ЗМІСТ

див. файл змісту

ПЕРЕЛІК СКОРОЧЕНЬ ТА ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

|  |  |  |
| --- | --- | --- |
| ТЗ | － | технічне завдання |
| БД | － | база даних |
| Front end | － | клієнтська частина проекту, яка відобаржається у браузері |
| Back end | － | серверна частина проекту, яка взаємодіє із даним БД та чат ботом |
| Паттерн | － | набір правил та ресурсів для структуризації розробки |
| Ментор | － | автентифікований користувач |
| UML | － | Unified Modeling Language |
| HTTP | － | Hyper Text Transfer Protocol |
| iOS | － | iPhone operating system |
| API | － | Application Programming Interface |

|  |
| --- |
|  |

**ВСТУП**

Насьогодні існує багато варіантів розуміння терміну «Тестування». Зокрема це стосується різних сфер життя: тестування якості виробів, тестування роботи програмного забезпечення та інші види тестування. Зокрема тестування дуже широко впроваджено в навчальні процеси у світі і є їх невідємною частиною. Ідея створити власний програмний продукт для перевірки знань і навчання під час проведення тестів виникла для розширення та закріплення отриманих умінь, навичок та знань в процесі навчання на кафедрі програмного забезпечення.

Проект швидкого тестування “Quick Test” - це система, яка складається із офіційного сайту, панелі ментора, IOS додатку та Telegram бота. Панель ментора і IOS додаток містять подібний функціонал, із різницею у доступності.

Система — це щось ціле, створене з окремих частин і елементів для цілеспрямованої діяльності. Ознаки системи характеризуються:

* безліччю елементів;
* єдністю головної мети для всіх елементів;
* наявністю зв'язків між елементами;
* цілісністю і єдністю елементів;
* структурою та ієрархічністю, відносною самостійністю;
* чітко вираженим управлінням. [1]

Поняття “система” відрізняється від інших своїм фонетичним наповненням та смисловим навантаженням. “WEB-сайт” чи “програмний модуль” теж можна назвати системою, тому що вони складаються із частин, які направленні на виконання певної роботи і наповнення сенсом існування. Оскільки, якщо ми із WEB-сайту заберемо наприклад файл, де описані каскадні таблиці стилів, то цілісність лишиться, але робота не буде виконана, так як без стилізації буде порушено закон і робота відповідно не буде виконана.

До основних загальносистемних властивостей відносяться цілісність, ієрархічність, емерджентність і функціональність. [2]. Цілісність системи полягає у тому, що одна підсистеми матиме вплив на користувача, урізуючи тим самим його можливості. Кожна підсистема проекту побудована ієрархічно, де програмні модулі вищого порядку, при видаленні, матимуть вплив на ті, що нижчого рівня в ієрархії.

Підсистеми Quick Test включають в себе: офіційний сайт, панель “ментора” (версія для браузерів та IOS додаток), чат бот під платформи Telegram (Messanger від Facebook). Перша має на меті донесення інформації про проект і можливість входу чи реєстрації для початку чи продовження роботи на панелі ментора. На ній користувач у статусі “авторизованого користувача”(метнора), має можливість створити запитання, тест чи теку для групування тестів, запитань чи інших тек; видалити запитання, тест чи теку, останнє із усім вмістом; включити запитання у тест; згрупувати запитання у тимчасовий тест; запустити тестування по вже створеному тесту. Запуск тесту супроводжується сторінкою із статистикою відповідей опитуваних для ментора і відповідним сценарієм проходження тесту в Telegram боті (чи Messanger від Facebook).

Продукт має займати місце на ринку програмного забезпечення, оскільки через ситуацію із карантинними режимами навчальні заклади потребують додаток для швидкої перевірки знань учнів дистанційно. Окрім цього підприємства, де від успішної здачі внутрішнього тесту залежить надбавка підвищення тощо. Призначення системи на даному етапі - створення системи для полегшення перевірки знань.

1. **ПОСТАНОВКА ЗАДАЧІ**
   1. **Актуальність та аналіз предметної області**

Провівши поверхневий аналіз ринку продуктів для тестування у школах, було виявлено, що здебільшого користуються знайомим для кожного тестування на листочках, відповіді на які необхідно перевіряти в ручну. При пошуку інформації в глобальній мережі Інтернет було виявлено 4 основних потенційних конкурентів: Kahoot!, Quizizz, ClassMarker і Google Forms.

“Kahoot!” – це навчальна платформа, за допомогою якої можна проводити інтерактивні заняття та перевірку знань студентів за допомогою онлайн-тестування. Отримати доступ до неї можна через WEB-браузер або додаток Kahoot у Google Play або App Store.[3] Цільова аудиторія, попри те, що є можливість зареєструвати аккаунт для підприємства, виглядає так, ніби основною цільовою аудиторією є навчальні заклади, школи та дитсадки. Авторизація відбувається за допомогою Google або електронної пошти і паролю. Тести можна об’єднувати у колекції, але колекцію в середині колекції зберегти не має можливості. Проходження тесту відбувається у додатку або у браузері. Є багато різного функціоналу, у якому буває тяжко зрозуміти як зробити те, що користувач хоче.

Серед плюсів можу виділити:

* широка картотека запитань і тестів;
* приємний дизайн.

Мінуси включають в себе:

* проходження тесту відбувається у браузері;
* продукт орієнтований більше на шкільну аудиторію, попри те, що є вибір направленості при реєстрації;
* погана локалізація.

Під час пошуку першого аналога було знайдено статтю із порівнянням “Kahoot!” із “Quizizz”. Останнє це сервіс для створення опитувань і вікторин.[4]. Сервіс орієнтований на навчальні заклади, школи та дитсадки.[5] адаптивність сайту під мобільні пристрої бажає ліпшого, так як деякі блоки зміщені. Є можливість перемикання мови, але при перемиканні ні російську, не весь текст перекладається на цю мову. Авторизація можлива за допомогою Google, електронної пошти з паролем. Є можливість групувати тести у сутності подібні текам. Проходження тесту відбувається у браузері.  
 Серед плюсів можу виділити:

* багато функціональних можливостей для запитань;
* можливість додавати теги до запитань і тестів;
* широкий діапазон часу проходження запитання;
* велика бібліотека вже створених запитань та тестів.

Мінуси включають в себе:

* текст запитання бачить тільки викладач;
* набридлива музика, яка впливає на концентрованість і сприйняття уваги;
* орієнтація на розважальний сектор, а не навчальний чи професійний.

ClassMarker - це професійна, проста у використанні система онлайн-тестування, яка дозволяє за лічені хвилини створювати власні тести.

Незалежно від того, тестуєте ви 5 або 5000 користувачів, ClassMarker заощадить вам ці довгі, болісні години створення, розповсюдження та оцінки тестів.

Вам ніколи не потрібно буде повторно вводити запитання чи тести, а результати обчислюються миттєво і точно.

Крок 1: Створіть свої тести.

Крок 2: Призначте свої тести, використовуючи загальнодоступні або приватні параметри.

Крок 3: Проаналізуйте результати.[6]

Авторизація відбувається за допомогою пошти та паролю. При реєстрації є вибір між тим для яких цілей відбувається реєстрування: для бізнесу чи для навчання. Залежно від вибору типу користувача необхідно вводити дані про те ким працюєш і в якій країні, для працівника компанії, і назву країни та часовий пояс для викладача. Обидва типів користувачів можуть створювати тест і функціонально панелі для створення, запуску та відстеження тестів нічим не відрізняються. Через певний час відбувається автоматичне розлогінення. Мету таких дій мені пояснили як забезпечення безпеки даних користувача, проте це виконано не ефективно в тому сенсі, що при вичерпаному часі залогінення не можливо зберегти зміни, якщо у цей користувач створював тест чи запитання. Функціонал доволі скупий, проте ефективний. Є можливість створювати запитання і тести різних варіацій. Також є можливість компанування запитань у тест або включати вже створені запитання у тест при його створенні чи редагуванні. Орієнтація на ринок для бізнесу і університетів. Можна переглядати статистику по кожному тесту. На мобільних пристроях сайт не відображається коректно. Є проблеми із семантикою.  
 Серед плюсів можу виділити:

* є широкого спектру шаблони запитань;
* можливість створювати запитання як окрему сутність і формувати із них тест.

Мінуси:

* відсутня адаптивність сайту;
* не привабливий дизайн.

Google Forms

Це програмне забезпечення, що допомагає отримувати відповіді швидко, стилізувати форму, створювати запитання різних типів та шаблонів: один із багатьох, кілька із багатьох, вказівка часу, вказівка дати, відкрита відповідь, коротка текстова відповідь, вибір варіантів із матриці відповідей по типу А1, Б3, В2; вказівка даних на прямій ліній для виявлення рівня чогось. Окрім цього є можливість перегляду детальної статистики відповідей у реальному часі, розробляти форму командою.  
 Плюси:

* широкий спектр типів запитань;
* можливість детально продив.итися статистику;
* приємний дизайн та адаптивність сайту;

Мінуси:

* проходження у браузері;
* погано продуманий UX дизайн, при вказівці правильної відповіді.

На основі зібраних даних було виявлено, що жодна із систем не використовує технологію чат-ботів для проведення тестування.

* 1. **Обґрунтування вибору технологій**

Оцінивши стек технологій, задіяних у конкурентних системах, було визначено, що будуть використані наступні технології:

* чат-бот технологія;

Чат-бот - це програмне забезпечення для штучного інтелекту (ШІ), яке може імітувати розмову (або чат) із користувачем природною мовою за допомогою додатків для обміну повідомленнями, WEB-сайтів, мобільних додатків або за допомогою телефону.[7]

* Angular CLI;`

Інтерфейс командного інтерфейсу Angular - це інструмент інтерфейсу командного рядка, який ви використовуєте для ініціалізації, розробки, побудови ескізів та обслуговування програм Angular безпосередньо з командної оболонки.[8]

* Typescript;

TypeScript - це мова з відкритим кодом, яка базується на JavaScript, одному з найбільш використовуваних у світі інструментів, додаючи визначення статичного типу.[9]

* Next.js;

Next.js надає найкращий досвід для розробників з усіма функціями, необхідними для виробництва: гібридна статична та серверна візуалізація, підтримка TypeScript, розумне групування, попереднє завантаження маршруту тощо. Конфігурація не потрібна.[10]

* PostgreSQL;

PostgreSQL - це вдосконалена реляційна база даних з відкритим кодом корпоративного класу, яка підтримує як запити SQL (реляційні), так і JSON (нереляційні).[11]

* WebSockets(WEB-сокети);

WebSocket - це технологія, яка дозволяє клієнту встановити двосторонній («повнодуплексний») зв'язок із сервером.[12]

* Sass.

Sass - це мова таблиці стилів, яка компілюється в CSS. Це дозволяє використовувати змінні, вкладені правила, комбінації, функції тощо, і все із повністю сумісним CSS-синтаксисом.[13]

Інструменти розробки:

* Figma;

Figma - це програма для інтерфейсу, яка працює в браузері.[14] Програма дозволяє розробляти дизайн командою із можливістю редагування та перегляду.

* ESlint;

ESlint - це інструмент, який перевіряє стилістику коду, відповідно до обраних правил. ESLint статистично аналізує ваш код для швидкого пошуку проблем. ESLint вбудований у більшість текстових редакторів, і ви можете запускати ESLint як частину конвеєра безперервної інтеграції.[15]

* TSlint;

TSLint - це розширюваний інструмент статичного аналізу, який перевіряє код TypeScript на читабельність, ремонтопридатність та помилки функціональності.[16] Інструмент має відкритий код і дозволяє робити те саме, що й ESlint тільки із Typescript.

* Git;

При створенні проекту на стадії реалізації з’являється необхідність у керуванні змінами у коді. Для цього було прийнято рішення, що необхідно використовувати технологію для реалізації цієї потреби. Нею є Git. Ця система дозволяє фіксувати зміни, тобто робити коміт, надсилати зміни(коміти) у репозиторій. Репозиторій - тека із програмним кодом, до якого підключений git. Він можу бути як локальним так і віддаленим, тобто залитим у хмарне середовище і бути закріпленим за одним із сервісом для віддаленого керування репозиторіями такими як: GitLab, GitHub, BitBucket тощо.[17]

* Husky;

Хаскі покращує ваші зобов'язання та багато іншого[18]. Ця бібліотека перевіряє код на помилки перед тим як робити фіксацію змін у коді або/та при надсиланні комітів на віддалений сервер.

* NPM;

NPM - це менеджер пакетів для Node.js. Він був створений у 2009 році як проект з відкритим кодом, щоб допомогти розробникам JavaScript легко ділитися упакованими модулями коду.[19]

* JWT;

JSON Web Tokens (JWT) - WEB-токени JSON - це відкритий галузевий стандарт RFC 7519 для надійного представлення претензій між двома сторонами.[20] Дозволяє декодувати, перевіряти та генерувати JWT. Він необхідний для обміну токенами між серверами системи. Токен - маркер - у різних сверах ІТ його визначення несе різний сенс. У цьому проекті воно нестиме значення токену як елементу програмування і є основним компонентом вихідного коду. Символи класифікуються як один із п’яти класів лексем, що описують їх функції (константи, ідентифікатори, оператори, зарезервовані слова та роздільники) відповідно до правил мови програмування.[21]

Після опису вимог у ТЗ інструменти розробки мають працювати спільно наступним чином: розробляється дизайн у ресурсі Figma після чого WEB-розробник має відповідно до дизайну реалізувати сторінку. Для цього він створює репозиторій у GitHub, паралельно із тим створює локально теку для front-end частини, де створює необхідну файлову систему, ініціалізує систему git, після чого створює проект Angular за допомогою команди ng new project-name. Під час генерації проекту автоматично встановлюються Sass, Typescript та генеруються стандартні ESlint, TSlint. Після розробник додає за допомогою NPM бібліотеку Husky. Після підключення всі файли перевіряється на наявність помилок у синтаксисі. Якщо є необхідність, то файли, конструкції, тощо регулюються у файлах налаштування конфігурації ESlint та TSlint. Після налаштування залежностей та правил користувач робить перший коміт і відправляє його набором команд у віддалений репозиторій. Перед тим як відправити дані, спрацьовує Husky, який запускає команду для перевірки стилістичних помилок у коді. Якщо він доступний іншим користувачам, то вони автоматично можуть відстежувати діяльність власника репозитоію чи тих, хто також виконує зміни у коді на WEB-сторінці проекту.

