# РЕФЕРАТ

Кваліфікаційна робота на здобуття освітнього ступеню «бакалавр» за спеціальністю 121 – Інженерія програмного забезпечення. Тернопільський національний технічний університет ім. Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СПс-43, 2023 рік. Пояснювальна записка до кваліфікаційної роботи на здобуття освітнього ступеню «бакалавр» містить: 49 с., 9 рис., 10 табл., 4 додатків. Тема роботи: Розробка веб сайту для проведення вікторин з використанням платформи Node Js.

У випускній роботі бакалавра детально розглянуті основні аспекти розробки системи для онлайн тестування та вікторин, використовуючи платформу Node.js. Проведено аналіз існуючих веб додатків для проведення онлайн тестування, а також виявлено їх переваги та недоліки. Спроектовано схему бази даних MongoDB, яка задовольняє всі потреби системи. Розроблено UML діаграми які описують колекції в базі даних та взаємодію компонентів системи.

Ключові слова: Node.js, онлайн тестування, вікторина, клієнт-серверна архітектура, MongoDB, UML, React.js

# ANNOTATION

Qualification work for the academic degree "Bachelor" in the specialty 121 - Software Engineering. Ternopil Ivan Puluj National Technical University, Faculty of Computer and Information Systems, Department of Software Engineering, Group SPs-43, 2023. Explanatory note for the qualification work for the academic degree "Bachelor" contains: 49 pages, 9 figures, 10 tables, 4 appendices. Work topic: Development of a website for conducting quizzes using the Node.js platform.

The bachelor's thesis extensively examines the main aspects of developing a system for online testing and quizzes using the Node.js platform. An analysis of existing web applications for online testing has been conducted, highlighting their advantages and disadvantages. A MongoDB database schema has been designed to meet all the system's needs. UML diagrams have been developed to describe the collections in the database and the interaction of system components.

Keywords: Node.js, online testing, quiz, client-server architecture, MongoDB, UML, React.js.

ЗМІСТ

# ВСТУП

У сучасному світі, де технології набувають все більшого значення в освіті та оцінці навичок, системи онлайн тестування стають необхідним інструментом для забезпечення ефективного та об'єктивного процесу оцінки. Ці системи дозволяють впровадити гнучкий та доступний спосіб проведення вікторин для широкої кількості людей у будь-якому місці та в будь-який час.

Застосування Node.js [1] для розробки системи онлайн тестування відкриває велику кількість можливостей. Node.js є потужним середовищем виконання JavaScript, яке базується на швидкій та ефективній роботі двигуна V8 [2].

В межах даної дипломної роботи будуть вивчені і реалізовані ключові аспекти системи онлайн тестування з використанням Node.js. Будуть розглянуті методи створення та збереження тестових питань, розроблено механізми автентифікації та авторизації користувачів, буде впроваджено механізми збереження результатів тестування та підтримки інтерактивності. Додатково, буде здійснена інтеграція системи з базою даних для забезпечення надійного збереження та обробки інформації.

Основною метою цієї роботи є створення високоефективної системи онлайн тестування, яка забезпечує зручний та надійний інструмент для проведення тестів в режимі онлайн. Ця система буде корисною для освітніх установ, де вона може використовуватись для оцінки знань студентів. Також вона буде цінним ресурсом для компаній, що займаються рекрутингом та підбором персоналу, оскільки система забезпечить зручний спосіб проведення тестування та оцінки кандидатів.

# 1 АНАЛІТИЧНА ЧАСТИНА

## 1.1 Аналіз предметної області

В нас час спостерігається тенденція на цифровізацію багатьох аспектів нашого життя. Традиційні методи тестування на паперових аркушах та ручна їх перевірка, стають все менш ефективними. На їх заміну приходять системи онлайн тестування, які дозволяють користувачам проходити тести в будь-який зручний для них час, без прив’язки до певної локації, забезпечуючи гнучкість в часі та місці.

В списку нижче наведено основні переваги систем онлайн тестування:

1. Гнучкість та доступність: Системи онлайн тестування дозволяють користувачам проходити тести у будь-який зручний час та в будь якій локації. Вони відкривають можливості для навчання на відстані, дистанційної освіти та самонавчання.

2. Ефективність та економія часу: Онлайн тестування дозволяє автоматизувати процес оцінювання і спрощує процедуру перевірки відповідей. Це дозволяє фокусуватися на більш важливіших завданнях.

3. Об'єктивність та надійність: В автоматизованих системах тестування відсутній людський фактор. Отримані результати оцінювання чітко базуються на визначених правилах та алгоритмах, що забезпечує об'єктивність та надійність процесу.

4. Моніторинг та аналітика: Системи онлайн тестування здатні збирати дані про відповіді користувачів, що дозволяє проводити моніторинг та аналізувати результати. Це дає можливість сформувати певну статистику в залежності від потреб. Також це допомагає виявити слабкі місця студентів, якщо тести використовувалися в якості інструменту для оцінювання знань.

5. Складніші завдання та інтерактивність: Онлайн тестування дозволяє використовувати більшу варіативність в типах запитань та завдань, на відміну від традиційних тестових завдань на папері. Це стимулює приймати більш активну участь в проходженні тестування.

## 1.2 Огляд подібних проектних рішень

Розглянемо найпопулярніші системи онлайн тестування та їх основні особливості а також оцінимо їх основні переваги та недоліки.

Google Forms – це онлайн-інструмент від Google, який дозволяє створювати опитування та анкети. Основною метою Google Forms є збір відповідей від користувачів та збереження зібраних даних в зручному форматі. Сторінку створення тесту в Google Forms зображено на рисунку 1.1.

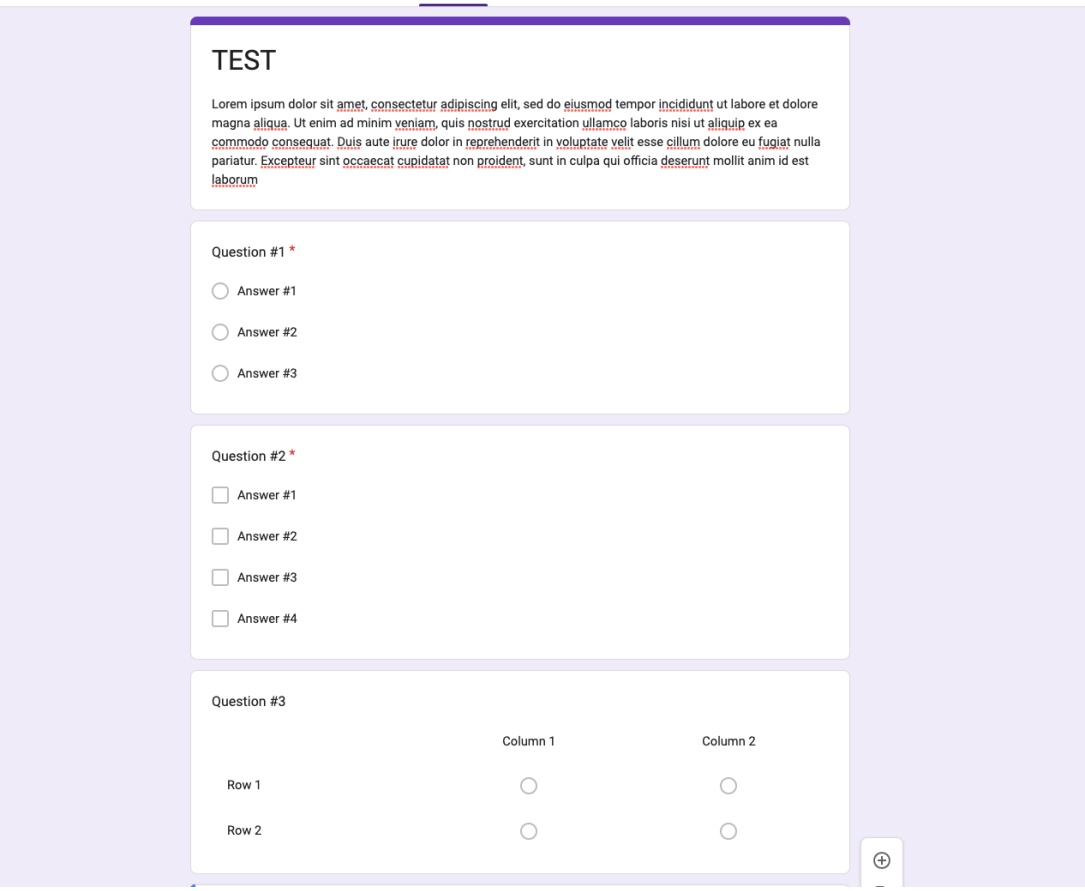



Рисунок 1.1 – Створення тесту в Google Forms

Переваги Google Forms:

1. Простота використання: Google Forms має зручний та інтуїтивно зрозумілий інтерфейс, що дозволяє швидко створювати тести.

2. Безкоштовність: Google Forms є безкоштовним інструментом, доступним для всіх користувачів з обліковим записом Google.

3. Інтеграція з іншими продуктами від Google: Google Forms легко інтегрується з іншими інструментами Google, такими як Google Sheets, що дозволяє автоматично експортувати дані з результатами тестів в зручному форматі.

4. Різноманітні типи питань: Google Forms пропонує достатню кількість типів питань, включаючи питання з варіантами відповідей, заповнення пропусків, матриці та інші. Це дозволяє що дозволяє створювати різноманітні тести.

5. Налаштування доступу: Google Forms надає можливість налаштовувати доступ до тестів, встановлювати обмеження на кількість повторних проходжень одного тесту від одного користувача.

Недоліки Google Forms:

1. Відсутність повного контролю над даними: При використанні Google Forms всі дані зберігаються на серверах Google. Це може бути проблематично в випадку використання чутливих даних.

2. Обмежена можливість настроювання логіки: Google Forms має обмежені можливості для налаштування логіки запитань та відповідей. Він не підтримує складну логіку, яка може бути потрібною в деяких специфічних випадках.

Найчастіше Google Forms використовують для проведення нескладних опитувань та тестувань.

