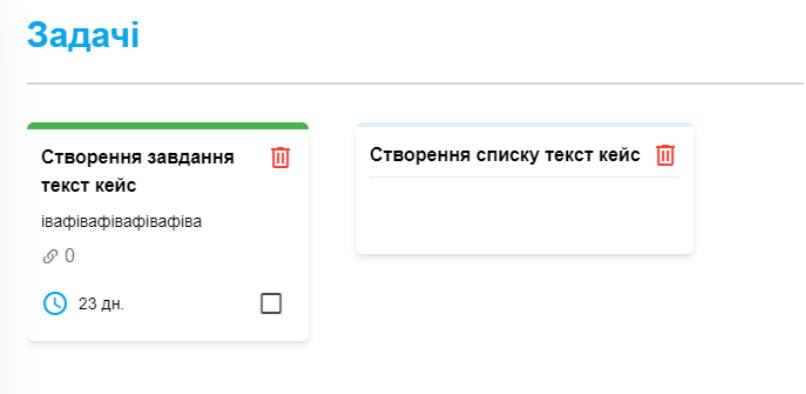

Рисунок 2.3.2 – Результат створення списку.

3. Створення проєкту:

Назва: Створення нового проєкту

Умова: Користувач авторизований в системі.

Кроки:

1) Відкрити сторінку "Проєкти".
2) Натиснути кнопку "Створити проєкт".
3) Ввести назву проєкту.
4) Ввести опис проєкту.
5) Ввести терміни проєкту
6) Натиснути кнопку "Зберегти".

Очікуваний результат: Новий проєкт з заданими параметрами з'являється в списку проєктів.

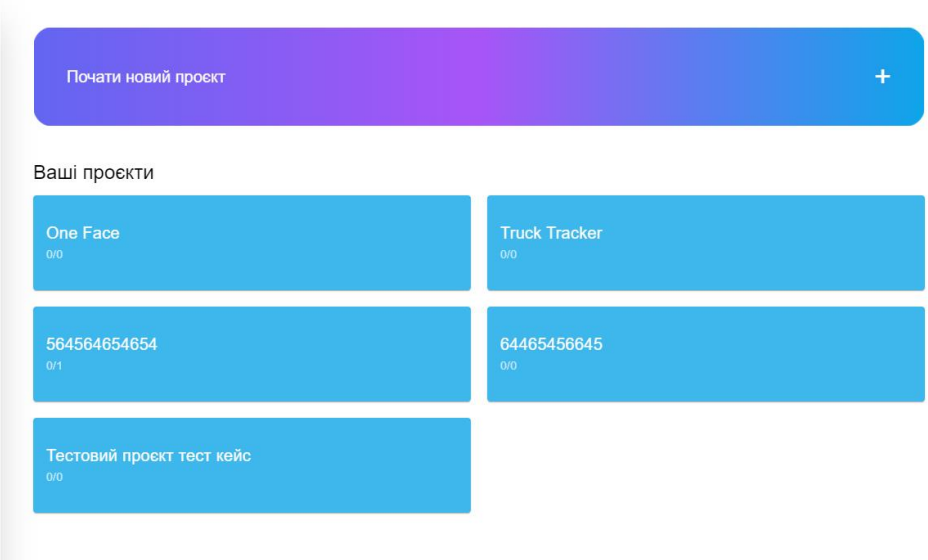

Рисунок 2.3.3 – Результат створення нового проєкту

4. Видалення завдання:

Назва: Видалення завдання

Умова: Користувач авторизований в системі та є власником завдання.

Кроки:

1) Відкрити сторінку "Задачі".
2) Обрати завдання для видалення.
3) Натиснути кнопку "Видалити".

Очікуваний результат: Обране завдання видаляється зі списку завдань.

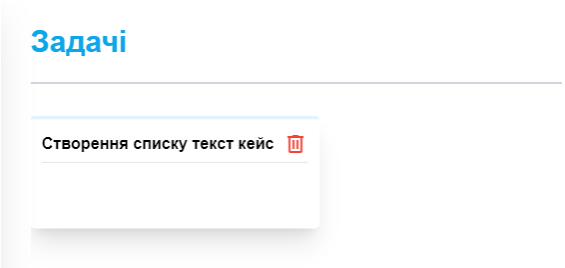

Рисунок 2.3.4 – Результат видалення завдання.