* 1. **Моделювання предметної області**

Проаналізувавши конкуруючі продукти на ринку, було вирішено, що система перш за все повинна складатися із модулів і включати у себе сутності для опису головних аспектів системи. Має бути єдина база даних для всієї системи. Алгоритм користування має бути наступним: користувач заходить на офіційний сайт, переходить на сторінку для реєстраці чи входу, успішно отримує актуальний токен аутентифікації. Аутентифікація на основі токенів - це протокол, який дозволяє користувачам перевіряти свою особу і взамін отримувати унікальний маркер доступу. Протягом життя маркера користувачі отримують доступ до WEB-сайту або програми, для якої маркер виданий, замість того, щоб повторно вводити облікові дані кожного разу, коли вони повертаються на ту саму WEB-сторінку, додаток або будь-який ресурс, захищений цим же маркером.[22] Такі маркери є тимчасовими і потребують оновлення через певний час, тому вони є доволі актуальними, оскільки забезпечують більш безпечну передачу даних між серверами.

Далі користувач переходить на сторінку панелі ментора, де є можливість створити тест, запитання та теку(далі файли). Перше є ключовою сутністю проекту, оскільки вся суть Quick Test - проведення швидкого тестування. Запитання було виокремлено як окрему сутність після аналізу конкурентів, у такому рішенні є доволі багато плюсів, але про них пізніше. Тека була обрана як одні із сутностей всього проекту тому, що якщо є потреба групувати файли. Така сутність була реалізована в “Kahoot!”, хоч там були колекції, тут планується реалізувати повноцінну файлову систему по типу тої, щ в операційних системах Windows чи дистрибутиві Linux.

Створення файлу має супроводжуватися формою із перевіркою на вимоги її полів. Кожна така форма повинна містити кнопку повернення на попередню сторінку або кнопку для скасування. Якщо введені дані не відповідають вимогам, то поряд із полями, де були помилки мають відображатися детальні повідомлення про помилки. Окрім цих базових правил, форми мають відповідати семантиці та сучасним UI/UX патернам.

Якщо я створення, то має бути і видалення, редагування та перегляд вмісту файлів. Це теж має бути реалізовано. Загальні вимоги до форм вже описані вище. Видалення файлів має бути реалізовано індив.ідуально, оскільки для кожної із сутностей є свої обмеження.

Після створення тесту та при перегляді його вмісту має бути можливість активувати тест. Після його активації його необхідно запустити, іншими словами відкрити доступ для проходження. Час активації закінчитися або коли всі учасники дадуть відповідь на запитання, або після завершення часу на надання відповіді останнього запитання. Поки тест є активованим, ментор переглядає динамічну статистику відповідей.

* 1. **Формування ТЗ**

Технічне завдання - це інструмент, який дозволяє привести бажання клієнта у відповідність із можливостями розробника і здатний врегулювати всі можливі конфлікти та заощадити час. Безумовно, технічне завдання визначає коло завдань, які слід виконати під час створення цього веб-сайту.

Необхідно надати розробникові структуру веб-сайту якомога детальніше і пояснити, як повинна виглядати остаточна версія. Таким чином, технічне завдання на розробку веб-сайту електронного магазину повинно містити таке:

* технічні характеристики;
* маркетингові вимоги;
* особливості дизайну.

В ідеальному випадку всі деталі роботи та зовнішнього вигляду повинні бути зазначені в технічному завданні, таким чином невідповідності будуть мінімізовані, а результат відповідатиме вимогам та очікуванням замовника.[23]

1. Так як явного замовника не має, то усі вимоги будуть відповідати тим, що були розроблені командою розробників системи. Таким чином до технічних вимог відносяться: вибір, обґрунтування та опис апаратних(hardware) та системних(software) засобів, архітектурні вимоги, опис системи, опис задіяних технологій та їх опис, вимоги до реалізації та подальших оновлень, Огляд продукту з точки зору різних типів користувачів. Маркетингові вимоги охоплюють дослідження цільової аудиторії, аналіз ринку, локалізація, ресурси для реалізації, потоки прибутків, потоки витрат, канали збуту, ризики, бізнес план та обумовлення існування продукту. Особливості дизайну ґрунтуються на описаних вище вимогах і дослідженнях. Оскільки, система на момент написання роботи не є комерційним продуктом, то не має потреби описувати бізнес план і все, що стосується Quick Test як продукту на ринку.
2. Опис технічних вимог

Система має включати у себе функціонал поділений по сутностям: запитання, тест та тека. Він має бути доступний тільки авторизованим чи автентифікованим користувачам. Інші повинні мати доступ до статичних сторінок офіційного сайту, де має міститися основна інформація про систему та інструкція користувача. Щоб змінити свій статус, користувачу необхідно увійти чи створити новий обліковий запис.

Функціонал сутностей має ґрунтуватися на шаблоні CRUD(Create Read Update Delete). CRUD Значення: CRUD - це абревіатура, що походить із світу комп’ютерного програмування і стосується чотирьох функцій, які вважаються необхідними для реалізації постійного сховища для зберігання даних: створення, читання, оновлення та видалення. Постійне сховище - це будь-який пристрій зберігання даних, який зберігає живлення після вимкнення пристрою, наприклад, жорсткий диск або твердотілий накопичувач. Навпаки, оперативна пам’ять та внутрішнє кешування є двома прикладами енергонезалежної пам’яті - вони містять дані, які будуть стерті, коли вони втратять силу.[24]

Згідно цьому скороченню, користувач, який увійшов у свій обліковий запис, повинен мати доступ до CRUD запитання, тест та теку. Окрім цього базового функціоналу, він може запустити тест для проведення тесту із подальшим переглядом актуальної статистики. Також має бути закладена основа для подальшої монетизації, при переході системи у статус комерційного стартапу, у вигляді блоків реклами, копіювання чужих запитань для використання у власних тестах, додання картинок до запитань, перегляд статистики відповідей окремо. Описані функціональні рішення доступні на панель ментора. Менторство – це спосіб передачі знань в рамках процесу навчання в компанії. Інструмент, який дозволяє розвивати людей і передавати практичний досвід "з рук в руки". У певний момент в рамках компанії з'являються потреби в розвитку, які можна кластеризувати наступним чином: бажання розвивати себе – люди, які хочуть розвиватися в своїй сфері і вивчати щось нове, розширювати знання в суміжних областях або навіть змінити професію; бажання розвивати інших – люди, які вже досягли експертності і готові своїми знаннями ділитися.[25] Відповідно ментор - людина, яка передає знання “із рук в руки” і перевіряє їх.

### Згідно із переліком пункту 1.2 першого розділу, мають бути використані технології: AngularCLI, TypeScript, Node.js, Nest.js, PostgreSQL, JWT, Alamofire, GodAnimation, Strongbox, SignalRClient; та інструменти розробки: Figma, ESlint, TSlint, Git, Husky, NPM, JWT.

### [Огляд продукту з точки зору різних типів користувачів](https://140c4d19-be2a-4690-a2ae-49bf342ef500.vscode-webview-test.com/vscode-resource/file///f:/Projects/Quick%20Test/quick-test-docs/README.md" \l "%D0%BE%D0%B3%D0%BB%D1%8F%D0%B4-%D0%BF%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82%D1%83-%D0%B7-%D1%82%D0%BE%D1%87%D0%BA%D0%B8-%D0%B7%D0%BE%D1%80%D1%83-%D1%80%D1%96%D0%B7%D0%BD%D0%B8%D1%85-%D1%82%D0%B8%D0%BF%D1%96%D0%B2-%D0%BA%D0%BE%D1%80%D0%B8%D1%81%D1%82%D1%83%D0%B2%D0%B0%D1%87%D1%96%D0%B2" \o "#%D0%BE%D0%B3%D0%BB%D1%8F%D0%B4-%D0%BF%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82%D1%83-%D0%B7-%D1%82%D0%BE%D1%87%D0%BA%D0%B8-%D0%B7%D0%BE%D1%80%D1%83-%D1%80%D1%96%D0%B7%D0%BD%D0%B8%D1%85-%D1%82%D0%B8%D0%BF%D1%96%D0%B2-%D0%BA%D0%BE%D1%80%D0%B8%D1%81%D1%82%D1%83%D0%B2%D0%B0%D1%87%D1%96%D0%B2)

Типів користувачів має бути 3: гість, неавтентифікований користувач та автентифікований. Перший та другий не мають доступу до функціоналу панелі ментора. Для того, щоб їм був доступний основний функціонал системи їм необхідно зареєструватися та увійти в обліковий запис відповідно. Тоді вони перейдуть в статус автентифікованого користувача і зможуть взаємодіяти із файлами в середині тек.

Детальніше про доступний функціонал користувачам див.ись у розділі другому підпункті №2.4.

Архітектура має бути розроблена із використанням уже розроблених шаблоні спільнотою програмістів або базуватися на них.

Для роботи із системою на ПК необхідно мати ОС, яка підтримує останню версію Google Chrome, станом на 06.06.21. Для роботи зі смартфону на платформі Android чи IOC вимоги аналогічні. Тільки для користувачів останніх для використання програми необхідно мати останню версію системи, щоб додаток працював коректно.

1. Особливості дизайну.

Дизайн має бути інтуїтивним і семантичним, відповідати шаблонам UI та UX. Елементи інтерфейсу мають повторювати форми логотипу і усі кольори мають гармонійними. Також мають бути розроблені шаблони усіх сторінок та форм під різні розміри екранів, за основу має бути взята ширина екрану - 375 пікселів, 640 пікселів, 1024 пікселів та 1440 пікселів. Допускаються пропуски на шаблонах офіційного сайту та формах авторизації та реєстрації. При реалізації дизайну сторінок допускаються розбіжності.

**Висновок до першого розділу**

Виявлено головних конкурентів: Kahoot!, Quizizz, ClassMarker і Google Forms. Головними плюсами серед них - велика варіативність шаблонів та типів запитань, а також деякі із них мають приємний інтерфейс та цікаву реалізацію проходження тестів. Серед мінусів можна узагальнити відсутність адаптивності, нав’язливе пропонування платних послуг або урізання функціоналу для схилення до оплати, нереалізована або часткова локалізація користувацького інтерфейсу.

Дослідив.ши технології, на яких реалізовані конкуренти, глибокому аналізі ринку та можливостей членів команди, було виявлено, що при написанні проектного коду необхідно використати фреймворки та мову програмування які дозволяють використовувати модульний підхід. Для більш легкого масштабування проекту в майбутньому. Окрім цього необхідна чітка типізація даних як для збереження у базу даних так і при обробці її на back-end та front-end частинах. Головними сутностями мають бути: запитання, тест та тека із можливістю створення, редагування та видалення. Мають бути використані наступні технології: Angular CLI, Typescript, Nest.js, PostgreSQL, WebSocket, Sass. При розробці мають бути задіяні наступні інструменти: Figma, ESlint, TSlint, Git, Husky, NPM, JWT.

1. **ПРОЕКТУВАННЯ СИСТЕМИ**

Проектування складається із послідовних дій: опису головних сутностей, формування на їх основі головних та вторинних програмних модулів, опис взаємодії різних типів користувачів із системою, проектування користувацького інтерфейсу, після розробка архітектури на основі модулів і вимог до інтерфейсу користувача відбувається проектування бази даних. Усе це включає у себе план реалізації проекту.

* 1. **План реалізації проекту**

Перед тим як почати процес проектування системи, було складено план реалізації, який включає у себе дані про дні на виконання, даних початку та завершення задач з переліком ресурсів, необхідних у процесі, відповідно до можливостей реалізаторів. Так як є частини, які не можливо реалізувати, якщо не має вже готового функціоналу, то вони були побудовані із включенням цієї залежності.