Kahoot – це платформа для створення інтерактивних онлайн-вікторин, опитувань та проведення презентацій з елементами опитування в режимі реального часу. Сторінка створення тесту на платформі Kahoot показана на рисунку 1.2

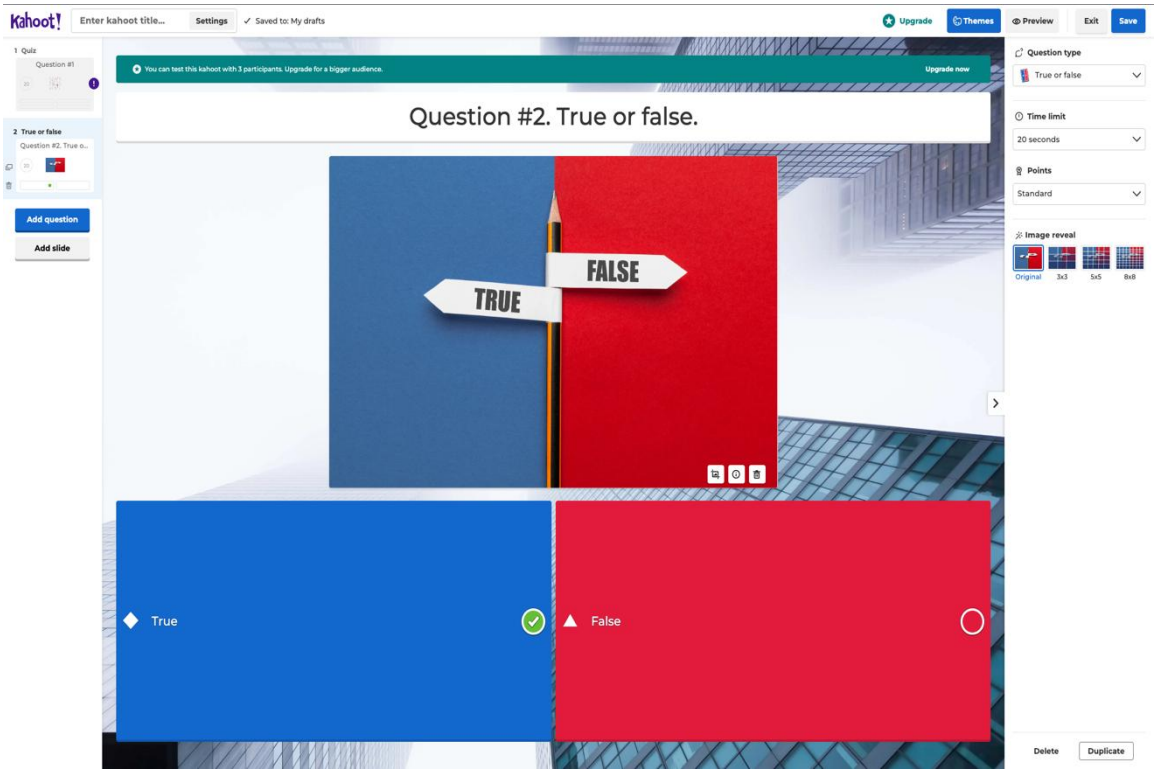



Рисунок 1.2 – Сторінка створення тесту на платформі Kahoot

Переваги Kahoot:

1. Інтерактивність: Kahoot дозволяє створювати інтерактивні запитання, які можуть бути використані для спільної роботи та обговорення. Також він надає можливість створювати вікторини в режимі реального часу. Він сприяє співпраці між студентами та викладачами, або доповідачем та аудиторією, дозволяючи їм працювати в команді, обмінюватися ідеями, налагоджувати комунікацію.

2. Легкість використання: Kahoot має простий та інтуїтивно зрозумілий інтерфейс, що робить його легким у використанні для всіх користувачів. Інтуїтивно зрозумілі інструменти дозволяють швидко створювати запитання та організовувати інтерактивні сесії.

3. Наявність готових шаблонів: Крім можливості створювати власні вікторини, опитування та тести, Kahoot надає доступ до великої кількості готових шаблонів, які можуть бути використані користувачем.

Недоліки Kahoot:

1. Фокус на швидкості: Концепція Kahoot базується на швидких відповідях, де на кожне запитання встановлюється певний ліміт часу(за замовчуванням 20 секунд).

2. Залежність від швидкого інтернету: Kahoot вимагає швидкого інтерне-з’єднання, оскільки більшість тестів проводиться в режимі реального часу.

3. Обмежена можливість збереження даних: Kahoot зберігає результати вікторини лише обмежений період часу.

Найчастіше платформу Kahoot використовують в навчальних цілях та для проведення інтерактивних презентацій.

# 2. ПРОЕКТНА ЧАСТИНА

## 2.1 Постановка завдання

Темою дипломного проекту є розробка веб сайту для проведення вікторин з використанням платформи Node Js. Необхідно розробити комплексну систему, яка дасть користувачам зручний та ефективний спосіб створення та проходження тестів різної складності.

Виходячи з аналізу предметної області можна скласти наступний список вимог до системи яка буде розроблятися:

1. Реєстрація та автентифікація: Користувачі повинні мати можливість створювати облікова записи та входити в систему.
2. Система управління тестами: Користувачі повинні мати можливість створювати тести з запитаннями та відповідями. Обов’язкова підтримка різних типів питань (одиночний вибір, множинний вибір, текстова відповідь, тощо). Також потрібно надати можливість обмежувати кількість часу на проходження тесту та кількість спроб для кожного користувача.
3. Проходження тесту: Користувачі повинні мати можливість пройти потрібний їм тест.
4. Оцінювання результатів: Система повинна автоматично перевіряти відповіді користувачів та виводити результати тесту після його закінчення.
5. Аналітика тестування: Після закінчення тесту результати мають зберігатися та бути доступними для перегляду користувачу який проходив тест, та користувачу який його створив. Адміністратор (тобто користувач який створив тест) повинен мати можливість переглядати статистику результатів тесту.

## 2.2 Технічні вимоги

Для розробки системи обрано клієнт-серверну архітектуру. Клієнт-серверна [3] архітектура в веб-розробці використовується для створення веб-сайтів та додатків. У цій архітектурі є два основних компоненти: клієнт і сервер.

Клієнт [4] – це програма або пристрій (наприклад, веб-браузер або мобільний додаток), який використовується користувачем для взаємодії з веб-сайтом чи додатком. Клієнт надсилає запити на сервер і отримує відповіді з необхідними даними. Він відображає ці дані на екрані та дозволяє користувачеві взаємодіяти з ними (наприклад, заповнювати форми, натискати кнопки тощо).

Сервер [5] – це комп'ютер або система, яка обробляє запити від клієнтів. Він надає необхідні ресурси (такі як HTML-сторінки, зображення, дані з бази даних) клієнтам, які звертаються до нього. Сервер обробляє запити клієнтів і відправляє їм відповіді, які можуть містити нові дані, оновлення або іншу інформацію, необхідну для клієнта.

Таким чином, клієнт-серверна архітектура в веб-розробці дозволяє розділити завдання між клієнтом і сервером. Клієнт забезпечує взаємодію користувача з веб-сайтом чи додатком, відображає дані та дозволяє користувачам взаємодіяти з ними. Такий підхід дозволяє покращити масштабованість.

Для обміну даними між клієнтом та сервером обрано мову запитів GraphQL [6, 7]. GraphQL (Graph Query Language) – це мова запитів, яка призначена для ефективного обміну даними між сервером та клієнтом. Основною її перевагою є гнучкість(оскільки клієнт може сам обрати які данні він хоче отримати від сервера) та типізація(клієнт може запросити або надіслати лише ті данні, які були визначені на сервері.).

В якості бази даних буде використовуватися MongoDB [8], яка ідеально підходить для розробки системи тестування завдяки своїй гнучкості. MongoDB - це потужна та гнучка система управління базами даних (СУБД) [9], яка належить

до категорії NoSQL [10] баз даних. Вона надає ефективне зберігання та обробку структурованих, напівструктурованих та неструктурованих даних.

Отже, для розробки системи обрано клієнт-серверну архітектуру, де клієнт (веб-браузер) взаємодіє з сервером, який надає необхідні дані. Для обміну даними між ними використовується мова запитів GraphQL, що забезпечує гнучкість та типізацію. В якості бази даних обрано MongoDB, яка надає можливість ефективного зберігання та обробки різноманітних даних.

## 2.3 Проектування бази даних

База даних для системи онлайн тестування використовується для зберігання всієї необхідної інформації, пов'язаної з тестами, питаннями, відповідями, результатами тестування та іншими даними. Вона дозволяє системі ефективно організовувати, керувати і швидко отримувати доступ до цих даних. Для проектування моделей даних в MongoDB було обрано ORM Mongoose.

ORM (Object-Relational Mapping) [11] – це підхід в програмуванні, який дозволяє розробникам працювати з базою даних, використовуючи об'єктно-орієнтований підхід. ORM-система надає можливість взаємодії з базою даних через об'єкти та класи, уникнувши прямої роботи з SQL-запитами та таблицями бази даних. ORM встановлює відповідність між об'єктами програми та записами у базі даних, автоматично виконуючи операції збереження, оновлення, видалення та вибірки даних. Це дозволяє розробникам працювати на вищому рівні абстракції, упускаючи деталі взаємодії з конкретною базою даних.

Для кожної колекції в базі даних буде певна кількість спільних полів. Список полів наведено у таблиці 2.1.

Таблиця 2.1 – Опис спільних полів для кожної схеми

| Назва поля | Тип | Опис | Обмеження | Значення NULL | Значення за замовчуванням |
|---|---|---|---|---|---|
| _id | ObjectId | Унікальний ідентифікатор документу | Лише унікальні значення | Ні | ObjectId |
| createdAt | Date | Дата створення документу | – | Ні | Дата створення документу |
| updatedAt | Date | Дата останнього оновлення документу | – | Ні | Дата створення документу |
| isDeleted | Boolean | Показує чи документ був видалений | – | Ні | False |

Щоб зменшити кількість коду та не допустити його дублювання, було прийнято рішення створити базову схему на основі класу, яка буде описувати спільні поля в базі даних та яку будуть наслідувати всі схеми в додатку. Програмна реалізація наведена нижче:

```
@ObjectType()
@Schema()
export class BaseSchema {
  @Field(() => String)
  _id: Types.ObjectId;

  @Field(() => Date)
  @Prop({ type: Date, default: new Date(), required: true })
  createdAt: Date;

  @Field(() => Date)
  @Prop({ type: Date, default: new Date(), required: true })
  updatedAt: Date;

  @Field(() => Boolean)
  @Prop({ type: Boolean, required: true, default: false })
  isDeleted?: boolean;
}
```

Основною одиницею майже кожної системи є користувач. Це також правдиво і для системи онлайн тестування, оскільки саме користувачі будуть

створювати та проходити тести. Користувач в системі онлайн тестування буде наслідувати базову схему та матимуть поля, які наведені у таблиці 2.2.