5. Виконання завдання:

Назва: Виконання завдання

Умова: Користувач авторизований в системі та є виконавцем завдання.

Кроки:

1) Відкрити сторінку "Задачі".
2) Обрати завдання для виконання.
3) Натиснути кнопку "Виконати".

Очікуваний результат: Обране завдання мітить як виконане, змінюється його статус.

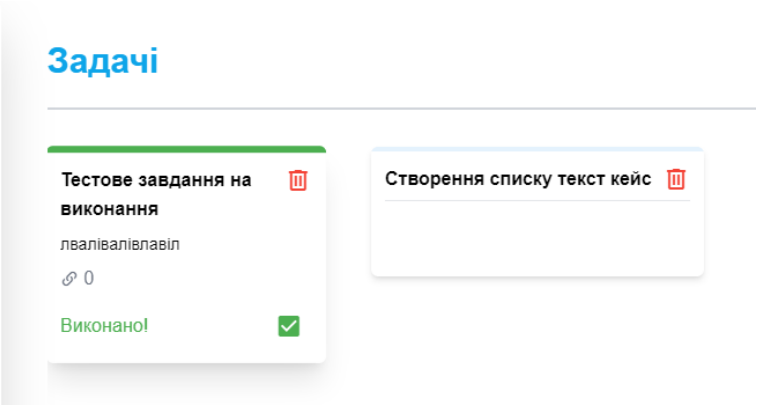

Рисунок 2.3.5 – Результат виконання завдання.

Для перевірки оптимізації веб-застосунку буде використовуватися спеціалізований інструмент LightHouse. Цей інструмент йду разом з інструментами розробника в браузері Google Chrome та інших. Для перевірки необхідно відкрити інструменти розробника клавішею F12 та обрати відповідний таб, на якому натиснути кнопку «Аналіз». Результати перевірки наведені на рисунку нижче:

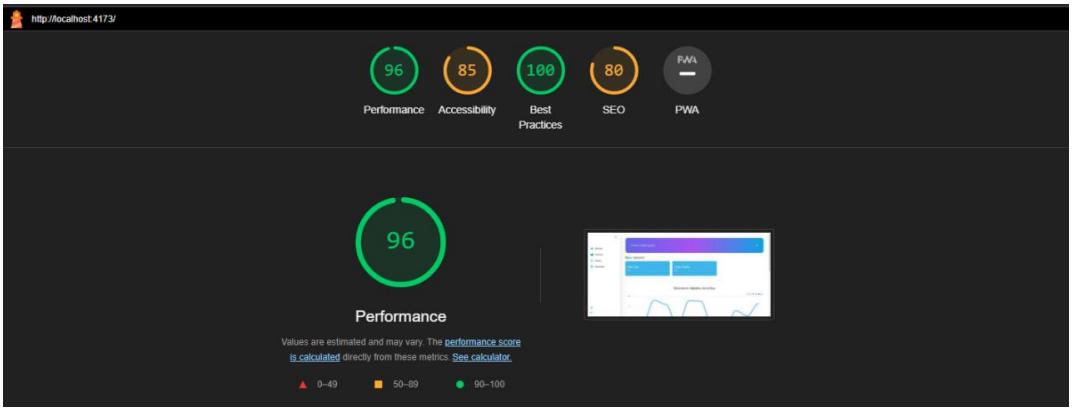

Рисунок 2.3.6 – Результати перевірки LightHouse.

Як можна бачити з результатів, сайт пройшов по всім основник показникам і знаходиться в зеленій зоні оптимізації, що є чудовим результатом.

Отже, в цьому розділі було спроєктовано архітектуру веб-застосунку. Обрано найоптимальнішу для нього архітектуру, а також підхід до реалізації клієнтської частини у вигляді Single page application. Було спроєктовано головні класи системи та структуру клієнтської частини веб-застосунку: зв’язки між сторінками та основні сценарії взаємодії.

Крім того було реалізовано серверну частину веб-застосунку. Реалізовані основні класи системи, створені відповідні моделі в базі даних. Створено програмний інтерфейс застосунку (API) для взаємодії з клієнтською частиною, налаштовано Celery та відповідні сигнали для надсилання сповіщень на електронну пошту користувача за певний час до дедлайну.

Також було описано процес створення клієнтської частини за допомогою фреймворку Vue та інструменту Vite. Розглянута реалізація основних компонентів інтерфейсу користувача та функій. Створений повний перелік сторінок, які були визначені при проєктуванні структури.