Таблиця 2.1

**Перелік необхідних робіт до виконання**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Назва задачі** | **Днів на виконання** | **Дата початку** | **Дата завершення** | **Задачі до** | **Ресурси** |
| **Ініціалізація** | | | | | |
| Пошук аналогів | 4 днів | Ср 1/13/21 | Пн 1/18/21 |  | Оренда офісу[1],  Трудові ресурси,Харчування |
| Встановлення вимог до пректування | 3 днів | Вт 1/19/21 | Чт 1/21/21 | 3 | Трудові ресурси,  Електроенергія[1] |
| Узагальнений спис головних сутностей | 15 днів | Пт 1/22/21 | Чт 2/11/21 | 5 | Електроенергія[1],  Оренда офісу[1]  ,Трудові ресурси,Харчування |
| **Проектування** | | | | | |
| Напитання ТЗ | 22 днів | Пт 1/22/21 | Пн 2/22/21 |  | Електроенергія[1],  Трудові ресурси,Харчування |
| Детальний опис сутностей | 14 днів | Вс 1/24/21 | Ср 2/10/21 |  | Трудові ресурси,  Електроенергія[1]  ,Харчування |
| Опис модулів проекту | 12 днів | Вс 2/7/21 | Пн 2/22/21 |  | Оренда офісу[1],  Електроенергія[1],  Харчування |
| Опис функціоналу модулів | 4 днів | Пт 1/29/21 | Ср 2/3/21 |  | Електроенергія[1],  Трудові ресурси,Харчування,  Оренда офісу[1] |
| Опис моделей проекту | 3 днів | Вс 2/7/21 | Вт 2/9/21 |  | Електроенергія[1],  Оренда офісу[1],  Харчування |

Продовження таблиці 2.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Проектування бази даних | 3 днів | Пн 2/1/21 | Ср 2/3/21 |  | Харчування,Електроенергія[1],  Оренда офісу[1],  Трудові ресурси |
| Реалізація дизайну головних сторінок | 12 днів | Чт 2/4/21 | Пт 2/19/21 | 17 | Електроенергія[1.5],  Оренда офісу[1.5],  Харчування,Трудові ресурси |
| **Реалізація** | | | | | |
| **Реєстрація** | 23 днів | Вт 3/30/21 | Чт 4/29/21 |  |  |
| backend | 3 днів | Вт 3/30/21 | Чт 4/1/21 | 65 |  |
| frontend | 5 днів | Вс 4/25/21 | Чт 4/29/21 | 26 |  |
| ios | 5 днів | Вс 4/25/21 | Чт 4/29/21 | 26 |  |
| **Вхід у систему** | 8 днів | Пн 2/15/21 | Ср 2/24/21 |  |  |
| backend | 3 днів | Пн 2/15/21 | Ср 2/17/21 |  |  |
| frontend | 5 днів | Чт 2/18/21 | Ср 2/24/21 | 30 |  |
| ios | 5 днів | Чт 2/18/21 | Ср 2/24/21 | 30 |  |
| **Test** | 52 днів | Пн 2/15/21 | Вт 4/27/21 |  |  |
| ***Create Test*** |  |  |  |  |  |
| backend | 4 днів | Пн 2/15/21 | Чт 2/18/21 |  |  |
| frontend | 10 днів | Чт 2/18/21 | Чт 3/4/21 | 39 |  |
| ios | 10 днів | Чт 2/18/21 | Чт 3/4/21 | 39 |  |
| ***Read test*** |  |  |  |  |  |
| backend | 3 днів | Пт 2/19/21 | Вт 2/23/21 | 39 |  |
| frontend | 10 днів | Пт 3/5/21 | Чт 3/18/21 | 45 |  |
| ios | 10 днів | Пт 3/5/21 | Чт 3/18/21 | 45 |  |
| ***Update test*** | 24 днів | Ср 2/24/21 | Пн 3/29/21 |  |  |
| backend | 5 днів | Ср 2/24/21 | Вт 3/2/21 | 39,45 |  |
| frontend | 7 днів | Пт 3/19/21 | Пн 3/29/21 | 49 |  |
| ios | 7 днів | Пт 3/19/21 | Пн 3/29/21 | 49 |  |
| ***Delete test*** |  |  |  |  |  |
| backend | 3 днів | Ср 3/3/21 | Пт 3/5/21 | 49 |  |
| frontend | 6 днів | Вт 3/30/21 | Вт 4/6/21 | 53 |  |
| ios | 6 днів | Вт 3/30/21 | Вт 4/6/21 | 53 |  |
| ***Start test*** |  |  |  |  |  |
| backend | 7 днів | Сб 3/6/21 | Пн 3/15/21 | 53 |  |
| frontend | 5 днів | Ср 4/7/21 | Вт 4/13/21 | 57 |  |
| ios | 5 днів | Ср 4/7/21 | Вт 4/13/21 | 57 |  |
| ***Show statistic for teacher*** |  |  |  |  |  |
| backend | 10 днів | Ср 3/17/21 | Вт 3/30/21 | 57 |  |
| frontend | 10 днів | Ср 4/14/21 | Вт 4/27/21 | 61 |  |
| ios | 10 днів | Ср 4/14/21 | Вт 4/27/21 | 61 |  |
| ***Переміщення між теками*** |  |  |  |  |  |
| backend | 2 днів | Ср 3/31/21 | Чт 4/1/21 | 61 |  |
| frontend | 7 днів | Пн 4/12/21 | Вт 4/20/21 | 65 |  |
| ios | 8 днів | Чт 4/15/21 | Пн 4/26/21 | 65 |  |
| **Question** |  |  |  |  |  |
| ***Create question*** | 25 днів | Пт 4/2/21 | Чт 5/6/21 |  |  |
| backend | 5 днів | Пт 4/2/21 | Чт 4/8/21 | 65 |  |
| frontend | 7 днів | Вс 4/25/21 | Пн 5/3/21 | 70 |  |
| ios | 7 днів | Ср 4/28/21 | Чт 5/6/21 | 70 |  |
| ***Read question*** | 23 днів | Сб 4/10/21 | Ср 5/12/21 |  |  |
| backend | 2 днів | Сб 4/10/21 | Пн 4/12/21 | 70 |  |

Продовження таблиці 2.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| frontend | 4 днів | Вт 5/4/21 | Пт 5/7/21 | 74 |  |
| ios | 4 днів | Сб 5/8/21 | Ср 5/12/21 | 74 |  |
| ***Update question*** | 32 днів | Вт 4/13/21 | Ср 5/26/21 |  |  |
| ***Перейменування*** | 29 днів | Вт 4/13/21 | Пт 5/21/21 |  |  |
| backend | 4 днів | Вт 4/13/21 | Пт 4/16/21 | 74 |  |
| frontend | 6 днів | Вс 5/9/21 | Пт 5/14/21 | 79,75 |  |
| ios | 6 днів | Пт 5/14/21 | Пт 5/21/21 | 79,76 |  |
| ***Переміщення між теками*** | 28 днів | Сб 4/17/21 | Ср 5/26/21 |  |  |
| backend | 2 днів | Сб 4/17/21 | Пн 4/19/21 | 79 |  |
| frontend | 4 днів | Сб 5/22/21 | Ср 5/26/21 | 83 |  |
| ios | 4 днів | Сб 5/22/21 | Ср 5/26/21 | 83 |  |
| ***Delete question*** | 33 днів | Вт 4/20/21 | Чт 6/3/21 |  |  |
| backend | 3 днів | Вт 4/20/21 | Чт 4/22/21 | 83 |  |
| frontend | 5 днів | Сб 5/29/21 | Чт 6/3/21 | 87 |  |
| ios | 5 днів | Сб 5/29/21 | Чт 6/3/21 | 87 |  |
| **Folder** | 84 днів | Пт 3/19/21 | Чт 7/15/21 |  |  |
| ***Create*** | 59 днів | Пт 3/19/21 | Чт 6/10/21 |  |  |
| backend | 3 днів | Пт 4/23/21 | Вт 4/27/21 | 87 |  |
| frontend | 5 днів | Сб 6/5/21 | Чт 6/10/21 | 93 |  |
| ios | 5 днів | Сб 6/5/21 | Чт 6/10/21 | 93 |  |
| ***Update*** | 74 днів | Пт 3/19/21 | Чт 7/1/21 |  |  |
| *Заміна кольору* | 64 днів | Пт 3/19/21 | Чт 6/17/21 |  |  |
| backend | 3 днів | Ср 4/28/21 | Пт 4/30/21 | 93 |  |
| frontend | 5 днів | Пт 6/11/21 | Чт 6/17/21 | 101 |  |
| ios | 5 днів | Пт 6/11/21 | Чт 6/17/21 | 101 |  |
| ***Переміщення між теками*** | 69 днів | Пт 3/19/21 | Чт 6/24/21 |  |  |
| backend | 3 днів | Пн 5/3/21 | Ср 5/5/21 | 101 |  |
| frontend | 5 днів | Пт 6/18/21 | Чт 6/24/21 | 108 |  |
| ios | 5 днів | Пт 6/18/21 | Чт 6/24/21 | 108 |  |
| ***Перейменування*** | 74 днів | Пт 3/19/21 | Чт 7/1/21 |  |  |
| backend | 3 днів | Чт 5/6/21 | Пн 5/10/21 | 108 |  |
| frontend | 5 днів | Пт 6/25/21 | Чт 7/1/21 | 115 |  |
| ios | 5 днів | Пт 6/25/21 | Чт 7/1/21 | 115 |  |
| **Read** | 79 днів | Пт 3/19/21 | Чт 7/8/21 |  |  |
| backend | 3 днів | Вт 5/11/21 | Чт 5/13/21 | 115 |  |
| frontend | 5 днів | Пт 7/2/21 | Чт 7/8/21 | 122 |  |
| ios | 5 днів | Пт 7/2/21 | Чт 7/8/21 | 122 |  |
| **Delete** | 84 днів | Пт 3/19/21 | Чт 7/15/21 |  |  |
| backend | 3 днів | Пт 5/14/21 | Вт 5/18/21 | 122 |  |
| frontend | 5 днів | Сб 7/10/21 | Чт 7/15/21 | 129 |  |
| ios | 5 днів | Сб 7/10/21 | Чт 7/15/21 | 129 |  |

Для простішого сприйняття що за чим має реалізовуватися, було розроблено схему планування задач із розподілом з приблизною кількістю годин на виконання. Час на виконання було прив’язано до дат, для прогнозування приблизного часу на реалізацію проекту.(див. таблицю 2.1)

## Виконання завдань має відбуватися відповідно до методології Agile, а саме scrum із використанням kumbam дошки для розподілу завдань.

Scrum - це фреймворк, в якій люди можуть вирішувати складні адаптаційні проблеми, одночасно продуктивно та творчо поставляючи продукти з найвищою можливою цінністю.

Scrum - це легкий фреймворк, який допомагає людям, командам та організаціям генерувати цінність завдяки адаптивним рішенням для складних проблем. Співавтори Scrum Кен Швабер та Джефф Сазерленд написали "Керівництво по Scrum", щоб чітко та коротко пояснити Scrum. Цей посібник містить визначення Scrum. Це визначення складається з підзвітності Скрума, подій, артефактів та правил, що пов’язують їх між собою.[26]

Схематично методологія scrum виглядає як а рисунку 2.1.

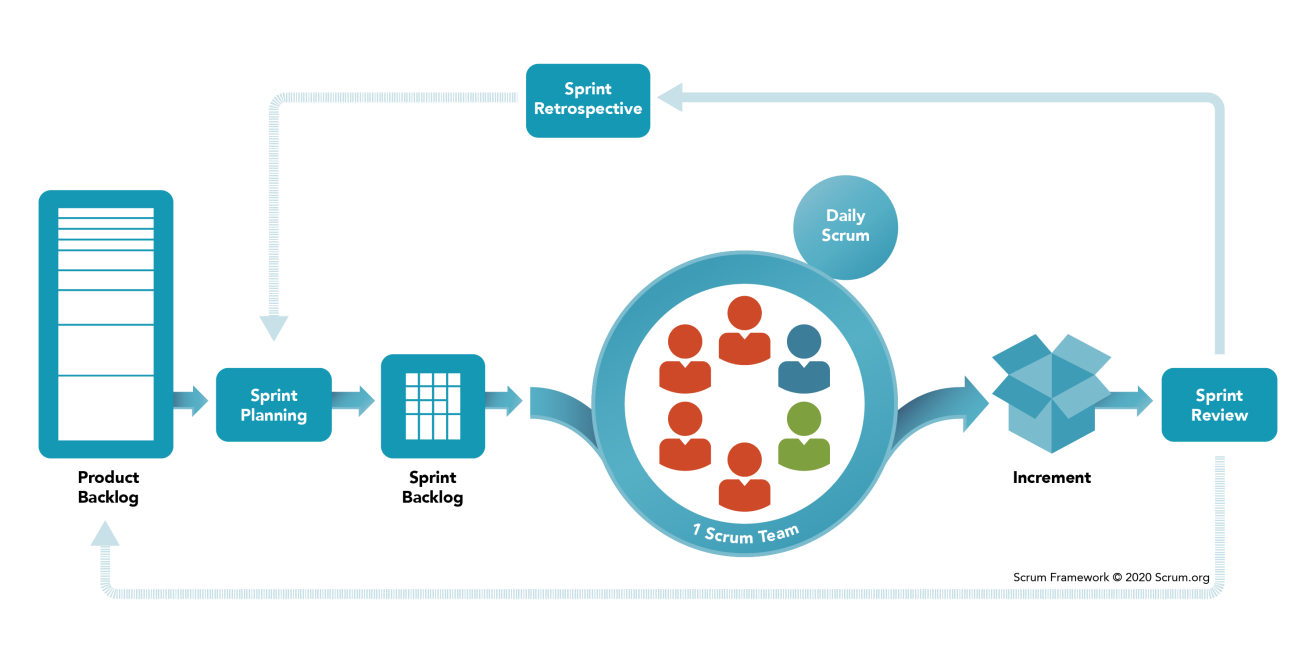


Рисунок 2.1－ Схема методології scrum[26]

* 1. **Опис головних сутностей**

Сервісів, де опитування відбувається за допомогою чат ботів не було знайдено. Проте ми провели аналітику ринку тестування людей і наближеними до нас виявилися: Kahoot!, Quizizz, ClassMarker і Google Forms.