Таблиця 2.2 – Опис схеми користувача

| Назва поля | Тип | Опис | Обмеження | Значення NULL | Значення за замовчуванням |
|---|---|---|---|---|---|
| firstName | String | Ім’я користувача | Максимальна довжина 32 символа | Ні | – |
| lastName | String | Фамілія користувача | Максимальна довжина 32 символа | Ні | – |
| email | String | Електрона пошта користувача | Унікальне значення. Нижній регістр. Максимальна довжина 32 символа | Ні | – |
| password | String | Зашифрований пароль користувача | Мінімальна довжина 8 символів | Ні | – |

Нижче наведена програмна реалізація схеми користувача

```
@ObjectType()
@Schema()
export class User extends BaseSchema {
  @Field(() => String)
  @Prop({ type: String, required: true, maxlength: 32 })
  firstName: string;

  @Field(() => String)
  @Prop({ type: String, required: true, maxlength: 32 })
  lastName: string;

  @Field(() => String)
  @Prop({
    type: String,
    required: true,
    unique: true,
    lowercase: true,
    maxlength: 32,
  })
  email: string;
```

```
  @Prop({ type: String, required: true, minlength: 8 })
  password: string;
}
```

В системі онлайн тестування також важливою складовою є тести. Опис схеми тестів наведено у таблиці 2.3

Таблиця 2.3 – Опис схеми тестів

| Назва поля | Тип | Опис | Обмеження | Значення NULL | Значення за замовчуванням |
|---|---|---|---|---|---|
| creatorUserId | ObjectId | Унікальний ідентифікат автора тесту | – | Ні | – |
| title | String | Назва тесту | Максимальна довжина 32 символа | Ні | – |
| desctiption | String | Опис тесту | – | Так | – |
| accessType | Enum | Тип доступу до тесту | Можливі варіанти: PUBLIC, PRIVATE, INVITE | Ні | PRIVATE |
| inviteOnlyEmails | [String] | Масив, що містить пошти запрошених користувачів для типу доступу INVITE | – | Ні | – |
| questions | [Object] | Масив об’єктів з запитаннями | – | Ні | – |

Нижче наведено програмну реалізацію схеми тесту.

```
export enum TestAccessTypes {
  PUBLIC = 'PUBLIC',
  PRIVATE = 'PRIVATE',
  INVITE_ONLY = 'INVITE_ONLY',
}

@ObjectType()
@Schema()
export class Test extends BaseSchema {
  @Field(() => String)
  @Prop({ type: Types.ObjectId, required: true, ref: User.name })
  creatorUserId: Types.ObjectId;

  @Field(() => String)
  @Prop({ type: String, required: true, maxlength: 32 })
  title: string;

  @Field(() => String, { nullable: true })
  @Prop({ type: String })
  description?: string;

  @Field(() => String)
  @Prop({
    type: String,
    required: true,
    enum: TestAccessTypes,
    default: TestAccessTypes.PRIVATE,
  })
  accessType: TestAccessTypes;

  @Field(() => [String], { nullable: true })
  @Prop({ type: [String], required: false })
  inviteOnlyEmails: string[];

  @Field(() => [Question])
  @Prop({ type: [Question], required: true })
  questions: Question[];
}
```

Кожен тест містить свої запитання, які розміщені безпосередньо в схемі тесту, у вигляді масиву об’єктів. Незважаючи на те, що запитання зберігаються у вигляді масиву об’єктів – їх теж потрібно описувати на рівні з іншими схемами. Опис схеми запитань наведено у таблиці 2.4

Таблиця 2.4 – Опис схеми запитання

| Назва поля | Тип | Опис | Обмеження | Значення NULL | Значення за замовчуванням |
|---|---|---|---|---|---|
| title | String | Запитання | – | Ні | – |
| type | Enum | Тип запитання | Можливі варіанти: SINGLE MULTIPLE TEXT_INPUT CONNECT | Ні | – |
| winPoints | Number | Кількість балів за вірну відповідь | – | Ні | 1 |
| options | [Mixin] | Масив з варіантами відповідей | – | Ні | – |

Нижче наведено програмну реалізацію схеми запитання.

```
export enum QuestionTypes {
  SINGLE = 'SINGLE',
  MULTIPLE = 'MULTIPLE',
  TEXT_INPUT = 'TEXT_INPUT',
  CONNECT_OPTIONS = 'CONNECT_OPTIONS',
}

@ObjectType()
@Schema({ _id: false })
export class Question extends BaseSchema {
  @Field(() => String)
  @Prop({ type: String, required: true })
  title: string;

  @Field(() => String)
  @Prop({ type: String, enum: QuestionTypes, required: true })
  type: QuestionTypes;

  @Field(() => Number)
  @Prop({ type: Number, required: true, default: 1 })
  winPoints: number;

  @Field(() => [OptionsUnion])
  @Prop({ type: [Mongoose.Types.Mixin], required: true })
  options: (typeof OptionsUnion)[];
}
```

Результати проходження тесту будуть зберігатися в окремій колекції в базі даних. Кожен користувач зможе побачити отриману кількість балів після проходження тесту. Опис схеми результатів тестування наведено у таблиці 2.5

Таблиця 2.5 – Опис схеми результатів тестування

| Назва поля | Тип | Опис | Обмеження | Значення NULL | Значення за замовчуванням |
|---|---|---|---|---|---|
| userId | ObjectId | Унікальний ідентифікатор користувача який пройшов тест | – | Ні | – |
| testId | ObjectId | Унікальний ідентифікатор тесту | – | Ні | – |
| maxPoints | Number | Максимальна кількість балів за тест | – | Ні | – |
| currentPoints | Number | Кількість балів набраних за тест | – | Ні | – |

Нижче наведено програмну реалізацію схеми результатів тесту

```
@ObjectType()
@Schema()
export class TestResult {
  @Field(() => Types.ObjectId)
  @Prop({ type: Types.ObjectId, ref: User.name, required: true })
  userId: Types.ObjectId;

  @Field(() => Types.ObjectId)
  @Prop({ type: Types.ObjectId, ref: Test.name, required: true })
  testId: Types.ObjectId;

  @Field(() => Number)
  @Prop({ type: Number, required: true })
  maxPoints: number;

  @Field(() => Number)
  @Prop({ type: Number, required: true })
  currentPoints: number;
}
```

Завдяки ORM Mongoose достатньо лише описати схеми у вигляді класів, після чого, при спробі додати новий документ(наприклад, створити новий тест), він буде валідуватися відповідно до описаної схеми.

За допомогою ORM Mongoose, достатньо описати потрібні схеми у вигляді класів, що дозволяє легко організовувати та повторно використовувати код. Ви можете створити базові класи зі спільними полями та функціональністю, а потім успадкувати їх для створення специфічних схем. Це спрощує розробку та підтримку, а також дозволяє швидко створювати нові схеми з використанням існуючих класів.

## 2.4 Розробка бізнес моделі

Розробка бізнес-моделі для системи онлайн тестування є важливим етапом проектування, який дозволяє визначити основні аспекти функціонування та структуру системи. Успішно розроблена бізнес-моделі забезпечує високу якість та ефективність системи тестування, задовольняючи потреби користувачів та бізнес-вимоги.

У процесі розробки бізнес-моделі для системи онлайн тестування, одним із корисних інструментів є Уніфікована мова моделювання (UML) [12]. UML дозволяє визначити основні елементи системи, їх взаємозв'язки та поведінку, що сприяє кращому розумінню всіх елементів проекту

Застосування UML у розробці бізнес-моделі системи онлайн тестування дозволяє:

1. Визначити структуру системи: Використання діаграм класів UML дозволяє ідентифікувати основні сутності (наприклад, користувачі, тести, питання) та їх взаємозв'язки. Це допомагає створити концептуальну модель системи та зрозуміти, які об'єкти взаємодіють між собою.

2. Описати поведінку системи: Діаграми використання UML дозволяють відображати послідовність дій, які відбуваються в системі під час проведення тестування. Це допомагає зрозуміти логіку проходження тестів, обробку результатів та взаємодію з користувачами.

3. Покращити комунікацію: UML надає стандартизований мовний засіб, що дозволяє ефективніше спілкуватись між розробниками, дизайнерами та зацікавленими сторонами проекту. Використання діаграм UML дозволяє уникнути непорозумінь та уточнити деталі проекту.

Діаграма використання (Use Case diagram) є одним з видів діаграм, які використовуються для моделювання взаємодій між системою та її зовнішніми акторами. Вона дозволяє зображати функціональність системи шляхом представлення основних випадків використання або сценаріїв, в яких актори взаємодіють з системою для досягнення певних цілей. Діаграма використання для системи онлайн тестування зображена на рисунку 2.1.


Рисунок 2.1 – Діаграма використання

Діаграма класів є одним із типів діаграм, який використовується в об'єктно-орієнтованому програмуванні для візуалізації структури класів і взаємозв'язків між ними в системі або програмі. Вона надає графічне представлення класів, їх атрибутів, методів та зв'язків між ними. Діаграма класів для системи онлайн тестування наведена на рисунку 2.2.

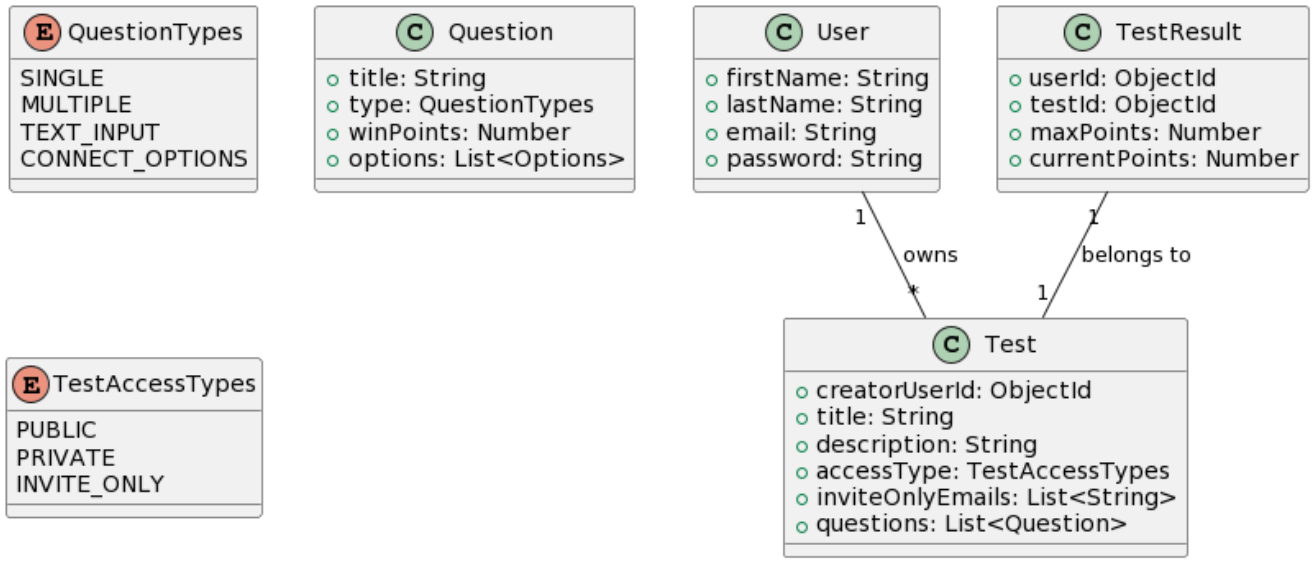



Рисунок 2.2 – Діаграма класів

Діаграма послідовності (Sequence Diagram) є одним із типів діаграм в рамках мови моделювання UML. Вона використовується для візуалізації послідовності взаємодії між об'єктами або компонентами системи в певному сценарії або процесі. Діаграма послідовності для реєстрації користувача в системі онлайн тестування наведено на рисунку 2.3.