І на завершення було проведене функціональне тестування основних функціональних вимог застосунку, а також пройдено текст оптимізації в інструменті LightHouse, щоб верифікувати головні вимоги до системи.

3 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Долікарська допомога при ураженні електричним струмом

Небезпека ураження струмом може загрожувати людині як вдома, так і на вулиці. Виявити пошкоджений або оголений провід, що знаходиться під напругою, надзвичайно важко, оскільки він не видає жодних звуків, запахів і візуально не відрізняється від неактивного проводу. Тому дуже важливо пам'ятати правила електробезпеки, щоб уникнути травм.

Людина, яка піддалася дії струму, не може покликати на допомогу або самостійно звільнитися від предмету, що став причиною удару струмом. Дотик до струмопровідних частин найчастіше призводить до судом м'язів, які обмежують можливість рухатися чи говорити. Зазвичай про небезпеку свідчить несподіване падіння людини на вулиці, неприродне відкидання від джерела струму, раптова втрата свідомості, судоми, мимовільне скорочення м'язів і опіки з чіткими межами на тілі.

Перша долікарська допомога – це комплекс простих термінових дій, спрямованих на збереження здоров’я і життя потерпілого [17].

Перш за все, при ураженні електричним струмом необхідно відключити електроустановку. За неможливості відключення, відтягнути потерпілого за сухий одяг у безпечне місце. З метою власної безпеки особи, яка надає допомогу, можна використовувати підручні предмети з ізоляційних матеріалів (діелектричні рукавички, суху дошку, палицю тощо [18].

Захистити себе при відтягуванні потерпілого від джерела струму дуже важливо. Для цього можна використовувати сухий одяг потерпілого, уникаючи дотику до металевих предметів і відкритих частин тіла. Не можна торкатися взуття потерпілого, оскільки воно може бути вологим і проводити струм. Той, хто надає допомогу, повинен одягнути діелектричні рукавиці або обмотати руки шарфом, надіти рукав піджака або пальта. Для ізоляції можна стати на гумовий килимок або суху дошку.

Для якісного непрямого масажу серця потрібно починати якомога швидше, натискати на нижню частину грудини зі швидкістю 100-120 натисків на хвилину, глибиною 5-6 см, уникаючи переривань і дозволяючи грудній клітці повністю випрямитися після кожного стиснення. Не можна спиратися на груди. Натискання слід виконувати на твердій поверхні.

На кожні 2 вдування треба проводити 15 натискань на грудину. За 1 хв. необхідно зробити не менше 60 натискань і 12 вдувань [18].

При наявності автоматичного зовнішнього дефібрилятора (АЗД) потрібно увімкнути його, прикріпити електродні прокладки до оголеної грудної клітки потерпілого і дотримуватись інструкцій пристрою.

Незалежно від того, чи прийшов до тями потерпілий, слід викликати лікаря за номером 103. Якщо потерпілий приходить до тями, його не можна відпускати додому без спостереження, оскільки наслідки ураження струмом можуть проявитися пізніше і призвести до серйозніших проблем.

3.2 Питання безпеки при експлуатації електроустановок високих, надвисоких та ультрависоких частот.

У промисловості широко застосовуються прилади, пристрої та обладнання, робота яких супроводжується утворенням електромагнітних випромінювань різних частот - від іонізуючих до радіохвиль. Це зумовлює вплив цих випромінювань на персонал, який обслуговує установки, а також на осіб, що знаходяться поблизу. Захист від шкідливої дії випромінювань стає особливо важливим, адже кожна людина, яка проживає або тимчасово перебуває на території України, має право на захист від іонізуючого випромінювання. Це право забезпечується комплексом заходів, що включають запобігання впливу іонізуючого випромінювання на організм людини в межах встановлених доз, компенсацією за перевищення цих меж і відшкодуванням шкоди, заподіяної випромінюванням.

Електромагнітні поля негативно впливають на організм людини, яка працює з джерелом випромінювання, і на населення, що проживає поблизу цих джерел. Основними джерелами електромагнітних полів (ЕМП) у системах «людина – побутове середовище» і «людина – виробниче середовище» є електронно-променеві трубки телевізорів і моніторів комп'ютерів, мікрохвильові печі, що використовують випромінювання надвисокої частоти (НВЧ), а також промислові електричні й радіотехнічні пристрої та системи [19].