Таблиця 2.2

**Порівняльний перелік особливостей існуючих подібних систем.**

| **Критерій** | **Kahoot!** | **Quizizz** | **ClassMarker** | **Google Forms** |
| --- | --- | --- | --- | --- |
| Цільова аудиторія | Школи/Дитсадки | Школи/Дитсадки | Школи, підприємства | Навчальні заклади, підприємства |
| Адаптивність | + | - | - | + |
| Динамічний перегляд статистики відповідей | + | + | - | - |
| Запитання бачить і опитуваний і ментор | - | + | + | + |
| Авторизація | Google, Facebook,  e-mail &  password | Google,  e-mail& password | e-mail& password | Google,  e-mail& password |
| Компонування тестів у щось подібне текам | можна компонування по колекції, але не можна зберігати колекцію в середині іншої | можна компонування по колекції, але не можна зберігати колекцію в середині іншої | - | - |
| Проходження тесту | Браузер/додаток | Браузер | Браузер | Браузер |
| Простота користувацького інтерфейсу | - | + | + | + |
| Інші особливості конкурентів | Проходження тесту вимагає присутності ментора і опитуваних в одному приміщенні | Музика на фоні, бібліотека запитань і тестів із широко-аспектним фільтром і пошуком | Бібліотека запитань | Докладна статистика відповідей, можливість експортувати у .pdf .exel |

# Аналізуючи аналоги, було виокремлено наступні сутності: тека, файл, запитання, відповідь на запитання, тест і користувач. Тека може включати у себе файли. Цими файлами можуть бути запитання, тести та інші теки із своїми файлами.

Структура даних теки:

* назва. Обмеження: максимальна довжина 40 символів, мінімальна довжина: 2 символи, заборонені символи включають у себе: @ / | \ \* ~ < >;
* колір. Маю бути типу string і відповідати стуктурі кольору типу hex(#000000 - #FFFFFF);
* дата i час створення формату: DD/MM/YYYY, HH:MM;
* дату i час останньої модифікації формату: DD/MM/YYYY, HH:MM;
* батькіський id теки типу int;
* максимальна кількість елементів типу int;
* актуальна кількість елементів типу int.

# Тест складається із запитань, які вже були створені та/або запитання, створені безпосередньо при створенні нового тесту.

Структура даних тесту:

* id корситувача, який створив тест;
* id теки розміщення;
* опис тесту;
* заголовок тесту.

Відображення або не відображення правильної відповіді при проходженні тесту, історія проходження, статистика проходження - це все можливе до реалізації, але не обов’язкове.

Тест складається із запитань, які можна додати в цій самій формі, по аналогії із Google Forms, або "скопіювати" одне із вже створених користувачем. Останнє обирається із окремої панелі.

Запитання складається із варіантів відповіді, де можна обрати тільки одну правильну відповідь.

Структура даних запитання:

* назва запитання. Обмеження: максимальна довжина 20 символів, мінімальна довжина 2 символи, заборонені символи: @ / | \ \* ~ < >.;
* текст запитання. Обмеження: максимальна довжина 200 символів, мінімальна довжина 2 символи, заборонені символи: @ / | \ \* ~ < >.;
* тип запитання. Може бути рівним “close” або “open”;
* таявність шаблону. Вказує значенням типу boolean на наявність шаблону;
* час на проходження;
* іd варіанта відповіді;
* дата i час створення формату: DD/MM/YYYY, HH:MM, Tz;
* дата i час останньої модифікації формату DD/MM/YYYY, HH:MM, Tz.

Оскільки, запитання включає у себе варіанти відповіді, то вони повинні мати структуровані дані:

* Id запитання, до якого відносяться;
* Текст відповіді. Обмеження: максимальна довжина 40 символів, мінімальна довжина: 1 символ;
* isTrue - значення типу boolean, яке вказує на правельнітсь варіанта відповіді. Якщо рівне true, то вірне, якщо false - не вірне.

До запитання може бути застосований шаблон. Він не повинен зберігатися у базі даних, оскільки його застосування має бути фішкою для WEB-версії та IOS додатку. Для версії у браузері мають бути наступні шаблони:

* так / ні;
* за / проти;
* true / false;
* yes / no.

Дані шаблони блокують зміну кількості варіантів відповіді, встановлюють значення за замовчуванням і дозволяють користувачу обрати яке із них буде вірним.

Кожне запитання, тест чи теку можна створити, видалити та відредагувати. Для цього мають бути створені окремі форми.

* 1. **Проектування програмних модулів**

## Опис продукту включає у себе перелік очікуваних до реалізації модулів, архітектурні вимоги, вказівку пріорітетів розробки, опис технологій, які знадобляться при розробці, а також огляд продукту з різних точок зору. Всі пункти можуть включати у себе діаграми та схеми, для ліпшого розуміння та уявлення як система має працювати.

Quick Test складається із програмних модулів. Програмний модуль включає у себе набір функціональних, графічних і архітектурних вимог, які спрямовані на спільний результат.

Основними програмними модулями є:

* авторизація та автентифікації: включає у себе сторінки, форми, HTTP запити та валідацію даних для реєстрації та входу в обліковий запис ментора.
* CRUD (Create Read Update Delete) запитання, а саме функціонал для створення, оновлення, перегляду та видалення.
* CRUD (Create Read Update Delete) тесту, а саме функціонал для створення, оновлення, перегляду та видалення.
* CRUD (Create Read Update Delete) теки, а саме функціонал для створення, оновлення, перегляду та видалення.
* пошукова система запитань, необхідна для пошуку запитань по назва та іншим параметрам у формі створення тесту.
* пошукова система файлів для можливості знайти тест, запитання чи теку по назві серед тих, що були створені користувачем.
* банківські платежі - мікротранзакції або пакет послуг.
* рекламний модуль, необзідний для показу нативної реклами, яка відповідає пошуковим запитам користувачів. Необхідний для отримання прибутку.
* чат бот у месенджерах Telegram, Viber та Facebook.
* магазин (Бібліотека тестів та запитань). Програмний модуль необхідний для пошуку запитань і тестів створених іншими користувачами за різними параметрами при створенні нового тесту.
* інші модулі по розширенню функціоналу запитань, тестів та тек, наприклад, переміщення між теками.

Серед них пріорітетними у розробці:

* авторизація;
* основний функціонал запитання;
* основний функціонал тесту;
* основний функціонал теки;
* рекламний модуль;
* телеграм бот;
* messanger бот.

## Система має бути реалізована із застосуванням задіяних технологій (див. Таблиця 2.3)

Таблиця 2.3

**Перелік запланованих технологій та інструменти до виконання**

|  |  |
| --- | --- |
| **Розробницька часина системи** | **Перелік запланованих технологій та інструменти до використання** |
| Офіційний Сайт (fronted) | AngularCLI, TypeScript, Webpack, Sass, NPM, TSlint, Git, Husky. |
| Панель ментора (fronted) | Angular CLI, Node.js, TypeScript, Webpack, Sass, TypeScript, Websockets, JWT, NPM, TSlint, Git, Husky. |
| Панель ментора (backend) | Nest.js, TypeScript, Node.js, PostgreSQL, JWT, NPM, TSlint, Git, Husky |
| Chatbot | Nest.JS, Node.js, WebSockets, TypeScript, TSlint, Git, Husky. |
| iOS Додаток(fronted) | JWT, Git, Husky, Alamofire, lottie-ios, GodAnimation, Strongbox, SignalRClient. |

* 1. **Проектування дії системи з точки зору різних типів користувачів**

Основними типами користувачів є гість, авторизований користувач, автентифікований користувач. Аутентифікація - перевірка, чи є хтось тим, за кого себе видає. Авторизація - перевірка, чи може автентифікований користувач виконувати певні дії (їх часто позначають як ресурси).[27]

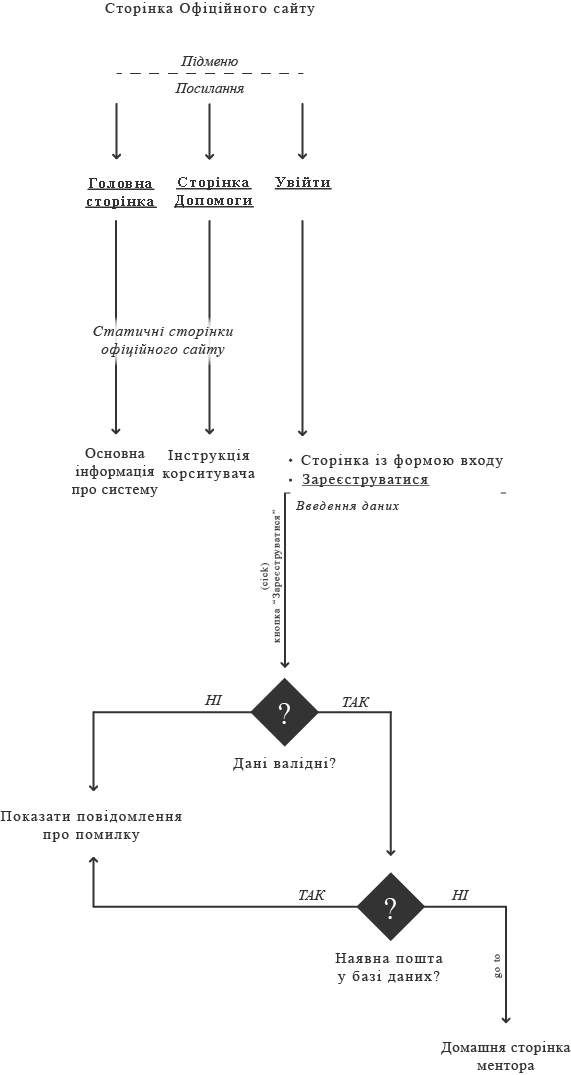


рисунок 2.2－UML діагама алгоритму дій гостя

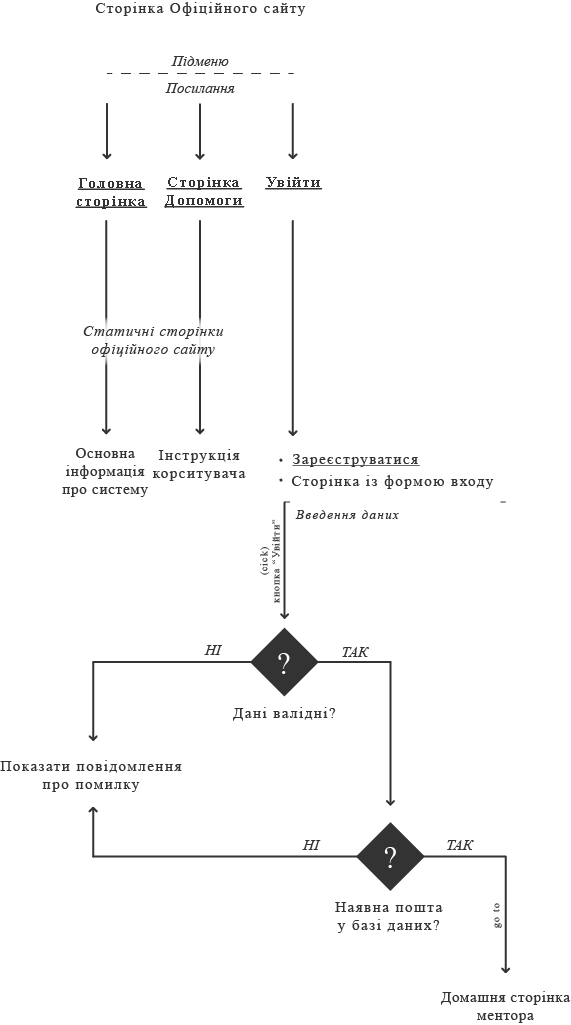


Рисунок 2.3－ UML діагама алгоритму дій неавторизованого користувача

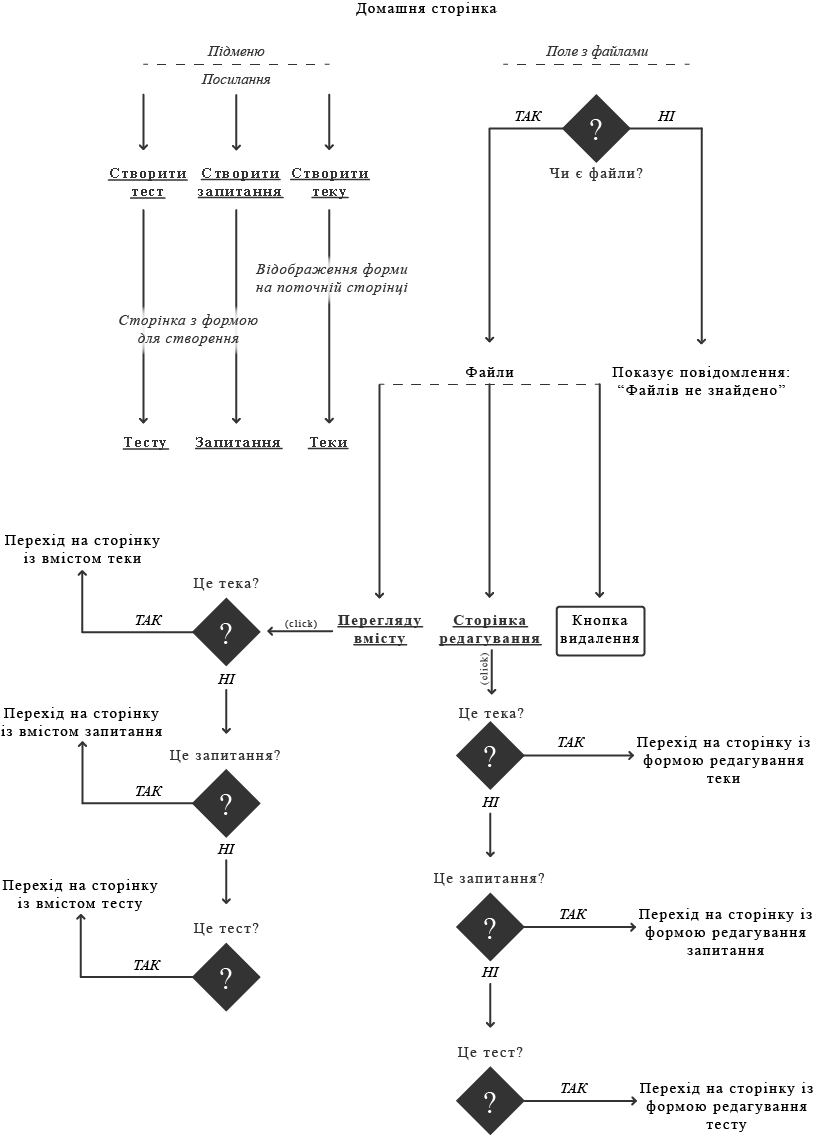


Рисунок 2.4 － UML діагама алгоритму дій авторизованого користувача

Як зображено на рисунку 2.2, гість має доступ до сторінок офіційного сайту, а саме сторінок допомоги, головної сторінки, сторінки із формою входу та реєстрації. Користувач такого типу може зареєструватися.