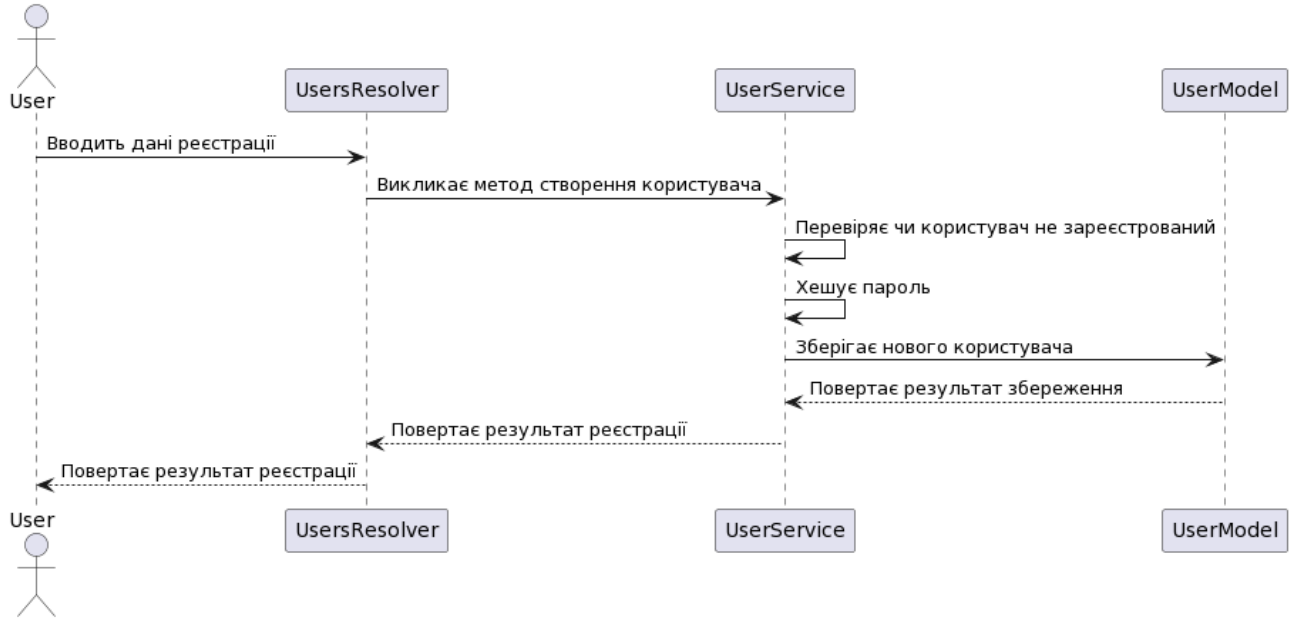



Рисунок 2.3 – Діаграма послідовності для реєстрації користувача

На рисунку 2.4 наведено діаграму послідовності для перевірки та збереження результатів тесту.

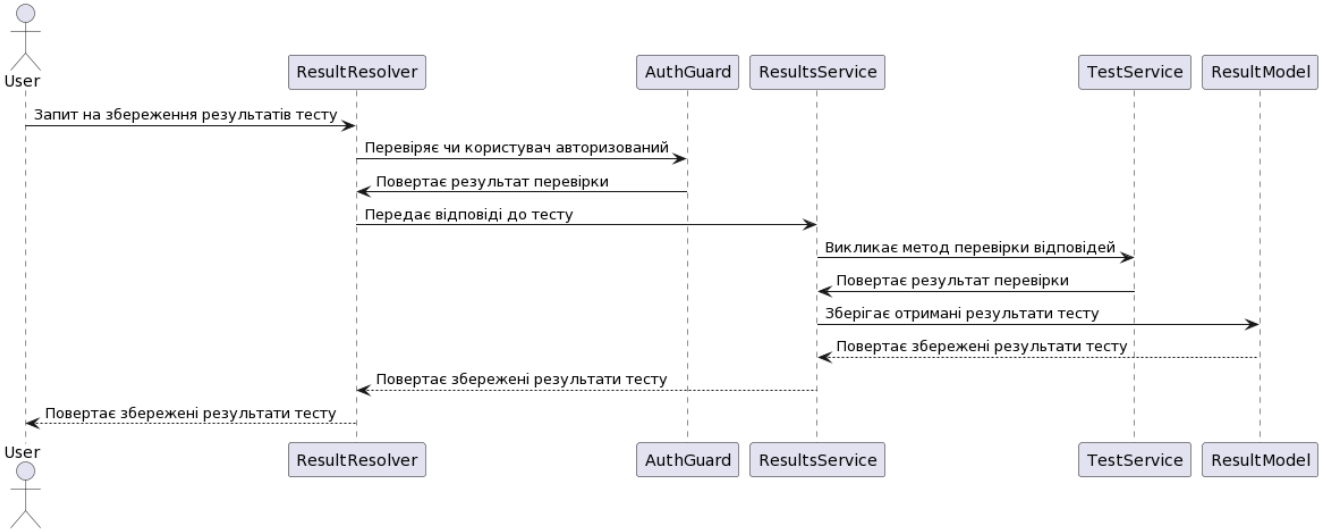



Рисунок 2.4 – Діаграма послідовності для перевірки та збереження результатів тесту

# 3 РОЗРОБКА ТА ТЕСТУВАННЯ ДОДАТКУ

## 3.1 Проектування інтерфейсу користувача

Вперше потрапивши на платформу онлайн тестування користувач буде направлений на сторінку авторизації, де йому буде запропоновано ввести свої дані щоб увійти в систему. Вигляд форми авторизації зображено на рисунку 3.1.

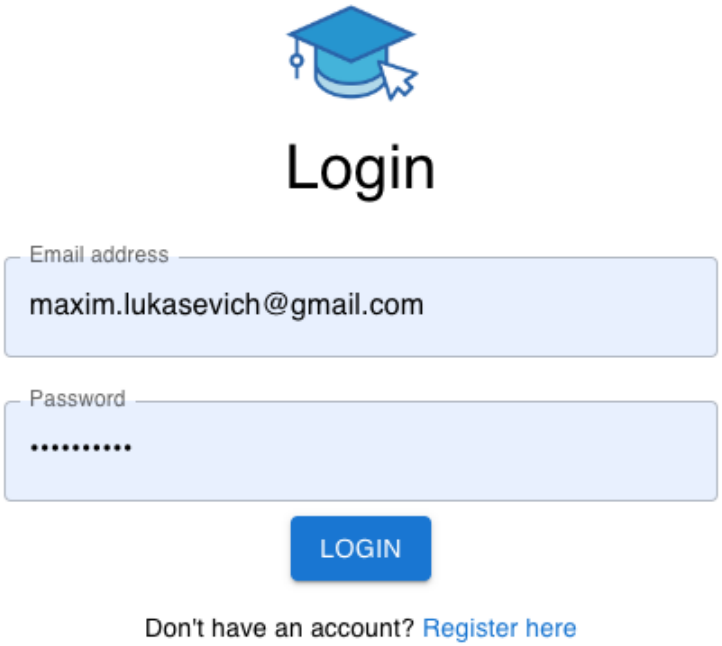



Рисунок 3.1 – Вигляд форми авторизації

Для користувачів які ще не створили власний акаунт, під кнопкою «Login» є посилання на сторінку реєстрації, клікнувши по якому користувач буде перенаправлений. Сторінка реєстрації зображена на рисунку 3.2.

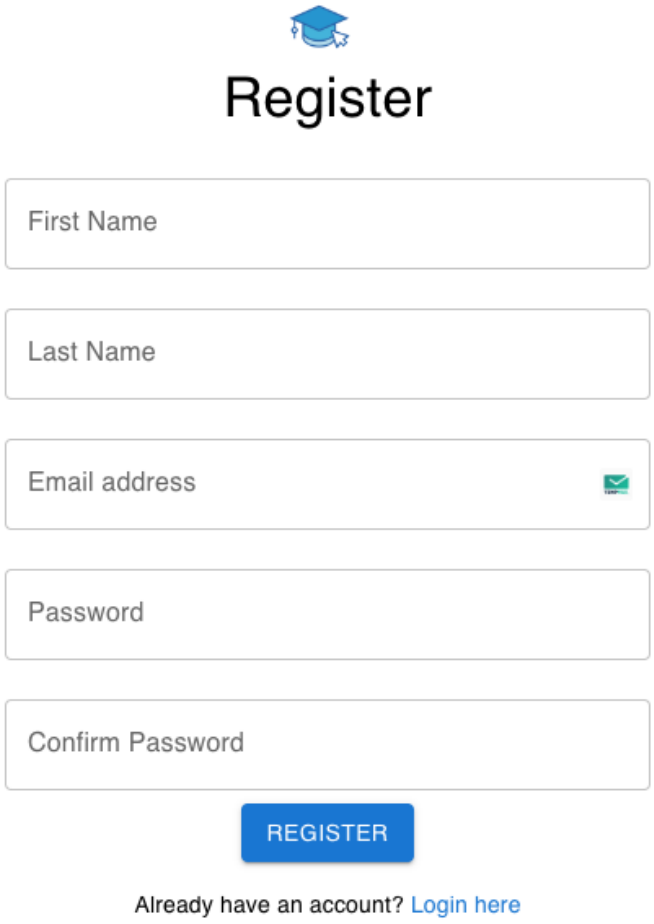



Рисунок 3.2 – Вигляд форми реєстрації

Після успішної авторизації або реєстрації, користувач перенаправляється на домашню сторінку. Зверху сторінки розташоване горизонтальне навігаційне меню на якому знаходяться наступні елементи:

1. Логотип додатку;
2. Посилання для навігації;
3. Поле для пошуку тестів;
4. Кнопка для створення тестів;
5. Ім’я та фамілія авторизованого користувача;
6. Кнопка для виходу з системи

Під навігаційним меню показуються всі тести які були створені авторизованим користувачем. Загальний вигляд головної сторінки наведено на рисунку 3.3