У діапазоні промислових частот (зокрема 50 Гц) вплив електричної і магнітної складових поля можна розглядати окремо. В будь-якій точці ЕМП промислової частоти енергія магнітної складової, яка поглинається тілом людини, значно менша від енергії електричної складової. Тому негативний вплив на організм в основному обумовлений електричною складовою поля.

Ступінь і характер впливу електромагнітних полів на організм людини визначаються напруженістю поля, щільністю потоку енергії, частотою випромінювання тривалістю впливу і його режимом, площею поверхні тіла людини, що опромінюється тощо [19].

ЕМП можуть викликати як гострі, так і хронічні порушення фізіологічних функцій організму, зокрема впливаючи на нервову систему, структуру головного та спинного мозку, серцево-судинну систему. В більшості випадків ці порушення є зворотними, але тривалий вплив може призвести до стійких змін та захворювань.

У 60-х роках науковці почали вивчати феномен, коли люди, опромінені імпульсами НВЧ коливань, чули звуки, що сприймалися як щебет, цвірінчання чи дзюркіт у голові. Це явище викликало зацікавлення і стало предметом систематичних досліджень на людях та тваринах.

Електромагнітне випромінювання як хвороботворний чинник слід розглядати на основі клінічних та експериментальних даних. Спільну дію випромінювань різного діапазону можна класифікувати як окрему радіохвильову хворобу, тяжкість якої залежить від напруженості ЕМП, тривалості впливу, частотного діапазону, умов зовнішнього середовища та стану організму. Поряд із

радіохвильовою хворобою зростає ризик загальних захворювань, зокрема органів дихання і травлення, навіть при невеликій інтенсивності ЕМП.

Дія електромагнітних випромінювань в діапазоні 30 кГц – 300 МГц викликає загальну слабкість, підвищену втому, сонливість, порушення сну, головний біль і біль в ділянці серця. Спостерігаються роздратованість, втрата уваги, сповільнення рухово-мовних реакцій, симптоми порушення роботи окремих органів, зміни в діяльності серцево-судинної системи, показниках білкового та вуглеводного обміну, складі крові та клітинах.

Змінне електричне поле високої напруженості викликає нагрівання тканин організму людини. В результаті цього тепловий ефект є наслідком поглинання енергії ЕМП. Надлишкова теплота, що виділяється в органах тіла людини, збільшує навантаження на механізм терморегуляції. Починаючи з певної межі поглиненої енергії ЕМП, організм не справляється з відведенням теплоти від окремих органів тіла людини і внаслідок цього їхня температура може підвищуватися. Перегрів ЕМП шкідливий, особливо для тканин зі слабко розвиненою судинною системою чи недостатнім кровообігом (око, мозок, нирки, жовчний і сечовий міхури) [19].

Дослідження впливу НВЧ випромінювань показують, що інтенсивність близько 20 мкВт/см² знижує частоту пульсу та артеріальний тиск, а при збільшенні інтенсивності відбуваються електрокардіографічні зміни і гіпотонія. Подальше збільшення інтенсивності до 6 мВт/см² викликає зміни у статевих залозах, складі крові, помутніння кришталика та інші фізіологічні зміни.

Випромінювання інтенсивністю до 100 мВт/см² викликає стійкі зміни серцево-судинної системи і двосторонню катаракту. Подальше опромінення може призвести до значних пошкоджень тканин і втрати зору. Інтенсивне НВЧ опромінення викликає сльозотечу, подразнення і звуження зіниці ока. При тривалому впливі пошкодження набувають незворотного характеру, зокрема в діапазоні 1-10 ГГц найбільш чутливим є кришталик ока. Сильні пошкодження кришталика викликаються тепловим впливом НВЧ, і при великій інтенсивності пошкодження спостерігаються по всьому об’єму кришталика.

ВИСНОВКИ

У даній кваліфікаційні роботі бакалавра було створено систему тайм-менеджменту на основі веб-застосунку з використанням фреймворку Vue для клієнтської частини та фреймворку Django для серверної частини. Створена системи відоповідає усім сучасним вимогам веб-розробки.