Автентифікований користувач може переглядати сторінки офіційного сайту і увійти в обліковий запис. Після отримання токену його статус змінюється на “авторизованого” і йому стають доступні всі ті сторінки та функціонал, що й для ментора.(див. рисунок 2.3, 2.4)

Авторизований користувач має доступ до сторінок панелі ментора та сторінки допомоги. Окрім цього ментор може:

* створювати: запитання, тести, теки;
* редагувати запитання, тести та теки;
* видаляти запитання, тести та теки;
* активувати тест для перевірки знань;
* переглядати вміст тесту, запитання, теки, статистику проходження під час того, як тест активізований;
* переміщувати одну теку в іншу і навпаки;
* вийти із облікового запису;
* оновити пароль.
  1. **Проектування користувацького інтерфейсу**

Оскільки у проекті брав участь дизайнер, то всі необхідні сторінки були заготовлені без скетчів.

Користувацький інтерфейс має бути розробленим у стилістиці логотипу, який був створений першим і який вже був представлений на публіку у на пітчінгу 2020-го року. бізнесу, виникнення якого пов’язане із сучасними інноваційними трендами у провідних країнах світу. Під пітчигом традиційно розуміють публічний виступ задля представлення та продажу свого продукту.[28] Елементи системи мають округлі форми, це відображається у шрифті, рамкам полів та формах кнопок. Також була підібрана базова палітра кольорів(див. рисунок 2.5)

Рисунок 2.5 － Палітра базових кольорів

Колір із параметром #404040 - потрібен для більшості тесту та кнопок у хедері програми. Сторінки до реаліазції: реєстрація, входу, головної сторінки сайту, головної сторінки панелі ментора, сторінки свторення тесту та запитаня, відображення форми створення теки, сторінки перегляду тесту, таки та запитання. Сторінки вісту тек та форми стоврення мають бути розроблені під типи екранів 375 пікселів, 640 пікселів, 1024 пікселів та 1440 пікселів.

Голова сторінка включає у себе 4 секції, де кожна секція показує можливості системи та має стислий опис того як користуватися системою студенту та ментору. Зображені на них кнопки є посиланнями на сторінку входу.(див. рисунок 2.6)

Дизайн даної сторінки був реалізований тільки під розмір екрану 375 пікселів у ширину, оскільки це ширина екрану iPhone 5. На момент розробки система була розрахована тільки під мобільний сектор ринку, це стосується і дизайну сторінок входу та реєстрації. Усі подальші сторінки були реалізовані під різну ширину екранів.

На етапі проектування було зазвначено сторінку допомоги як онду із складових частини Офіційного сайту, вона так само як і сторнки редагування, перегляду та видалення не були розроблені через те, що навіть у при проектуванні процесу розробки, описаному у першому підпункті поточного розділу, крайня дата реалізації набагато пізніше, за здачу роботи. Тому ці та перелічені сторінки мають дизайн, що спирається на вже розроблені стилі. Наприклад, поля вводу, кнопок та посилань, що мають схожі риси з поправкою на розмір і відступи від них.

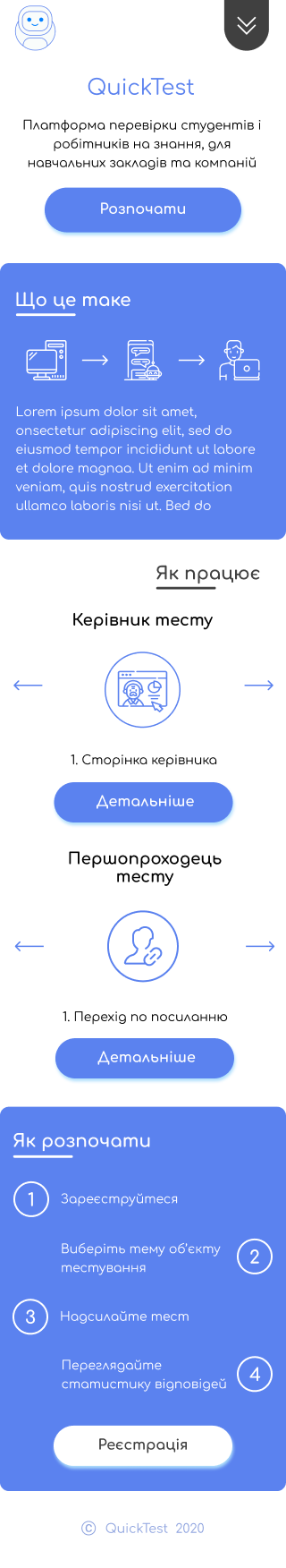


Рисунок 2.6 － Головна сторінка сайту

Наступними сторінками по алгоритму розробки є форми входу та реєстрації. Форма реєстрації повинна мати поля вводу імені, e-mail та паролю з повторенням. Дія створення нового облікового запису має затверджуватися натисненням на кнопку з написом “Зберегти”.(див. рисунок 2.7)

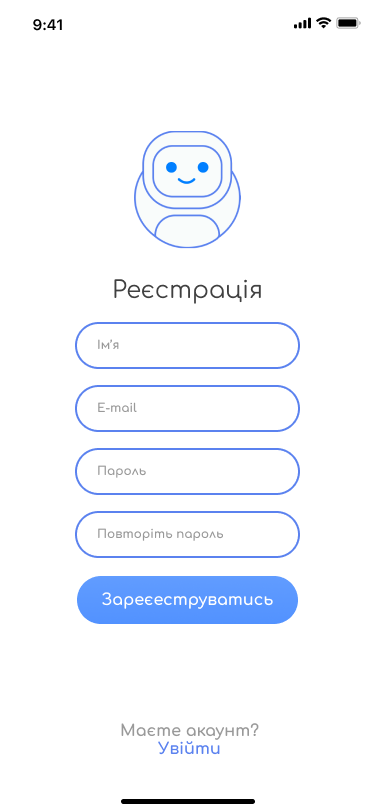


Рисунок 2.7 － Сторінка реєстрації

Форма для входу має ідентичний дизайн окремих її частин: полів вводу, кнопки, підменю для переключення на форму реєстрації. Форма на рисунку 2.8 має поля для вводу пошти та паролю.



Рисунок 2.8 － Дизайн Сторінка входу

При виникненні помилки вводу чи при перевірці існування користувача у БД, мають бути показані повідомлення.(див. рисунок 2.9)

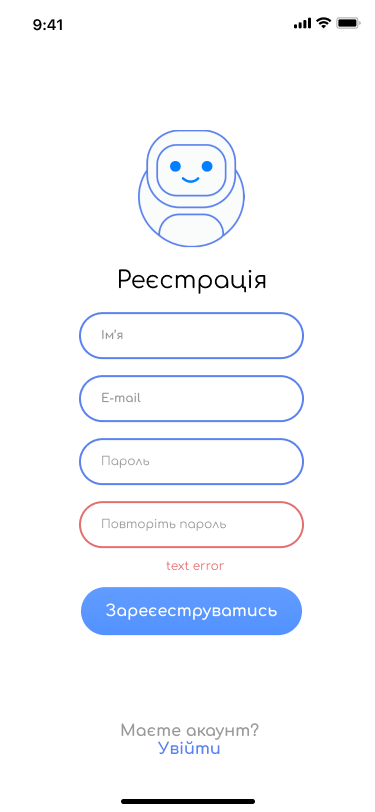


Рисунок 2.9 － Дизайн схематичного розміщення тексту помилки

Наступними сторінками є головна сторінка панелі ментора. Вона має включати у себе відображення назви поточної теки, перелік файлів та адаптивне меню із списком посилань на сторінки створення екземплярів сутностей. Останнє доступне до перегляду після натиснення на зелену кнопку у правому верхньому кутку.(див. рисунок 2.10) При натисненні на кнопку, яке має відкриватися натисненням на зелену кнопку(див. рисунок 2.11)



Рисунок 2.10 － Дизайн головної сторінки панелі ментора для екрану шириною 1024 пікселів

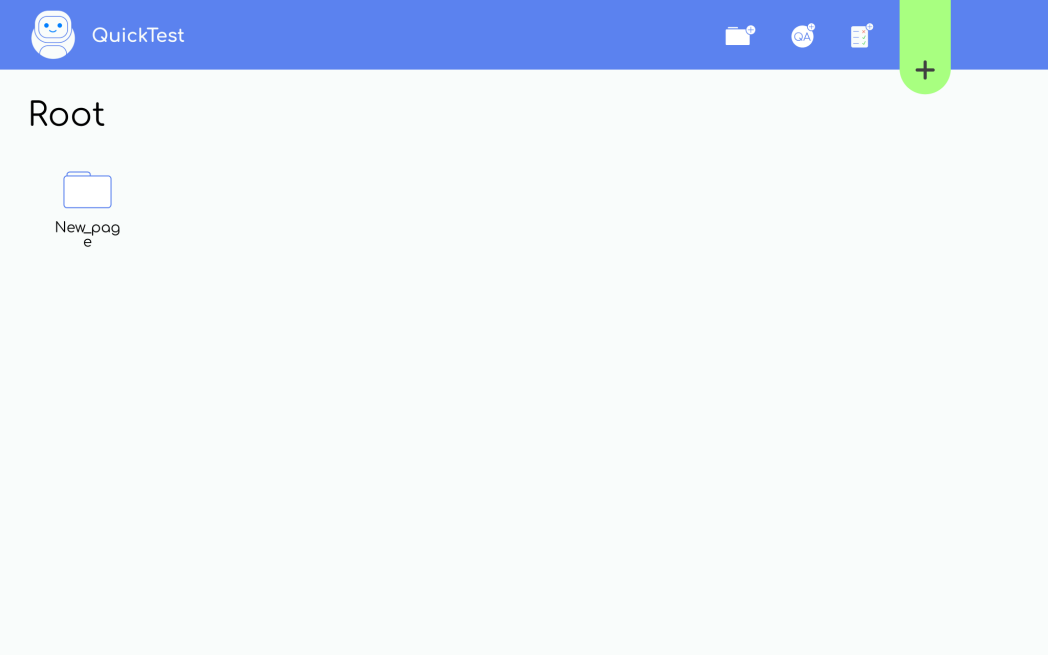


Рисунок 2.11 － Дизайн відкритого меню для додання елементів головної сторінки панелі ментора для екрану шириною 1024 пікселів

Для менших розмірів усі посилання мають бути винесені в окреме меню, яке має бути зафіксоване внизу екрану.(див. рисунок 2.12)

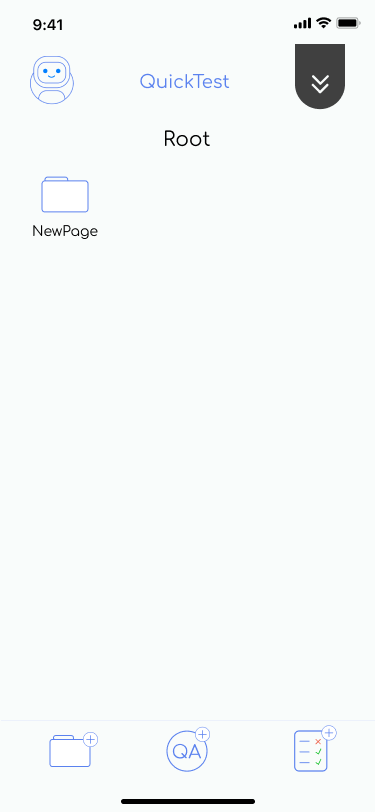


Рисунок 2.12 － Головна сторінка панелі ментора

Так як згідно ТЗ та описаним вимогам форма створення теки має бути на сторінці вмісту поточної теки і займати стільки ж місця, як і створений файл теки, то форма повинна мати вигляд як на рисунку 2.13.

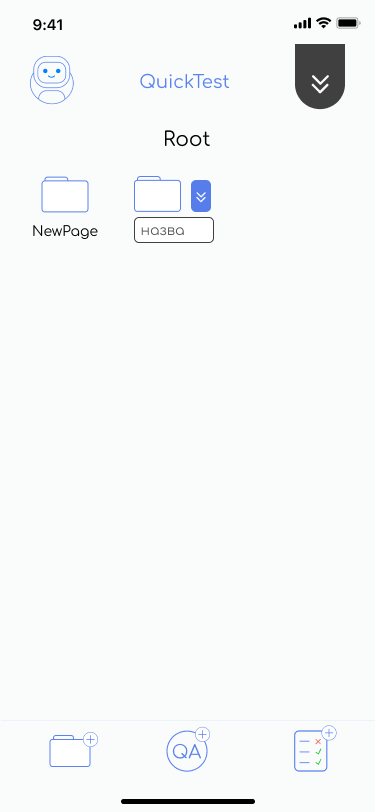


Рисунок 2.13 － Вигляд форми створення теки

При переході за посиланням для створення новго запитання має відбутися перехід на сторінку,я ка містить поле для вводу назви запитання, велике тектове поле для вводу тексту запитання. Нижче має бути інтерактивний елемент випадаючого списку із вибором типу відповіді. Це має бути вибір між “один із багатьох”, “багато із багатьох”, але через складнощі проектуванням цієї частини, цей елемент має бути замінено на вибір шаблону та часу на надання відповіді.(див. рисунок 2.14) Також форма повинна мати посилання для переходу на попередню сторінку та кнопка збереження. Обидва елементи мають бути розміщені внизу форми.