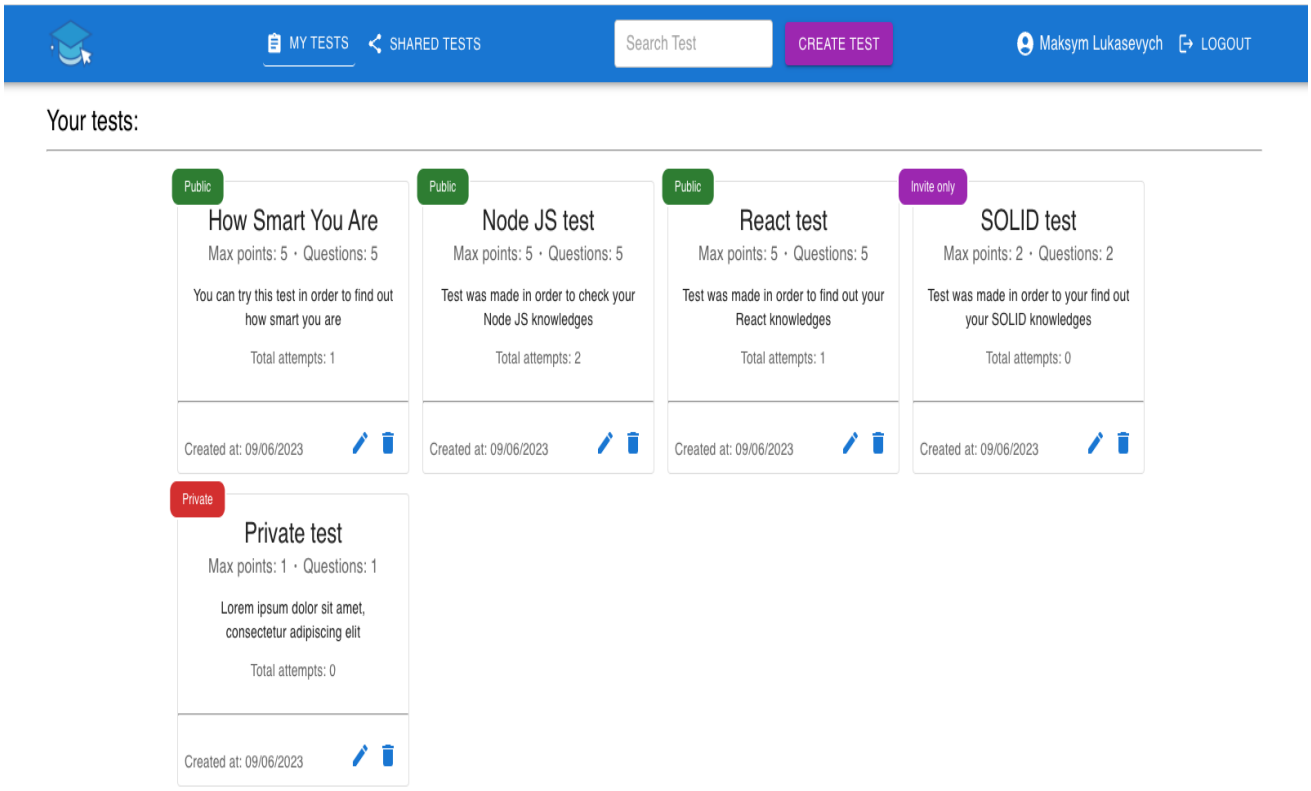



Рисунок 3.3 – Загальний вигляд головної сторінки

Натиснувши на кнопку «Create test» на панелі навігації, користувач буде направлений на сторінку створення тесту, на якій йому потрібно буде заповнити загальну інформацію про тест, а також додати потрібну йому кількість запитань. Після вводу всієї необхідної інформації та кліку по кнопці «Create test» над формою – новий тест буде додано на головну сторінку(рис. 3.3). Вигляд сторінки створення тесту наведено на рисунку 3.4.

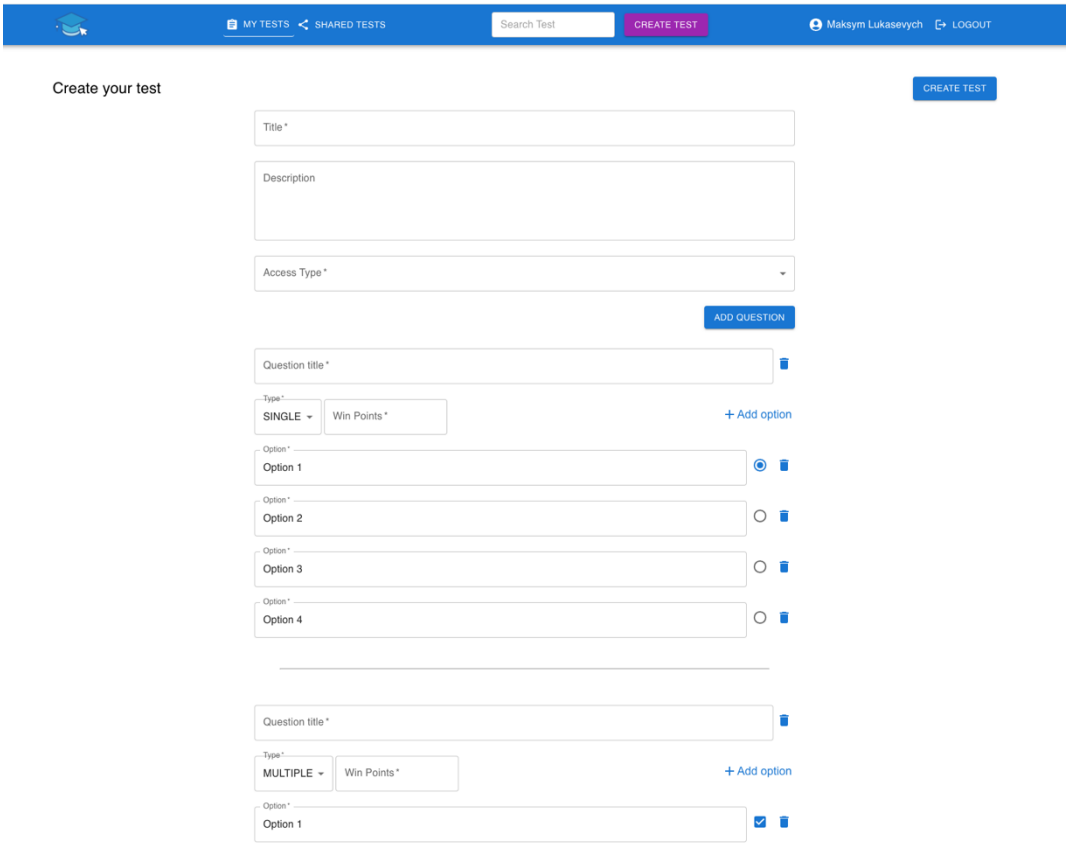



Рисунок 3.4 – Сторінка створення тесту

При створені тесту, в користувача є можливість обрати тип доступу «Invite Only». Такий тип тестів можуть бачити та проходили лише користувачі, які були запрошені до проходження цього тесту його автором. Кожен користувач може переглянути список тестів, до яких його було запрошено, натиснувши на посилання «Shared tests» на навігаційному меню. Сторінка з тестами, до яких користувач отримав запрошення, наведена на рисунку 3.5.

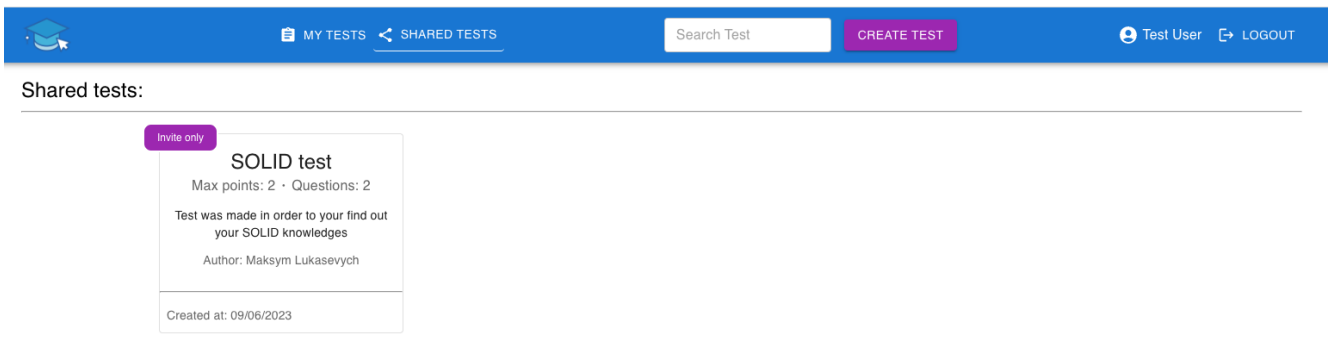



Рисунок 3.5 – Сторінка з тестами, до яких користувач отримав запрошення

Натиснувши на будь який доступний тест на сторінці, користувача буде направлено на іншу сторінку для проходження цього тесту. Сторінка для проходження тесту показана на рисунку 3.6

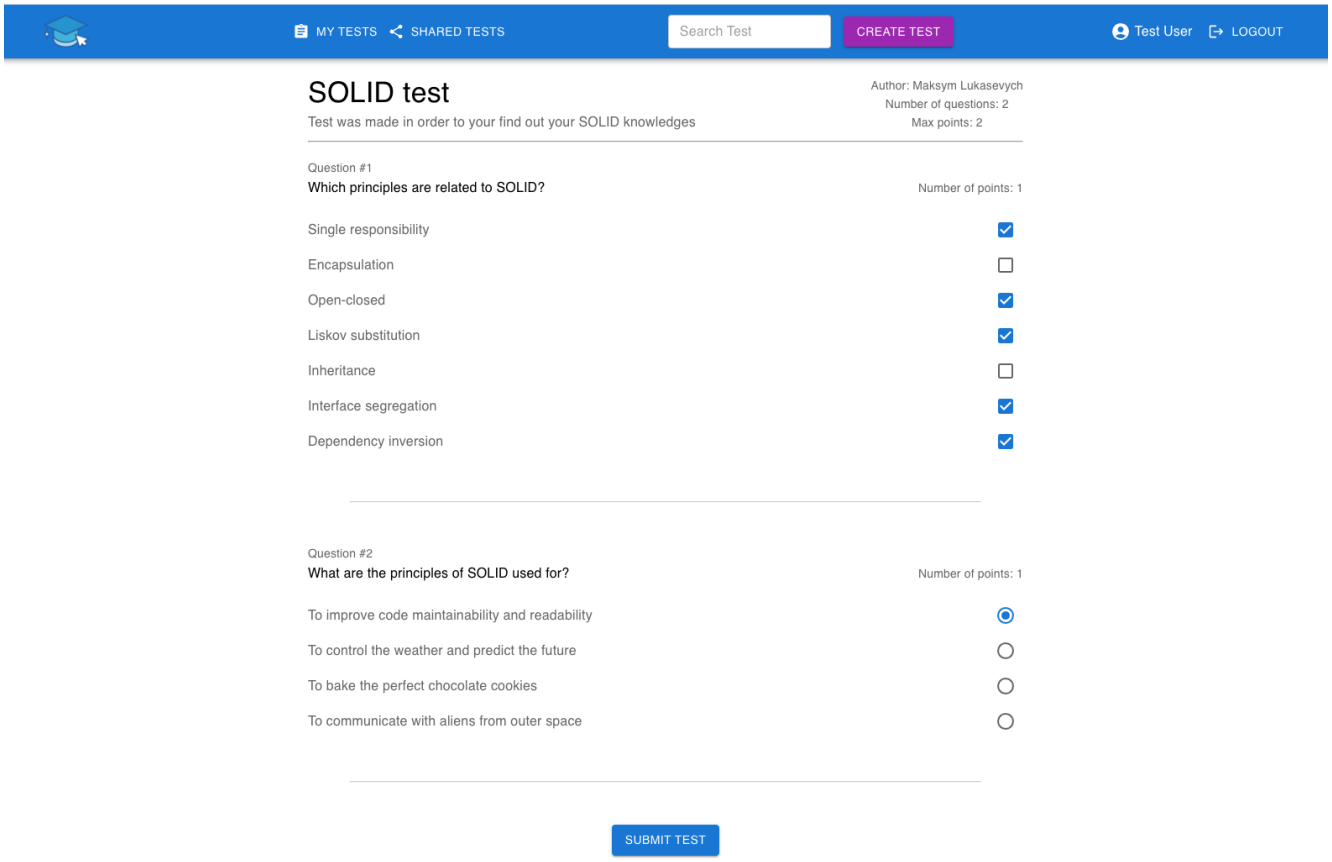



Рисунок 3.6 – Сторінка проходження тесту

Для закінчення проходження тесту, користувач має натиснути кнопку «Submit Test», після чого в нього відкриється вікно, в якому він побачить результат тестування. Вікно з результатом тестування показано на рисунку 3.7.