При створенні даної системи було проведено детальний огляд предметної області, в якому було визначено основних конкурентів, після чого проведено їхній порівняльний аналіз, щоб визначити переваги та недоліки цих систем. На основі результатів аналізу було сформовано вимоги до системи тайм-менеджменту, в яких намагався втілити основні переваги конкурентів а також ті можливості, яких їм бракувало. Разом з цими вимогами було спроєктовано відношення між акторами і прецедентами, створено діаграму варіантів використання. Також було обрано методологію розробки, а саме Scrum, який допоміг розбити процес розробки на невеличкі спринти, під час яких вівся регулярний огляд результатів праці, що давало можливість тримати все під контролем.

При проєктуванні було спроєктовано архітектуру веб-застосунку. Обрано найоптимальнішу для нього архітектуру, а також підхід до реалізації клієнтської частини у вигляді Single page application. Було спроєктовано головні класи системи та структуру клієнтської частини веб-застосунку: зв’язки між сторінками та основні сценарії взаємодії.

Крім того було реалізовано серверну частину веб-застосунку. Реалізовані основні класи системи, створені відповідні моделі в базі даних. Створено програмний інтерфейс застосунку (API) для взаємодії з клієнтською частиною, налаштовано Celery та відповідні сигнали для надсилання сповіщень на електронну пошту користувача за певний час до дедлайну.

Також було описано процес створення клієнтської частини за допомогою фреймворку Vue та інструменту Vite. Розглянута реалізація основних компонентів

інтерфейсу користувача та функій. Створений повний перелік сторінок, які були визначені при проєктуванні структури.

І на завершення було проведене функціональне тестування основних функціональних вимог застосунку, а також пройдено текст оптимізації в інструменті LightHouse, щоб верифікувати головні вимоги до системи.

У результаті виконання даної кваліфікаційної роботи було спроєктовано та сконструювано надійну, швидку та ефективну систему тайм-менджемент, із закладеними можливостями масштабування та реалізації нового функціоналу. Було продемонстровано вміння проєктування як загальної архітектури системи, так і дрібніших її компонентів. Крім того, також висвітлені навички у реалізації сучасних веб-застосунків із використанням передових технологій та фреймворків. Створену систему можна використовувати як для невеликих робочих проєктів, так і для домашнього користування.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Офіційна сторінка Microsoft To-do. URL: https://www.microsoft.com/uk-ua/microsoft-365/microsoft-to-do-list-app (дата звернення 10.05.2024)

2. Офіційна сторінка Atlassian Trello. URL: https://trello.com/ (дата звернення 10.05.2024)

3. Ghindani K. Everything You Need to Know About Front-End Development: A Step-by-Step Approach to Getting Started with Programming. Amazon Kindle Direct Publishing, 2022. 283 p.

4. Thanh N. L. What Is Backend Development?. Medium. URL: https://medium.com/@namtheartist95/what-is-backend-development-bcc6a15f8472 (date of access: 28.05.2024).

5. O'Grady B. What is a Framework? Why We Use Software Frameworks. Code institute. URL: https://codeinstitute.net/global/blog/what-is-a-framework/ (date of access: 28.05.2024).

6. Stack Overflow Insights - Developer Hiring, Marketing, and User Research. URL: https://insights.stackoverflow.com/trends?tags=reactjs,vue.js,angular,svelte,angularjs,vuejs3 (date of access: 28.05.2024).

7. Most Popular Backend Frameworks – 2012/2023. Statistics & Data. URL: https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2023/ (date of access: 29.05.2024).

8. What is scrum and how to get started. Collaboration software for software, IT and business teams. URL: https://www.atlassian.com/agile/scrum (date of access: 31.05.2024).

9. Сергій Г. Архітектура програмного забезпечення: все що треба знати. Wezom. URL: https://wezom.com.ua/ua/blog/arhitektura-programmnogo-obespecheniya (дата звернення: 24.05.2024).

10. Luchaninov Y. Web Application Architecture: Choosing the Right Type in 2024. MobiDev. URL: https://mobidev.biz/blog/web-application-architecture-types (дата звернення: 17.06.2024).

11. Петрик М. Р. Введення в ООП. Об'єктно-орієнтоване програмування (CS1021). URL: https://dl.tntu.edu.ua/content.php?cid=152806 (дата звернення: 01.06.2024).

12. IONOS editorial team. UML State Machine Diagrams. IONOS Digital Guide. URL: https://www.ionos.com/digitalguide/websites/web-development/uml-state-machine-diagram/ (date of access: 02.06.2024).