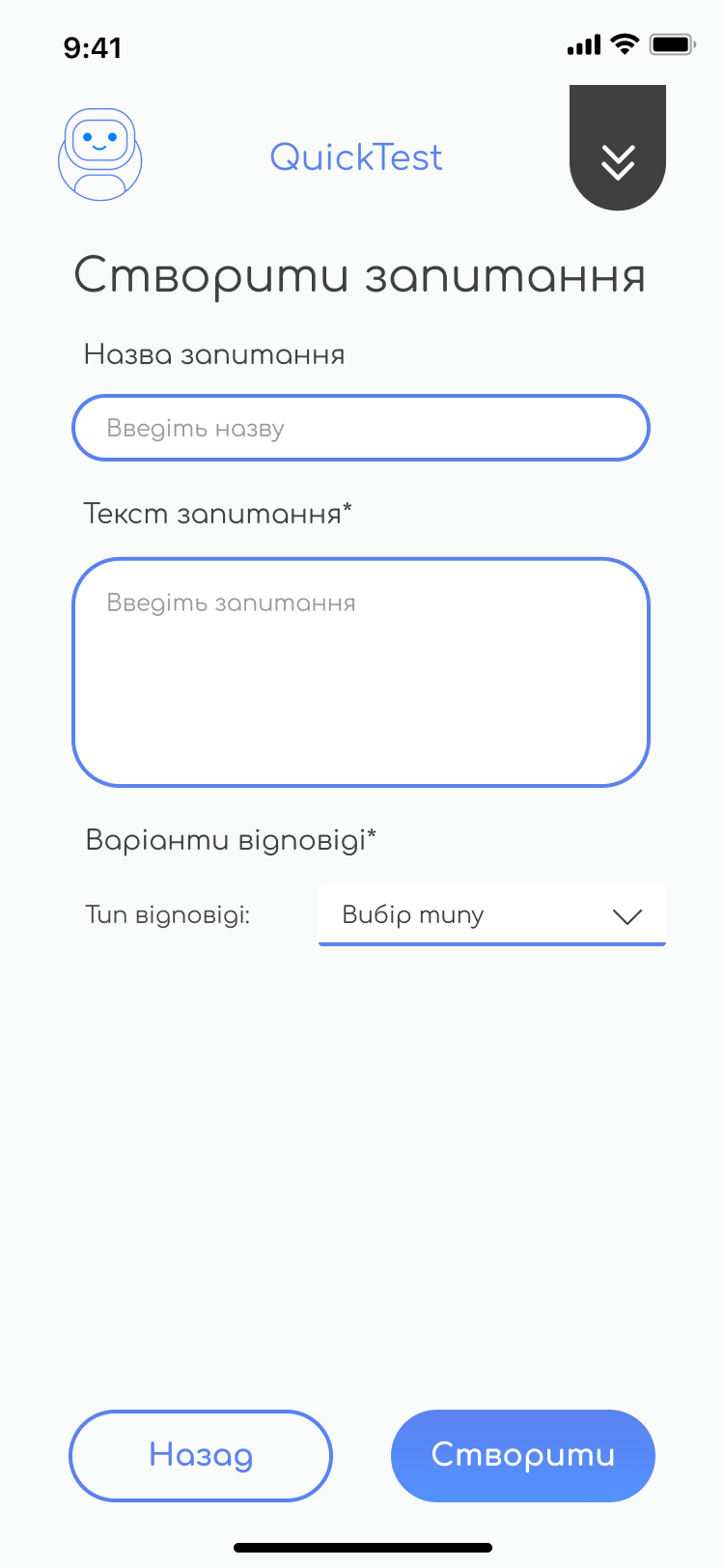


Рисунок 2.14 － Дизайн сторінки форми для створення запитання розміру екрану 375 пікселів

Після вибору шаблону має з’явитися перелік відповідей, і якщо обрано “Власне”, то поля доступні до редагування.(див. рисунок 2.15)

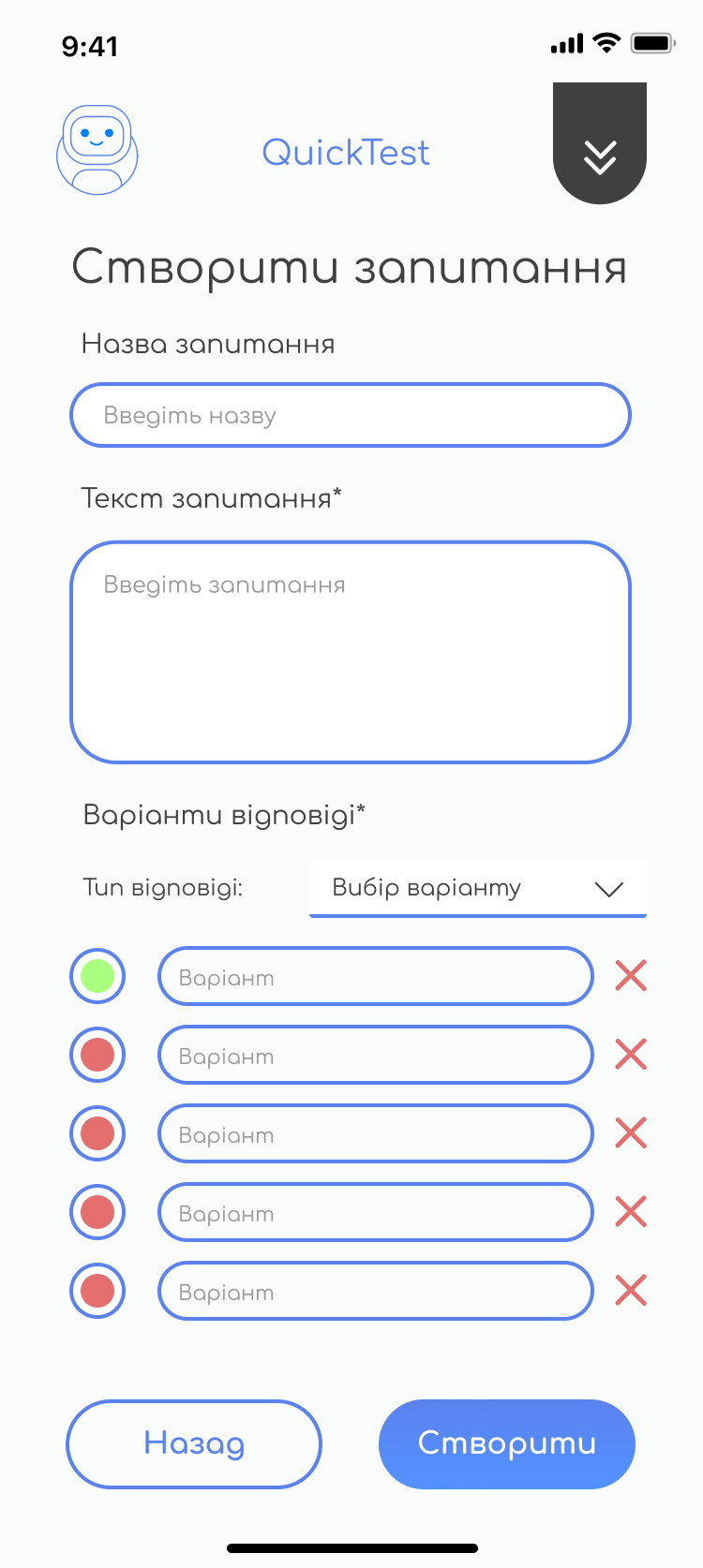


Рисунок 2.15 － Дизайн варіантів відповіді

Варіанти відповіді, якщо їх більше 2х, можна видалити натиснувши на кнопку у вигляді зображення червоного повернутого хреста. На залежно від кількості відповідей, користувач може обрати той варіант відповіді, який має бути правильним, натиснувши на поле вводу типу radio.

Якщо користувач ввів некоректні дані, то під полем має бути відображений текст і рамка полягає змінити колір на червоний, аналогічно формі входу та реєстрації.(див.рисунок 2.9)

Якщо дані було введено коректно, то відбувається перехід на головну сторінку із показом новоствореного запитання. Згідно дизайну, усі файли мають бути відсортовані за датою оновлення. А саме спочатку мають бути найновіші.

Після користувач натискає на посилання створення тесту і переходить на сторінку із формою, яка має включати у себе поле для вводу назви тесту, велике текстове поле для вводу його опису, список карток вже створених запитань і форма для створення запитання в середині тесту.

Картка запитання із списку запитань має включати у себе назву запитання, час на надання відповіді та тип шаблону відповіді, який застосовується.

Форма має включати у себе текстове поле вводу тексту запитання, випадні списки із вибором шаблону відповіді та часу. На цьому етапі розробки дизайну виникли проблеми із реалізацією його на екрані, тому було розроблено скетч(див. рисунок 2.16) тому при реалізації цієї сторінки допускаються допущення.

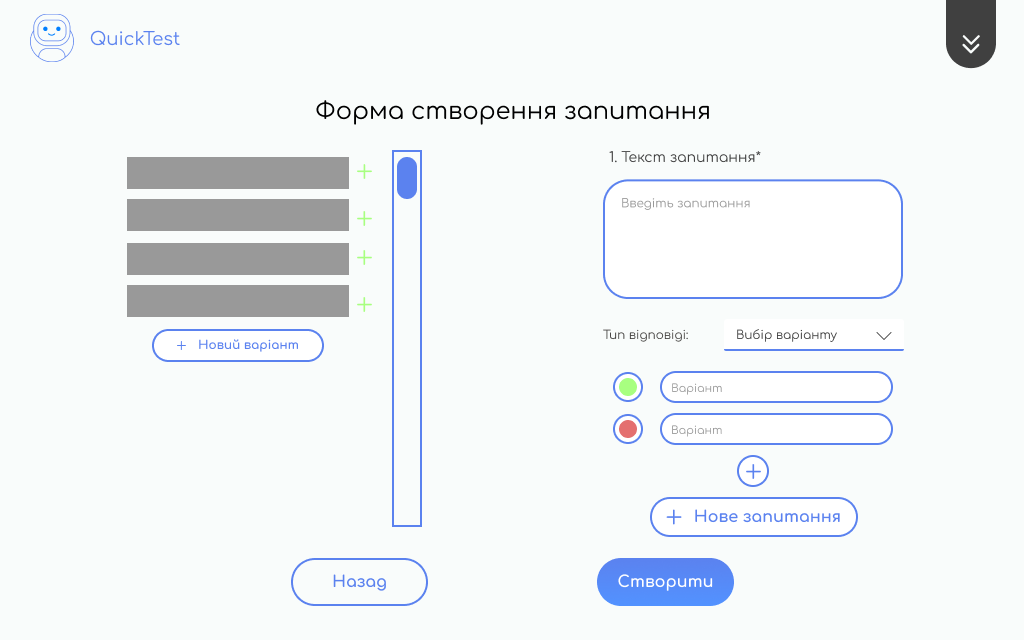


Рисунок 2.16 － Скетч сторінки створення тесту із запитаннями

При реалізації допускають погрішності та невідповідність у порядку розміщення тощо.