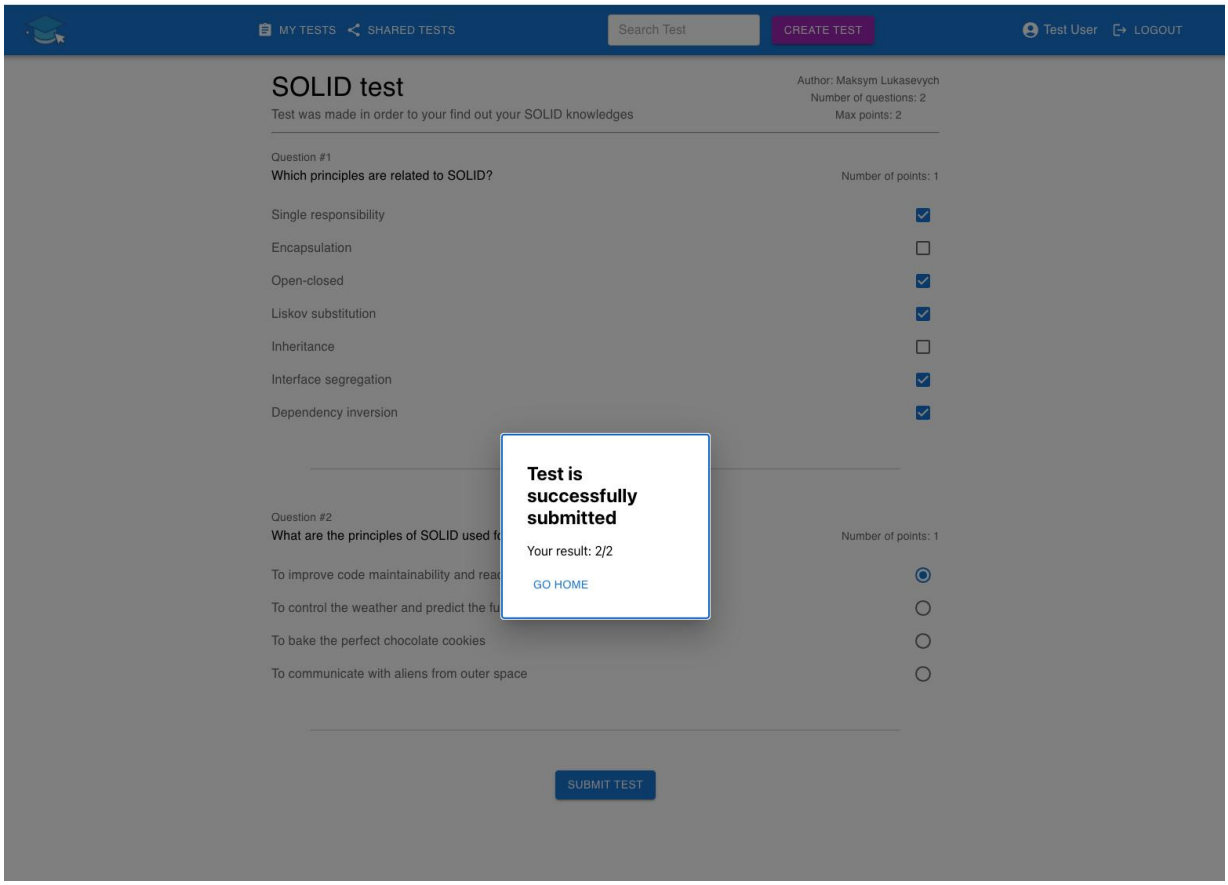



Рисунок 3.7 – Вікно з результатом тестування

## 3.2 Опис програмних модулів

Для керуванням збереженими документами в базі даних, було обрано два інструменти: MongoDB Compass [13] та Studio 3T [14]. Обидва інструменти мають свої переваги і недоліки. MongoDB Compass позиціонується як легкий в використанні інструмент. Ним зручно переглядати вміст бази даних, редагувати та видаляти одиночні документи. Studio 3T використовується для більш тісної роботи з базою даних. Він надає можливість створення резервних копії окремих колекцій, або цілої бази даних. Завдяки Studio 3T можна виконувати запити до бази даних без використання терміналу.

Серверна частина додатку побудована на модулях, які імпортуються в головний модуль під назвою AppModule. Цей модуль імпортується в файлі main.ts, в якому відбувається запуск серверної частини додатку.

Основні модулі серверної частини описані у таблиці 3.1.

Таблиця 3.1 – Опис модулів серверної частини

| Назва модулю | Назва файлу | Опис |
|---|---|---|
| AuthModule | auth.module.ts | Головний файл модулю авторизації |
| | auth.service.ts | Файл з сервісом авторизації |
| | auth.resolver.ts | Файл з описом всіх запитів та мутацій в GraphQL, які відносяться до авторизації |
| | auth.strategies.ts | Файл з стратегіями авторизації(логін-пароль, json web token) |
| | auth.guards.ts | Файл з охоронцями для авторизації, які перевіряють чи користувач авторизований |
| UserModule | user.module.ts | Головний файл модулю користувача |
| | user.service.ts | Файл з сервісом користувача |
| | user.resolver.ts | Файл з описом всіх запитів та мутацій в GraphQL, які відносяться до користувача |
| | user.schema.ts | Файл з описом схеми користувача в базі даних |
| TestModule | test.module.ts | Головний файл модулю тестів |
| | test.service.ts | Файл з сервісом тестів |
| | test.resolver.ts | Файл з описом всіх запитів та мутацій в GraphQL, які відносяться до тестів |
| | test.schema.ts | Файл з описом схеми тесту та запитань до тесту в базі даних |
| ResultModule | result.module.ts | Головний файл модулю результаті тестування |
| | result.service.ts | Файл з сервісом результатів тестування |
| | result.resolver.ts | Файл з описом всіх запитів та мутацій в GraphQL, які відносяться до результатів тестування |
| | result.schema.ts | Файл з описом схеми результатів тестування в базі даних |

Крім вище описаних модулів, в додатку також містяться модулі, які відповідають за конфігурацію підключення до бази даних, конфігурацію GraphQL серверу та конфігурацію значень оточення(.env файл) [15].

Основні модулі клієнтської частини описані у таблиці 3.2.

Таблиця 3.2 – Опис основних модулів клієнтської частини

| Назва модулю | Опис |
|---|---|
| App.tsx | Головний компонент додатку |
| AppRoutes.tsx | Компонент, який відповідає за логіку навігації між сторінками |
| Login.tsx | Компонент для авторизації користувача |
| Register.tsx | Компонент для реєстрації користувача |
| MyTests.tsx | Компонент в якому виводяться всі тести, які були створені авторизованим користувачем |
| SharedTests.tsx | Компонент в якому виводяться всі тести, до яких користувач отримав запрошення |
| Test.tsx | Компонент в якому користувач проходить обраний тест |
| ApolloClient.ts | Файл з налаштуваннями GraphQL клієнту для обміну даних з сервером |

## 3.3 Опис результатів тестування

Тестування відіграє невід’ємну роль у розробці програмного забезпечення. Воно допомагає забезпечити якість продукту шляхом виявлення помилок, дефектів та некоректної роботи функцій. Через проведення тестування можна перевірити, чи програмне забезпечення працює відповідно до вимог і очікувань користувачів. Тестування також сприяє виявленню проблем на ранніх стадіях розробки, що дозволяє зменшити кількість зусиль та часу, необхідних для виправлення помилок. Крім того, тестування допомагає забезпечити стабільність та надійність програмного забезпечення, зменшуючи ризики виникнення непередбачених проблем після випуску продукту. В загальному, тестування ділиться на 3 напрями: автоматизоване, ручне та функціональне.

Автоматизоване тестування – це процес тестування програмного забезпечення з використанням програмних засобів та без прямого втручання людини. Замість ручного виконання кожного кроку тесту, автоматизовані тестові

сценарії виконуються за допомогою програмних інструментів, які виконують певну послідовність дій та перевіряють результати.

Ручне тестування – це процес тестування програмного забезпечення, який виконується людиною або групою людей в ручному режимі. Тестувальник виконує різні дії в програмному забезпеченні, перевіряє його функціональність, реагує на взаємодію з користувачем та оцінює результати тестування. Ручне тестування дозволяє тестувальникам виявляти неочікувану поведінку, помилки та проблеми, які можуть бути пропущені автоматизованими тестами. Воно також дозволяє більш гнучко адаптуватися до змін у вимогах та швидко реагувати на нові сценарії. Проте, ручне тестування може вимагати багато часу і ресурсів, а також залежить від вмінь та досвіду тестувальника.

Функціональне тестування є одним з видів тестування програмного забезпечення, спрямованого на перевірку функціональності системи. Його основна мета полягає в тому, щоб переконатися, що програмне забезпечення працює так, як очікується відповідно до вимог та специфікацій.

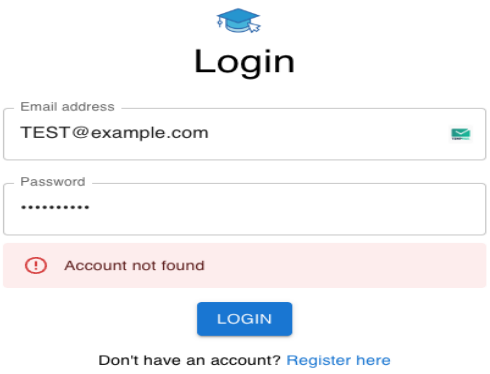
Для тестування додатку розробленого в межах дипломного проекту було обрано ручне та функціональне тестування.