13. Pyenv: Встановіть кілька версій Python у вашій системі. Ubunlog. URL: https://ubunlog.com/uk/pyenv-встановлює-кілька-версій-python-у-вашій-системі/ (дата звернення: 02.06.2024).

14. Migrations | Django documentation. Django Project. URL: https://docs.djangoproject.com/en/5.0/topics/migrations/ (date of access: 02.06.2024).

15. Celery що це: основні концепції, архітектура та налаштування. FoxmindEd. URL: https://foxminded.ua/celery-shcho-tse/ (дата звернення: 03.06.2024).

16. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 – Інженерія програмного забезпечення (Освітньо-професійна програма - «Програмне забезпечення систем», Освітньо-наукова програма - «Інженерія програмного забезпечення») для студентів усіх форм навчання / Упор.: М.Р. Петрик, Д.М. Михалик, О.Ю. Петрик, Г.Б. Цуприк - Тернопіль: ТНТУ, 2020-51с.

17. Безпека життєдіяльності та цивільний захист [Електронний ресурс]: підручник для студ. спеціальностей з природничих, соціально-гуманітарних наук та інженерно-комунікаційних технологій / О. Г. Левченко, О. В. Землянська, Н. А. Праховнік, В. В. Зацарний; КПІ ім. Ігоря Сікорського. – Електронні текстові данні (1 файл: 10,2 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2019. – 267 с.

18. Стиценко Т.Є., Пронюк Г.В., Сердюк Н.М., Хондак І.І. «Безпека життєдіяльності»: навч. посібник / Т.Є Стиценко, Г.В. Пронюк, Н.М. Сердюк, І.І. Хондак. – Харків: ХНУРЕ, 2018. – 336 с.

19. Безпека життєдіяльності та охорона праці : підручник : у 2 ч. / Я. О. Сєріков, Л. Ф. Коженевські, М. В. Хворост ; Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова [та ін.]. – Харків : ХНУМГ ім. О. М. Бекетова ; Краків : ЄАС, 2021.

20. Мудрик І. Проблеми аналізу унікальності контенту веб-сайтів у роботі SEO-оптимізатора / І. Мудрик, Р. Карагодін., 2023. – 161 с. (ТНТУ).

ДОДАТКИ

ДОДАТОК А

Тези для VII Міжнародної студентської науково-технічної конференції

УДК 004.41
Грицишин П. – ст. гр. СП-4, Мудрик І. - д.філософії
Тернопільський національний технічний університет імені Івана Пулюя

Проєктування та розробка системи тайм-менеджменту з використанням фреймворку Vue

Науковий керівник: д. ф., с. в. Мудрик І. Я.

Hrytsyshyn P., Mudryk I.
Ternopil Ivan Puluj National Technical University

Design and development of a time management system using Vue framework

Supervisor: PhD, Senior Lecturer Mudryk I. Y.

Ключові слова: веб-розробка, тайм менеджмент, реактивність, JavaScript, Vue
Keywords: web development, time management, responsiveness, JavaScript, Vue

У сучасному динамічному житті невід’ємною частиною є час та його менеджмент. Саме тому платформа, де кожен зможе відстежити свої особисті задачі, буде корисним інструментом для кожного.

При плануванні часу інструмент має бути швидким та ефективним, адже який сенс у такому планувальнику, якщо планування займає багато часу. Вирішенням можуть стати сучасні JavaScript-фреймворки, наприклад, Vue. Vue (вимовляється /vjuː/) - це JavaScript-фреймворк для створення користувацьких інтерфейсів. Він базується на стандартних HTML, CSS та JavaScript і надає декларативну, компонентну модель програмування, яка допомагає ефективно розробляти користувацькі інтерфейси будь-якої складності. [1]

Vue.js здобув велику популярність серед розробників завдяки своїй простоті та гнучкості. Його легко вивчити, навіть якщо ви новачок у світі фронтенд-розробки, завдяки зрозумілому синтаксису та добре структурованій документації.

Незважаючи на простоту, Vue.js дозволяє створювати як невеликі інтерактивні елементи, так і складні односторінкові додатки. Його компонентний підхід сприяє модульності та повторному використанню коду, що спрощує розробку та підтримку проектів.

Vue.js також славиться своєю продуктивністю. А реактивність фреймворку автоматично оновлює інтерфейс при зміні даних, позбавляючи розробників від ручного маніпулювання DOM.