* 1. **Проектування бази даних**

Беручи за основу головні сутності та розробленпий дизайн користувацького інтрефейсу, було розроблено інфологічну схему бази даних. Таким чином було виділено, що модель повинна складатися із 6ти об’єктів: запитання, варіант відповіді, тест, тека, користувач, студент.(див. рисунок 2.17)

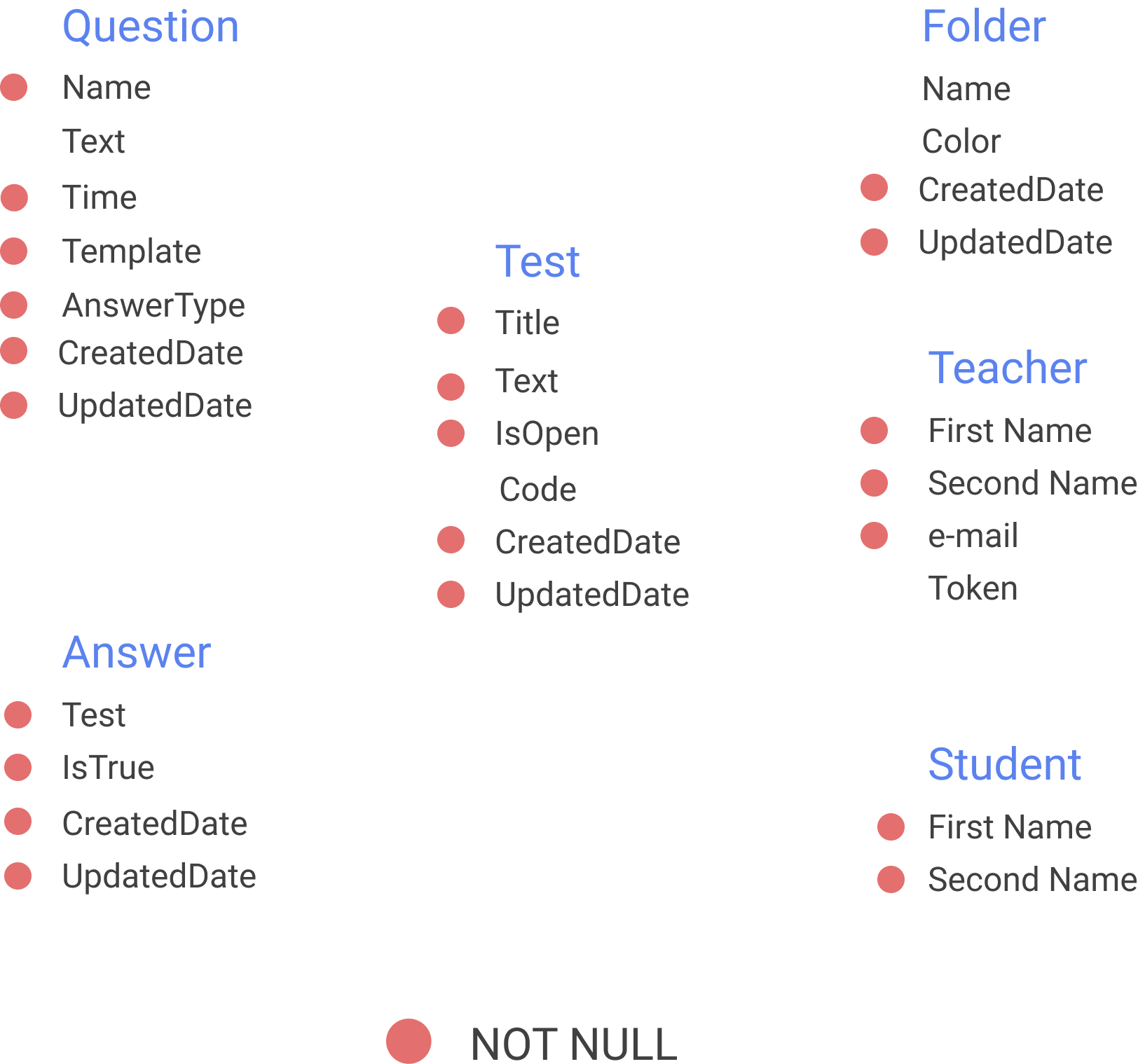


Рисунок 2.17 － Інфологічна діаграма бази даних

Для реалізації бази даних відповідно до інфологічної моделі, база даних повинна мати зв’язні таблиці для:

* поєднання одних сутностей з іншими, які включають у себе таку залежність;
* мінімізації дублювання даних;
* структуризації даних.

Таким чином, окрім таблиць для опису сутностей, було реалізовано проміжні таблиці:

* question\_answer - зв’язує зв’язки один до багатьох дані таблиць questions та answers;
* test\_question - зв’язує зв’язки один до багатьох дані таблиць test та questions;
* folder\_file - зв’язує зв’язки один до багатьох дані таблиць folders та files. Files теж є проміжною таблицею, яка за допомогою зв’язків одного до багатьох, зв’язує тест, запитання та інші теки з однойменних таблиць;
* mentor\_file - зв’язує зв’язки один до багатьох дані таблиць mentors та files, які були створені ментором.

Згідно цієї модулі була розроблена база даних, структура якої знаходиться на рисунку №. Вона була реалізована мовою програмування PostrgeSQL на базі мови SQL.(див. рисунок 2.17)

Згідно рисунку 2.18 далі виконано детальніший опис таблиць сутностей: questions, students, mentor, answers, tests, folders, files; та проміжних: student\_test\_answers, questions\_answers, mentor\_files.



Рисунок 2.18 － Даталогічна модель бази даних

Так як весь основний функціонал та особливість проекту зав’язані на сутності запитання, то першою таблицею до розробки є “Questions”(див. таблиця 2.4)

Таблиця 2.4

**Таблиця запитань (questions)**

|  |  |  |
| --- | --- | --- |
| **questions** | | |
| **Назва поля** | **Тип даних формату SQL** | **Опис поля** |
| id | Varchar(36) | Унікальний ідентифікатор, для прив’язки у проміжних таблицях. |
| name | Varchar(20) | Назва запитання, подібна наві файлу. |
| text | Varchar(255) | Текст запитання. |
| time | int | Час на надання відповіді. |
| template | boolean | Вказує на наявність використання шаблону запитання. |
| image | link | Для майбутнього додання до проекту зображення. |
| answer\_type | enum | Застосування типу відповіді у чат боті. |
| id\_deleted | boolean | Вказує на видаленість. |
| created | date | Дата створення запитання. |
| updated | date | Дата оновлення запитання. |

Оскільки, згідно ТЗ запитання включає у себе дві відповіді мінімум, то має бути розроблена таблиця із описом полів відповіді(див. таблиця Відповіді) та проміжна таблиця для прив’язки відповіді до запитання(див. таблиця 2.5).

Таблиця 2.5

**Таблиця відповідей (answers)**

|  |  |  |
| --- | --- | --- |
| **Назва поля** | **Тип даних формату SQL** | **Опис поля** |
| id | Varchar(36) | Унікальний ідентифікатор, для прив’язки у проміжних таблицях. |
| name | Varchar(100) | Текст відповіді. |
| Is\_true | boolean | Вказівка на правильність відповіді. |
| created | date | Дата створення. |
| updated | date | Дата оновлення. |

Таблиця 2.6

**Проміжна таблиця між відповідями та запитанням(questions\_answers)**

|  |  |  |
| --- | --- | --- |
| **Назва поля** | **Тип даних формату SQL** | **Опис поля** |
| Id\_question | Varchar(36) | Унікальний ідентифікатор запитання. |
| Id\_answer | Varchar(36) | Унікальний ідентифікатор відповідь. |

Таблиця 2.6 має зв’язки один до одного у полі Id\_question і зв’язок багато до багатьох із сторони поля Id\_answer.

Для того щоб застосувати запитання до тесту має бути реалізована таблиця із описом полів тесту(див. таблиця 2.7).

Таблиця 2.7

**Таблиця із тестами tests**

|  |  |  |
| --- | --- | --- |
| **Назва поля** | **Тип даних формату SQL** | **Опис поля** |
| id | Varchar(36) | Унікальний ідентифікатор, для прив’язки у проміжних таблицях. |
| Owner\_id | Varchar(40) | Унікальний ідентифікатор ментора. |
| status | enum | Статус активності тесту. |
| code | Varchar(6) | Код доступу до проходження тесту. |
| Show\_question\_result | boolean | Вказівка для показу правильної відповіді опитуваному. |
| created | date | Дата створення. |
| updated | date | Дата оновлення. |

Для поєднання тесту із запитанням має існувати проміжна таблиця для зв’язку тесту із запитанням(див. таблиця 2.8)

Таблиця 2.8

**Проміжна таблиця запитань і тесту (test\_questions)**

|  |  |  |
| --- | --- | --- |
| **Назва поля** | **Тип даних формату SQL** | **Опис поля** |
| Id\_test | Varchar(36) | Унікальний ідентифікатор тесту. |
| Id\_question | Varchar(36) | Унікальний ідентифікатор запитання. |

Поля таблиці 2.9 є аналогічними тим, що й у проміжній таблиці відповідей та запитання.

Дані файли має створювати ментор, для цього має існувати таблиця для опису його полів, зоб система могла його автентифікувати і надати доступ до створених ним файлів. (див. таблиця 2.9)

Таблиця 2.9

**Таблиця бази даних з даними ментора (mentors)**

|  |  |  |
| --- | --- | --- |
| **Назва поля** | **Тип даних формату SQL** | **Опис поля** |
| Id | Varchar(36) | Унікальний ідентифікатор, для прив’язки у проміжних таблицях. |
| Name | Varchar(40) | Унікальний ідентифікатор запитання. |
| E\_mail | Varchar(30) | Електронна пошта ментора. |
| hash | Varchar(255) | Зашифровані дані пароля ментора. |
| account | Enum | Тип акаунта, для реалізації доступу до функціоналу в майбутньому. |

Так як для авторизованого користувача потрібно показувати створені ним файли, то має існувати проміжна таблиця для зв’язування даних користувача із файлом.(див. таблиця 2.10)

Таблиця 2.10

**Проміжна таблиця файлів з користувачем (mentor\_files)**

|  |  |  |
| --- | --- | --- |
| **Назва поля** | **Тип даних формату SQL** | **Опис поля** |
| Id\_file | Varchar(36) | Унікальний ідентифікатор файлу. |
| Id\_mentor | Varchar(36) | Унікальний ідентифікатор ментора. |

Поля таблиця mentor\_files пов’язуються аналогічно всім вище описаним таблицям.

Файли теж потрібно зберігати у базі даних таким чином має бути та, яка має відповідати таблиці 2.11.

Таблиця 2.11

**Таблиця файлів (files)**

|  |  |  |
| --- | --- | --- |
| **Назва поля** | **Тип даних формату SQL** | **Опис поля** |
| Id | Varchar(36) | Унікальний ідентифікатор файлу. |
| Parent\_id | Varchar(36) | Унікальний ідентифікатор теки. Поля теки див. таблицю |
| Child\_id | Varchar(36) | Унікальний ідентифікатор запитання, тесту чи теки. Регулюється валідатором у коді. |

Таблиця 2.12

**Таблиця теки (folders)**

|  |  |  |
| --- | --- | --- |
| **Назва поля** | **Тип даних формату SQL** | **Опис поля** |
| Id | Varchar(36) | Унікальний ідентифікатор теки для проміжних таблиць. |
| name | Varchar(20) | Назва теки. |
| color | Varchar(7) | Колір іконки теки. Відповідає формату hex, включаючи знак ришитки (#). |

Не варто забувати, що описаний контент у певному розмінні має бути доступний користувачам типу “студент”, дані яких теж потрібно зберігати. Як наслідок має існувати таблиця 2.13.

Таблиця 2.13

**Таблиця студентів(students)**

|  |  |  |
| --- | --- | --- |
| **Назва поля** | **Тип даних формату SQL** | **Опис поля** |
| Id | Varchar(36) | Унікальний ідентифікатор студента. |
| messager | enum | Тип месенджера, у якому відбувається тестування. |
| Name | Varchar(40) | Ім’я. отримане із меседжера, для показу статистики ментору. |

Дані студента потрібні для того, щоб ментор міг переглядати статистику відповідей. Для реалізації такого функціоналу в базі даних мають зберігатися у таблицях із поєданням даних тесту, запитання і відповіді студента. Так як статистика має бути у майбутньому доступною окремо від процесу проходження тесту, яке планується у майбутньому.

Таблиця 2.14

**Сумісна таблиця динамічної статистики(student\_test\_answer)**

|  |  |  |
| --- | --- | --- |
| **Назва поля** | **Тип даних формату SQL** | **Опис поля** |
| Id\_student | Varchar(36) | Унікальний ідентифікатор студента. |
| Id\_answers | Varchar(36) | Унікальний ідентифікатор відповіді. |
| Id\_questions | Varchar(36) | Унікальний ідентифікатор запитання. |
| Id\_test | Varchar(36) | Унікальний ідентифікатор тест. |

Описані таблиці мають бути реалізовані мовою SQL PostgreSQL.

* 1. **Проектування архітектури**

Усі частини системи повинні відповідати певному шаблону програмування. Це типовий спосіб вирішення певної проблеми, що часто зустрічається при проектуванні архітектури програм, необхідний для стандартизації коду, зменшення часу на розробку, полегшення розуміння програмного коду.

На відміну від готових функцій чи бібліотек, патерн не можна просто взяти й скопіювати в програму. Патерн являє собою не якийсь конкретний код, а загальний принцип вирішення певної проблеми, який майже завжди треба підлаштовувати для потреб тієї чи іншої програми.[29]

Описи шаблонів зазвичай дуже формальні й найчастіше складаються з таких пунктів:

* проблема, яку вирішує патерн(pattern);
* мотивація щодо вирішення проблеми способом, який пропонує патерн;
* структура класів, складових рішення;
* приклад однією з мов програмування;
* особливості реалізації в різних контекстах;
* зв’язки з іншими шаблонами. [29]

Back-end частина має бути використана вбудована архітектура фреймворку. Дана архітектура підтримує мікросервісний архітектурний стиль розробки.[30]

Більшість концепцій, обговорених в документації, таких як введення залежностей, декоратори, фільтри винятків, труби(pipes), захисні елементи(guards) та перехоплювачі, однаково стосуються мікросервісів.(див. рисунок 2.19)

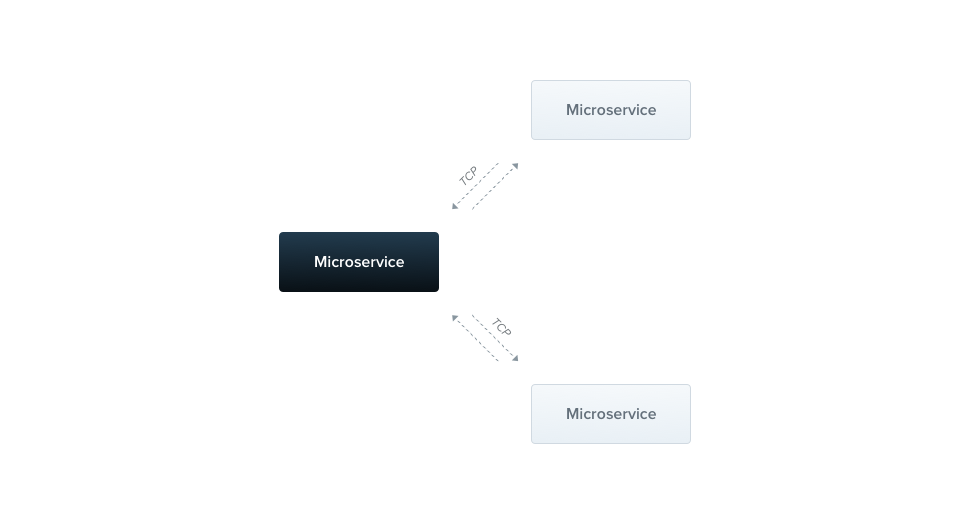


Рисунок 2.19 － Схема взаємодії мікросервісів

Згідно цієї концепції архітектури, back-end частина повинна складатися із мікросервісів, у випадку із цим проектом - модулями. Кожен модуль зав’язаний на сутності, включає у себе entities(суб’єкти), сервіси, контроллера(controller), dto - об'єкт передачі даних - це об'єкт, який визначає, як дані будуть передаватися по мережі [31]. Контролери повинні обробляти HTTP-запити та делегувати більш складні завдання провайдерам- Сервіси включають у себе обробку запитів контроллера.[32] Entities для додавання об’єктів вручну до масиву сутностей параметрів підключення може бути нудним. Крім того, посилання на сутності з кореневого модуля порушує межі домену програми та спричиняє витік деталей реалізації до інших частин програми.[33]

Програмний код клієнстської чатини повинен відповідати патерну MVC. MVC Pattern розшифровується як Model-View-Controller Pattern. Цей шаблон використовується для відокремлення частин програми.