В ході ручного тестування реєстрації та авторизації було виявлено помилку та створено баг-репорт [17]. Структуру баг-репорту показано у таблиці 3.3.

Таблиця 3.3 – Структура баг-репорту

| Система онлайн тестування | |
|---|---|
| 1 | 2 |
| Заголовок | Помилка авторизації. Поле електронної пошти користувача чутливе до регістру |
| Кроки для відтворення | 1. Відкрити сторінку реєстрації користувача<br>2. Зареєструвати нового користувача електронною поштою в нижньому регістрі (test@example.com)<br>3. Перейти на сторінку авторизації<br>4. Спробувати авторизуватися з створеною поштою в верхньому регістрі (TEST@example.com) |
| Очікувана поведінка | Електрона пошта користувача не є чутлива до регістру символів. Система має дозволити авторизуватися |

Продовження таблиці 3.3

| 1 | 2 |
|---|---|
| Фактична поведінка | Електрона пошта користувача є чутлива до регістру. Система не дозволяє авторизуватися |
| Серйозність дефекту | S2 Major (основний) |
| Пріоритет дефекту | P2 Hight (високий) |
| Автор | Лукасевич Максим |
| Призначений для | Лукасевич Максим |
| Опис оточення | MacOS 12.5 Monterey / Google Chrome, Version 113.0.5672.126 |
| Прикріплені файли |  |

Наступний крок після ручного тестування – це функціональне тестування. Короткий опис тест-кейсів [18] функціонального тестування наведено у таблиці 3.4.

Таблиця 3.4 – Короткий опис тест-кейсів функціонального тестування

| № | Тип | Опис | Очікуваний результат | Фактичний результат | Pass/Fail |
|---|---|---|---|---|---|
| 1 | Positive | Реєстрація користувача | Після заповнення форми користувач успішно реєструється | Після заповнення форми користувач успішно зареєструвався | Pass |
| 2 | Positive | Авторизація користувача | Користувач авторизовується після вводу коректної електронної | Користувач успішно авторизувався після вводу коректних даних | Pass |

| | | | | | |
|---|---|---|---|---|---|
| | | | пошти та паролю | | |

Продовження таблиці 3.4

| № | Тип | Опис | Очікуваний результат | Фактичний результат | Pass/Fail |
|---|---|---|---|---|---|
| 3 | Positive | Вихід з системи | Користувач вийде з системи після кліку по кнопці «Log out» на панелі навігації | Користувач успішно розавторизований після кліку по кнопці | Pass |
| 4 | Negative | Авторизація з невірним паролем | Користувач побачить помилку «Wrong password» коли спробує авторизуватися з непраивльним паролем | Користувач отримує помилку «Wrong password» | Pass |
| 5 | Negative | Помилка якщо користувач не заповнив обовязкові дані для тесту | Користувач отримає помилку, якщо під час створення/редагування тесту він не створить хоча б одне запитання | Користувач не отримав помилку. Після кліку по кнопці «Create test» нічого не відбувається | Fail |

Розширений опис тест-кейсу №5 наведено у таблиці 3.5.

Таблиця 3.5 – Розширений опис тест кейсу №5 з таблиці 3.4

| Назва тесту | Некоректне введення даних тесту | | |
|---|---|---|---|
| Функція | Створення/редагування тесту | | |
| Дата тесту | 27.05.2023 | | |
| Дія | Очікуваний результат | Фактичний результат | Pass/Fail |
| Перейти на сторінку створення нового тесту | Відкриється сторінка створення нового тесту | Відкрилася сторінка створення тесту | Pass |
| Заповнити доступні поля форми, не додаючи нове | Всі поля форми заповняться | Всі поля форми заповнені | Pass |

| | | | |
|---|---|---|---|
| запитання | | | |

Продовження таблиці 3.5

| Дія | Очікуваний результат | Фактичний результат | Pass/Fail |
|---|---|---|---|
| Натиснути кнопку «Create test» | На екрані зявиться помилка яка буде вказувати, що треба додати як мінімум одне запитання | Помилка на екрані не зявилася. Після кліку по кнопці нічого не відбулося | Fail |

Після проведення тестування всі знайдені помилки були виправлені.

Виправлення помилки, пов’язаної з чутливістю поля електронної пошти до регістру символів, відбувалося в два етапи. Перший етап – редагування схеми користувача в коді. В схемі, для поля email було додано додаткову властивість під назвою lowercase. Програмну реалізацію наведено нижче.

```
@Field(() => String)
@Prop({
  type: String,
  required: true,
  unique: true,
  lowercase: true,
  maxlength: 32,
})
email: string;
```

Оскільки ці зміни будуть застосовуватися лише до нових документів у колекції користувачів, наступним етапом було написання та запуск міграції, яка перетворить електронні пошти всіх існуючих користувачів в базі даних у нижній регістр. Для написання та запуску міграції використовувався інструмент Studio 3T. Нижче наведено код міграції.

```
db.getCollection("users").find({}).forEach(function(doc) {
  var lowercaseEmail = doc.email.toLowerCase();
  db.getCollection("users").updateOne({ _id: doc._id }, { $set: {
email: lowercaseEmail } });
})
```

# 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

## 4.1 Ергономічні проблеми безпеки життєдіяльності

Ергономічні проблеми безпеки життєдіяльності [19] відносяться до аспектів, пов’язаних з взаємодією людини з робочим оточенням та засобами, які впливають на безпеку її життя та здоров’я. Ці проблеми виникають, коли робоче середовище, обладнання або робочі процеси не враховують фізіологічні та психологічні характеристики людини.

Брак адекватного ергономічного обладнання, такого як ергономічні стільці, клавіатури, миші та підлокітники, може призвести до незручностей та негативного впливу на здоров'я офісних працівників та програмістів. Недостатня підтримка спини, неправильна позиція рук та зап'ястків можуть спричинити біль та травми. Використання ергономічного обладнання може забезпечити правильну поставу тіла та зменшити ризики виникнення подібних проблем.

Додатково, погана освітленість робочого простору, шум, вібрація та інші фізичні фактори можуть впливати на безпеку та комфортність праці. Наприклад, недостатня освітленість може призвести до напруження очей та погіршення зору, а високий рівень шуму може спричинити стрес та зниження концентрації.

Для запобігання ергономічним проблемам безпеки життєдіяльності необхідно враховувати потреби та можливості працівників під час проектування робочих місць та обладнання. Важливо забезпечити належну регулювання висоти, підтримку правильної постави тіла, а також забезпечити належну освітленість та контроль за шумом.

Загалом, ергономічні проблеми безпеки життєдіяльності впливають на здоров'я, комфорт та продуктивність працівників. Забезпечення належної ергономіки робочого середовища є важливим аспектом забезпечення безпеки та благополуччя працівників.

Робоче місце розробника програмного забезпечення в офісному середовищі повинно відповідати вимогам ергономічної безпеки та забезпечувати комфорт під час виконання роботи. Розміщення комп'ютерного обладнання, монітора,

клавіатури та миші повинно бути таким, щоб зменшити навантаження на м'язи та суглоби. Освітлення повинно бути достатнім, розподілене рівномірно та не викликати блискотіння. Робоче місце повинно мати належну підтримку для спини, регулювання висоти стільця та столу, а також додаткові ергономічні пристрої, які забезпечують комфортну поставу. Вентиляція повинна забезпечувати свіже повітря та контроль температури.

Для забезпечення безпеки та комфорту на робочому місці необхідно дотримуватись принципів ергономіки. Це означає правильне розташування робочого столу, стільця та інших робочих інструментів, що забезпечує правильну поставу тіла та підтримку м'язів. Використання ергономічного обладнання, такого як стільці з належною підтримкою, клавіатури з правильним кутом нахилу та миші з підлокітниками, також відіграє важливу роль у запобіганні травм та зайвого напруження.

Працівники та роботодавці повинні усвідомлювати важливість ергономіки в робочому середовищі та приділяти їй належну увагу. Регулярні перерви для розтяжок та фізичної активності також можуть допомогти знизити напруження та покращити кровообіг. Крім того, освітлення робочого простору має бути оптимальним, щоб запобігти зоровому напруженню та втомі.

Загальний підхід до ергономічних проблем безпеки життєдіяльності полягає в поєднанні наукових знань, розробки ефективного обладнання та свідомого використання робочого простору. Тільки забезпечивши ергономічні умови працівникам, можна знизити ризик виникнення травм, покращити продуктивність та добробут у робочому середовищі.

## 4.2 Загальні вимоги безпеки з охорони праці для користувачів ПК

Загальні вимоги безпеки з охорони праці для користувачів персональних комп'ютерів (ПК) встановлюються відповідно до законодавства України.

Закон України "Про охорону праці" [20] є основним нормативно-правовим актом, який регулює відносини в галузі охорони праці на території України. Його прийнято з метою забезпечення безпеки та здоров'я працівників, запобігання професійним захворюванням та нещасним випадкам на виробництві.

Закон передбачає визначення прав та обов'язків роботодавців та працівників. Роботодавці зобов'язані забезпечити безпечні та здорові умови праці, включаючи надання необхідного спеціального одягу та засобів індивідуального захисту. Працівники, у свою чергу, зобов'язані дотримуватися правил та вимог безпеки праці.

Основні принципи та норми охорони праці встановлені законом. Він передбачає створення системи управління охороною праці, проведення обов'язкових атестацій робочих місць та регулярних перевірок їх відповідності нормам безпеки.

Закон "Про охорону праці" також встановлює порядок реєстрації та обліку професійних захворювань та нещасних випадків на виробництві. Він передбачає обов'язкове страхування від нещасних випадків на виробництві та виплату компенсацій постраждалим працівникам.

Контроль за дотриманням вимог охорони праці та відповідальність за їх порушення також регулюються законом. Передбачаються проведення перевірок, адміністративна та кримінальна відповідальність для осіб, які порушують правила безпеки праці.

Закон України "Про охорону праці" є важливим інструментом для забезпечення безпеки та здоров'я працівників в Україні. Забезпечення безпеки праці є невід'ємною складовою умов праці з ПК, що має на меті запобігання виникненню небезпек, пов'язаних з роботою за комп'ютером.

Згідно зі статтею 11 Закону України "Про охорону праці" організація робочого місця користувача ПК повинна забезпечувати безпечні та здорові умови праці. Робоче місце має бути організоване таким чином, щоб забезпечувати безпеку працівника під час виконання роботи з ПК.

Однією з основних вимог є ергономічне облаштування робочого місця. Згідно зі статтею 12 Закону України "Про охорону праці", розміщення монітора ПК повинно відповідати вимогам ергономіки. Монітор повинен бути розташований на відстані 50-70 см від очей користувача, належно підсвічений і забезпечувати відсутність відблисків.