ДОДАТОК Б

ДІАГРАМИ ТА РИСУНКИ

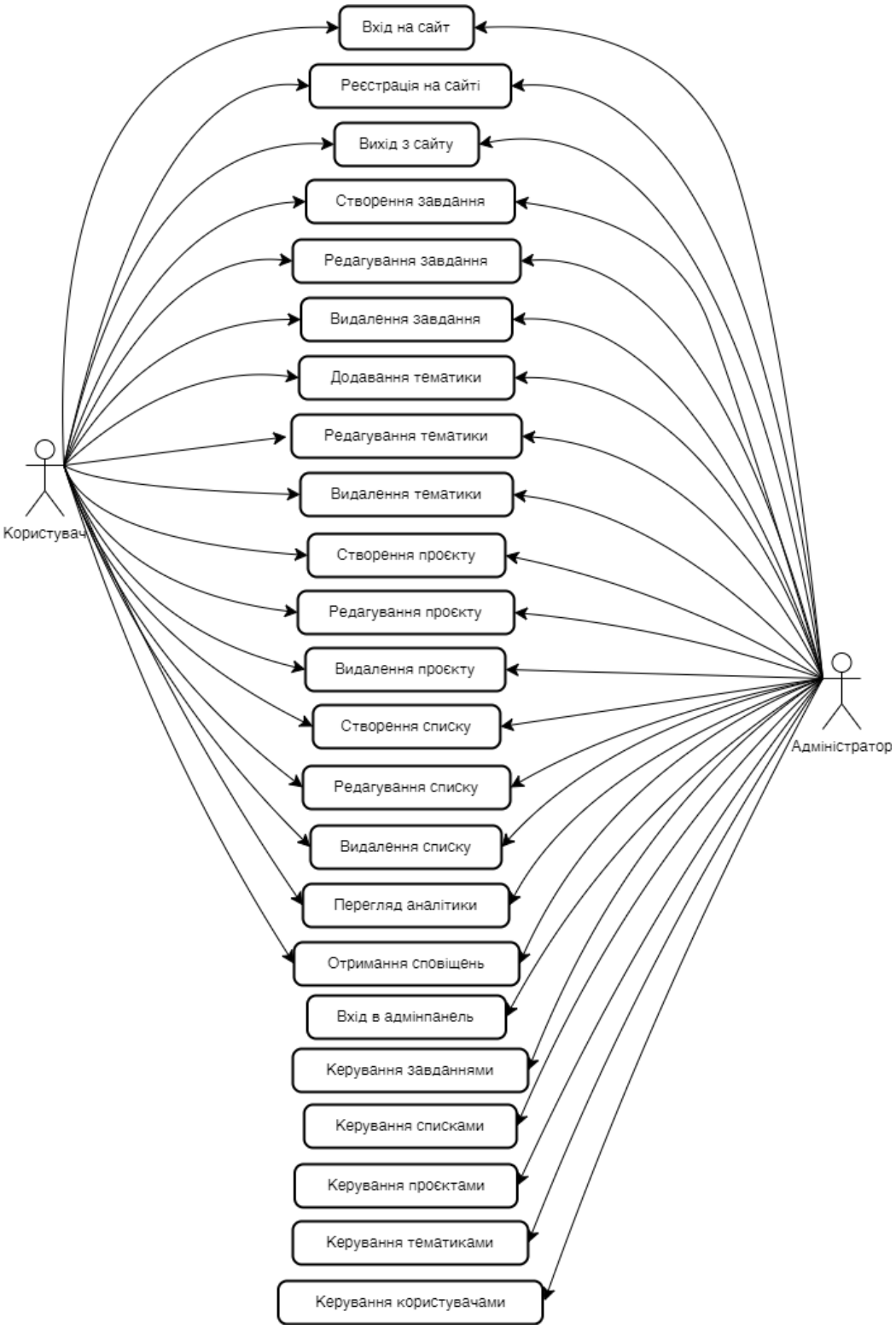

Рисунок Б.1 – Діаграмі варіантів використання

Рисунок Б.2 – Діаграма класів.

ДОДАТОК В
ДИСК

1. Репозиторій клієнтської частини. URL: https://github.com/zerrty/task-manager.
2. Репозиторій серверної частини. URL: https://github.com/zerrty/taskmanagerback.