Додатково, згідно зі статтею 13 Закону України "Про охорону праці", клавіатура та миша повинні бути розміщені таким чином, щоб забезпечити зручність та комфорт користувача під час роботи з ПК. Клавіатура повинна бути розташована перед користувачем, належним чином закріплена та мати плоскість, що відповідає фізіологічному положенню рук під час набору тексту. Миша повинна бути зручною для тримання та використання, не викликати зайвого напруження у руці користувача.

Поміж вимог до безпеки з охорони праці для користувачів ПК також важливо враховувати час виконання роботи за комп'ютером. Згідно зі статтею 15 Закону України "Про охорону праці", регламентується встановлення періодів регулярних перерв під час тривалої роботи з ПК. Це дозволяє зменшити навантаження на очі, спину та руки, що можуть виникати внаслідок тривалого сидіння за комп'ютером.

Також, згідно зі статтею 17 Закону України "Про охорону праці", працівникам, які працюють з ПК, необхідно надавати інструктаж з правил безпеки та гігієни праці, пов'язаних з роботою з ПК. Це допомагає підвищити свідомість користувачів ПК щодо можливих небезпек та захисту їхнього здоров'я.

Законодавство України також передбачає, що роботодавці зобов'язані забезпечити високу якість та безпеку використання ПК, включаючи регулярну перевірку технічного стану обладнання, проведення профілактичного обслуговування та вчасне виявлення можливих несправностей.

Узагалі, забезпечення безпеки з охорони праці для користувачів ПК є важливим аспектом організації праці з ПК. Дотримання встановлених законодавством вимог сприяє збереженню здоров'я та попередженню виникнення професійних захворювань, пов'язаних з роботою за комп'ютером.

# ВИСНОВКИ

В аналітичній частині дипломної роботи проведено детальний аналіз предметної області, зосередженої на системах онлайн тестування. Виявлено, що у сучасному світі спостерігається тенденція до цифровізації багатьох сфер життя, включаючи тестування. Традиційні методи тестування на паперових аркушах і ручна перевірка втрачають свою ефективність, а системи онлайн тестування надають користувачам багато переваг.

Однією з ключових технічних вимог було використання клієнт-серверної архітектури з метою розділення завдань між клієнтом та сервером. Це дозволяє забезпечити ефективну комунікацію між ними та полегшити масштабування системи. Клієнтська частина, що представлена веб-браузером, забезпечує взаємодію користувача з системою, відображення даних та надання можливості взаємодії з ними. Серверна частина обробляє запити від клієнтів, забезпечує доступ до необхідних ресурсів та надає відповіді з необхідними даними.

Для обміну даними між клієнтом та сервером обрано мову запитів GraphQL, що є потужним і гнучким інструментом для ефективного обміну даними. GraphQL дозволяє клієнтам самостійно вибирати дані, які вони хочуть отримати від сервера, і зменшує кількість зайвих запитів на сервер. Також GraphQL забезпечує типізацію запитів, що дозволяє знизити ймовірність помилок і покращити розробку та підтримку системи.

У розробці схеми бази даних було вирішено використовувати MongoDB, яка є гнучкою системою управління базами даних. MongoDB належить до категорії NoSQL баз даних та забезпечує ефективне зберігання та обробку різноманітних даних. Вона дозволяє зберігати дані в гнучкому форматі документів JSON, що спрощує роботу зі змінними структурами даних.

Отже, після аналізу предметної області, аналізу бізнес- та технічних вимог, було розроблено і протестовано додаток для проведення онлайн тестування.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Node.js. Node.js. URL: https://nodejs.org/en (дата звернення: 14.06.2023).

2. The V8 JavaScript Engine. The V8 JavaScript Engine. URL: https://nodejs.dev/en/learn/the-v8-javascript-engine/ (дата звернення: 14.06.2023).

3. Клієнт-сервер архітектура - Бібліотека BukLib.net. Головна - Бібліотека BukLib.net. URL: https://buklib.net/books/23148/ (дата звернення: 14.06.2023).

4. Клієнт (інформатика) – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Клієнт_(інформатика)#:~:text=Клієнт%20–%20апаратний%20або%20програмний%20компонент,п. (дата звернення: 14.06.2023).

5. Сервер – що це таке, як працює і навіщо потрібен. Види, функції та приклади. Termin.in.ua. URL: https://termin.in.ua/server/ (дата звернення: 14.06.2023).

6. GraphQL | A query language for your API. GraphQL | A query language for your API. URL: https://graphql.org/ (дата звернення: 14.06.2023).

7. Introduction to GraphQL | GraphQL. GraphQL | A query language for your API. URL: https://graphql.org/learn/ (дата звернення: 14.06.2023).

8. Document Database - NoSQL. MongoDB. URL: https://www.mongodb.com/document-databases (дата звернення: 14.06.2023).

9. СУБД (Системи управління базами даних) - що це, види та функції. Highload.today - медиа для разработчиков. URL: https://highload.today/uk/subd-yaki-buvayut-yak-vibrati/ (дата звернення: 14.06.2023).

10. Точна різниця між SQL та NoSQL (знайте, коли використовувати NoSQL та SQL) - Інший. Огляди, Ігри, Розваги, Червень 2023.

URL: https://uk.myservername.com/sql-vs-nosql-exact-differences (дата звернення: 14.06.2023).

11. ORM (Object Relational Mapping): автоматизація запису даних. Hardware libre. URL: https://www.hwlibre.com/uk/реляційне-відображення-об’єкта-orm/ (дата звернення: 14.06.2023).

12. Основи UML. KDE Documentation -. URL: https://docs.kde.org/trunk5/uk/umbrello/umbrello/uml-basics.html (дата звернення: 14.06.2023).

13. What is MongoDB Compass? MongoDB Compass. MongoDB: The Developer Data Platform | MongoDB. URL: https://www.mongodb.com/docs/compass (дата звернення: 14.06.2023).

14. The Professional Client, IDE and GUI for MongoDB | Studio 3T. Studio 3T. URL: https://studio3t.com/ (дата звернення: 14.06.2023).

15. Основи Node та Express - Використання .env-файлу. The freeCodeCamp Forum. URL: https://forum.freecodecamp.org/t/node-express-env/569655 (дата звернення: 14.06.2023).

16. Огляд видів тестування | Безкоштовний онлайн-курс від компанії QATestLab. Безкоштовний онлайн-курс від компанії QATestLab | Головна сторінка. URL: https://training.qatestlab.com/blog/technical-articles/review-the-types-of-testing/ (дата звернення: 14.06.2023).

17. Як скласти якісний баг-репорт? - Clovertech. Clovertech. URL: https://clover.tech/jak-sklasti-jakisnij-bag-report/#:~:text=Баг-репорт%20–%20це%20звіт,,1. (дата звернення: 14.06.2023).

18. Правила оформлення теми тест-кейса | Безкоштовний онлайн-курс від компанії QATestLab. Безкоштовний онлайн-курс від компанії QATestLab | Головна сторінка. URL: https://training.qatestlab.com/blog/course-materials/test-case-topic/ (дата звернення: 14.06.2023).

19. Ергономічні вимоги як елемент покращання життєдіяльності людини. Освіта та самоосвіта. URL: https://referatss.com.ua/work/ergonomichni-vimogi-jak-element-pokrashhannja-zhittiedijalnosti-ljudini/ (дата звернення: 14.06.2023).

20. Про охорону праці. Офіційний вебпортал парламенту України. URL: https://zakon.rada.gov.ua/laws/show/2694-12#Text (дата звернення: 18.06.2023).

# ДОДАТОК А

## Код стратегій авторизації серверної частини в файлі auth.strategy.ts

```
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { PassportStrategy } from '@nestjs/passport';
import { ExtractJwt, Strategy as JwtStrategy } from 'passport-jwt';
import { Strategy as LocalStrategy } from 'passport-local';
import { User } from 'src/core/user/user.schema';
import { UserService } from 'src/core/user/user.service';

@Injectable()
export class AccessTokenStrategy extends
PassportStrategy(JwtStrategy, 'jwt') {
  constructor(private readonly userService: UserService) {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      ignoreExpiration: false,
      secretOrKey: process.env.JWT_SECRET,
    });
  }
  async validate(validationPayload: { email: string }):
Promise<User> | null {
    return this.userService.userByEmail(validationPayload.email);
  }
}

@Injectable()
export class EmailPasswordStrategy extends PassportStrategy(
  LocalStrategy,
  'local',
) {
  constructor(private readonly userService: UserService) {
    super({ usernameField: 'email', passReqToCallback: true });
  }
  async validate({
    body,
  }: {
    body: { email: string; password: string };
    req: Request & { headers: any };
  }): Promise<User> {
    const { email, password } = body;
    return this.userService
      .validateUserByPasswordAndEmail(email, password)
      .then((data) => data)
      .catch(() => {
        throw new UnauthorizedException();
      });
  }
}
```

# ДОДАТОК Б

## Код захисників серверної частини в файлі auth.guards.ts

```
import { ExecutionContext } from '@nestjs/common';
import { GqlExecutionContext } from '@nestjs/graphql';
import { AuthGuard } from '@nestjs/passport';

export class EmailPasswordGuard extends AuthGuard('local') {
  getRequest(context: ExecutionContext) {
    const ctx = GqlExecutionContext.create(context);
    const request = ctx.getContext();
    request.body = ctx.getArgs();

    return request;
  }
}

export class AccessTokenGuard extends AuthGuard('jwt') {
  getRequest(context: ExecutionContext): any {
    const ctx = GqlExecutionContext.create(context);
    const request = ctx.getContext();

    request.headers = request.req.headers;

    const tokenSource = request.req.query || request.headers;
    const accessToken = tokenSource['accessToken'];

    if (!request.headers.authorization && accessToken) {
      request.headers['authorization'] = `Bearer ${accessToken}`;
    }

    return request;
  }
}
```

# ДОДАТОК В

Код модулю з налаштуваннями серверної частини в файлі app-config.module.ts

```
import { ApolloDriver, ApolloDriverConfig } from '@nestjs/apollo';
import { Module } from '@nestjs/common';
import { ConfigModule } from '@nestjs/config';
import { GraphQLModule } from '@nestjs/graphql';
import { MongooseModule } from '@nestjs/mongoose';
import { join } from 'path';

@Module({
  imports: [
    ConfigModule.forRoot({ envFilePath: '.env.dev' }),
    MongooseModule.forRoot(process.env.DB_URL),
    GraphQLModule.forRoot<ApolloDriverConfig>({
      context: (context) =>
        context?.extra?.request
          ? {
              req: {
                ...context?.extra?.request,
                headers: {
                  ...context?.extra?.request?.headers,
                  ...context?.connectionParams,
                },
              },
            }
          : { req: context?.req },
      autoSchemaFile: join(process.cwd(), '..', 'schema.gql'),
      driver: ApolloDriver,
      sortSchema: true,
      playground: true,
    }),
  ],
})
export class AppConfigModule {}
```

ДОДАТОК Г. Диск з роботою