Модель(Model) - модель представляє об'єкт або JAVA POJO, що містить у собі опис полів сутності. Він також може мати логіку для оновлення контролера, якщо його дані змінюються.

Перегляд(View) - подання представляє візуалізацію даних, що містять модель.

Контролер(Controller) - Контролер діє як на модель, так і на вигляд. Він контролює потік даних в об'єкт моделі та оновлює дані для перегляду, коли вони змінюються. Таким чином view та модель відокремлюються.[34]

Через особливості фреймворку Angular, контролер складається із 2х частин: компонента і сервісу. Компонент включає у себе бізнес логіку, яка обробляє дії користувача при взаємодії із користувацьким інтерфейсом. При необхідності надіслати чи отримати дані з бази даних, викликаються методи із сервісів. Сервіси включають у себе http запити на back-end.

Усі спільні статичні дані, що використовується у кількох мікросерсвісах мають зберігатися у теці utils.

На front-end частині WEB-версії системи очікується файлова система поділена на мікросервіси аналогічно до back-end. Для останнього файлова система має бути аналогічною. Один модуль - одна сутність чи блок функціоналу. Схематично це має виглядати наступним чином:

├── environments

│ └── environment.ts

├── src

│ └── app

│ │ └── modules

│ │ │ └── auth

│ │ │ │ └── components

│ │ │ │ │ └── login

│ │ │ │ │ │ └── login.component.ts

│ │ │ │ │ │ └── login.component.ts

│ │ │ │ │ └── registration

│ │ │ │ │ │ └── registration.component.ts

│ │ │ │ │ │ └── registration.component.ts

│ │ │ │ └── views

│ │ │ │ │ └── login

│ │ │ │ │ │ └── login.view.ts

│ │ │ │ │ │ └── login.view.ts

│ │ │ │ │ └── registration

│ │ │ │ │ │ └── registration.view.ts

│ │ │ │ │ │ └── registration.view.ts

│ │ │ │ │ └── auth.styles.sass

│ │ │ │ └── User.class.ts

│ │ │ │ └── auth.quard.ts

│ │ │ │ └── auth.service.ts

│ │ │ │ └── auth.module.ts

│ │ │ │ └── auth.routing.ts

│ │ │ └── folder

│ │ │ │ └── components

│ │ │ │ └── views

│ │ │ │ └── models

│ │ │ │ │ └── Model.class.ts

│ │ │ │ │ └── Model.interface.ts

│ │ │ │ │ └── Model.enum.ts

│ │ │ │ └── folder.quard.ts

│ │ │ │ └── folder.service.ts

│ │ │ │ └── folder.module.ts

│ │ │ │ └── folder.routing.ts

│ │ │ └── test

│ │ │ │ └── components

│ │ │ │ └── views

│ │ │ │ └── models

│ │ │ │ │ └── Model.class.ts

│ │ │ │ │ └── Model.interface.ts

│ │ │ │ │ └── Model.enum.ts

│ │ │ │ └── test.quard.ts

│ │ │ │ └── test.service.ts

│ │ │ │ └── test.module.ts

│ │ │ │ └── test.routing.ts

│ │ │ └── question

│ │ │ │ └── components

│ │ │ │ └── views

│ │ │ │ └── models

│ │ │ │ │ └── Model.class.ts

│ │ │ │ │ └── Model.interface.ts

│ │ │ │ │ └── Model.enum.ts

│ │ │ │ └── question.quard.ts

│ │ │ │ └── question.service.ts

│ │ │ │ └── question.module.ts

│ │ │ │ └── question.routing.ts

│ │ │ └── site

│ │ │ │ └── page-name

│ │ │ │ │ └── page-name.view.html

│ │ │ │ │ └── page-name.view.ts

│ │ │ │ │ └── page-name.view.sass

│ │ │ │ └── site.module.ts

│ │ │ │ └── site.routing.ts

│ │ │ └── UI

│ │ │ │ └── ui-element-name

│ │ │ │ │ └── ui-element-name.component.html

│ │ │ │ │ └── ui-element-name.component.ts

│ │ │ │ │ └── ui-element-name.component.sass

│ │ │ │ └── ui.module.ts

│ │ └── app-routing.module.ts

│ │ └── app.module.ts

│ │ └── remove.js

│ └── utils

│ │ └── enums

│ │ │ └── some.enum.ts

│ │ └── pipes

│ │ │ └── some.pipe.ts

│ │ └── consts

│ │ │ └── some.const.ts

│ │ └── guards

│ │ │ └── some.guard.ts

│ │ └── services

│ │ │ └── some.service

│ │ │ │ └── some.service.ts

│ │ └── styles

│ │ │ └── file.sass

│ └── app.component.html

│ └── app.component.sass

│ └── app.component.ts

│ └── assets

│ │ └── file

│ └── favicon.ico

Тека src включає у себе три основні теки: app, де знаходяться всі компоненти, сервіси, модулі(modules), pipes тощо; assets - тека, де зберігаються всі статичні картинки та іконки; environments - тека для зберігання даних для підключення до бекенд серверу.

Тека app є кореневою для всіх “робочих” файлів сайту, з якими на пряму працюють розробники. Там описуються всі модулі та їх похідні(компоненти, сервіси, файли маршрутизації, файли стилізації тощо). Ця тека включає у себе: теку modules, де описані модулі сутностей сайту; utils - тека із спільним файлами, які складно віднести до якогось одного модуля. Всі файли цієї теки або мають власний модуль, який імпортується у необхідні модулі, де вони є потрібними; або імпортуються у глобальний app.module.ts - кореневий модуль, де реєструються всі модулі та спільні для всіх модулів файли.

Файли теки module поділені за основними сутностями і логікою, яка їх поєднує. Кожна тека модуля має теку із компонентом, теку із файлами для відображення статичної сторінки без бізнес логіки - view, файл сервісу чи теку із сервісами, файл із описом моделі, який використовує модуль, чи теку з ними; тека обов’язково включає у себе файл із описом модуля відповідного до назви теки. Згідно із документацією Angular CLI усі компоненти, сервіси, pipes тощо у модуль. Якщо цей модуль є не кореневим(App.module.ts), то він має імпортуватися у кореневий AppModule і реєструватися там у відповідному полі полі об’єкту анотації.

Назви файлів повинні містити у собі назви назву теки модуля. Для компонентів назви класів повинні відповідати схемі: NameModuleComponent, для views назва відповідна: NameModuleViewComponent, це потрібно щоб, по-перше, назви відповідали анотації, завдяки якій клас статичної сторінки ініціалізується.

**Висновок до другого розділу**

Були виокремленні головні сутності проекту, серед них: тека, запитання, тест, користувач типу ментор та користувач типу студент. Тест може містити у собі створені запитання, тека може містити у собі як сутності запитання та тесту так й іншу теку.

Сутності мають бути розбиті між програмними модулями. Вони пов’язані із базовими функціоналом серед якого: реєстрація та вхід в обліковий запис, створення редагування та перегляд вмісту запитання, тесту та теки, запуск тесту, офіційний сайт та чат бот.

Користувачі системи поділяються на неавторизованого, авторизованого, автентифікованого та студента(того, хто проходить тест у чат-боті). Для кожного із них було розроблено своє діаграми алгоритму можливих відкриття сторінок.

Щоб система працювала цілісно і була доступною до масштабування у майбутньому було підібрано та частково розроблено архітектурні шаблони для окремих її частин. Архітектура включає у себе файлову систему та взаємодію файлів. Для бекенду - система мікросервісів, для фронтенду - MVC із системою мікросервісів та модифікаціями у розбивці коду за призначенням запитів. Файлова система має складатися із головних тек: app, utils, assets. У середині теки app має знаходитися, окрім кореневих файлів компонентів, теки мікросервісів розбитими за головним сутностями, якщо сутність має компоненту із власним функціоналом, то вона зберігається у вигляді мікросервісу в середині теки children.

Щоб реалізувати систему був розроблений і описаний план роботи. Він включає у себе таблицю задач із розписаними по датам початку та завершення дій у залежності від пріорітетності та залежностей функціональних одиниць. Задачі поділені таким чином, щоб сама реалізація відповідала шаблону проектування Ajax, а саме Scrum.

1. **РОЗРОБКА ТА ТЕСТУВАННЯ СИСТЕМИ**
   1. **Розробка бази даних**

База даних була реалізована за допомогою інструменту PgAdmin4, де було створено окремий сервер для неї та безпосередньо базу даних із таблицями, що були описані у попередньому розділі, включно із усіма таблицями. Створення бази даних було за допомогою інтерфейсу інструменту. Створення тек було реалізовано за допомогою QueryTool. Код реалізації таблиць див.ись у додатку (Щось там про код реалізації БД).

Були реалізовані таблиці: questions, students, mentors, answers, tests, folders, files; та проміжних: student\_test\_answers, questions\_answers, test\_questions, mentor\_files. Із запланованими зв’язками один до багатьох.

Таким чином було реалізовано наступні таблиці:

* questions. Код реалізації мовою SQL:

CREATE TABLE "questions" (

"id" SERIAL PRIMARY KEY,

"name" varchar(20) NOT NULL,

"text" varchar(255) NOT NULL,

"time" int NOT NULL,

"template" enum NOT NULL,

"image" link,

"answer\_type" enum NOT NULL,

"id\_deleted" boolean NOT NULL,

"created" date NOT NULL,

"updated" date NOT NULL

);

* students. Код реаліазції мовою SQL:

CREATE TABLE "students" (

 "id" SERIAL PRIMARY KEY,

  "messager" enum NOT NULL,

  "name" varchar(40) NOT NULL

);

* mentors. Код реаліазції мовою SQL:

CREATE TABLE "mentors" (

"id" SERIAL PRIMARY KEY,

"name" varchar(40) NOT NULL,

"e\_mail" varchar(30) NOT NULL,

"hash" varchar(255) NOT NULL,

"account" enum NOT NULL

);

* answers. Код реаліазції мовою SQL:

CREATE TABLE "answers" (

"id" SERIAL PRIMARY KEY,

"name" varchar(100) NOT NULL,

"is\_true" boolean NOT NULL,

"created" date NOT NULL,

"updated" date NOT NULL

);

* tests. Код реаліазції мовою PostgresSQL:

CREATE TABLE "tests" (   "id" SERIAL PRIMARY KEY,   "owner\_id" varchar(36) NOT NULL,   "status" enum NOT NULL,   "code" varchar(6),   "show\_question\_result" boolean NOT NULL,   "created" date NOT NULL,   "updated" date NOT NULL );

* folders. Код реаліазції мовою PostgresSQL:

CREATE TABLE "folders" (

"id" SERIAL PRIMARY KEY,

"name" varchar(20) NOT NULL,

"color" varchar(7) NOT NULL

);

* files. Код реаліазції мовою PostgresSQL:

CREATE TABLE "files" (

"id" SERIAL,

"parent\_id" SERIAL,

"child\_id" SERIAL,

PRIMARY KEY ("id", "parent\_id", "child\_id")

);

та проміжні:

* student\_test\_answers. Код реаліазції мовою PostgresSQL:

CREATE TABLE "student\_test\_answers" (

"id\_student" varchar(36),

"id\_answers" varchar(36),

"id\_questions" varchar(36),

"id\_test" varchar(36)

);

* questions\_answers. Код реаліазції мовою PostgresSQL:

CREATE TABLE "questions\_answers" (

"id\_question" varchar(36) NOT NULL,

"id\_answer" varchar(36) NOT NULL

);

* mentor\_files. Код реаліазції мовою PostgresSQL:

CREATE TABLE "mentor\_files" (   "id\_mentor" varchar(36),   "id\_file" varchar(36)).

Для реалізації зв’язків між таблицями сутностей та їх складових із проміжними таблицями було прописано запити типу ALTER TABLE.

* 1. **Опис серверної частини**

Було створено окремі репозиторії для front end, back end та IOS додатку частин. Вони були об’єднані рамками проекту в сервісі для керування версіями - GitHub.

Файлова система була реалізована відповідною до тої, що була описана у другому розділі(див. рисунок 3.1)

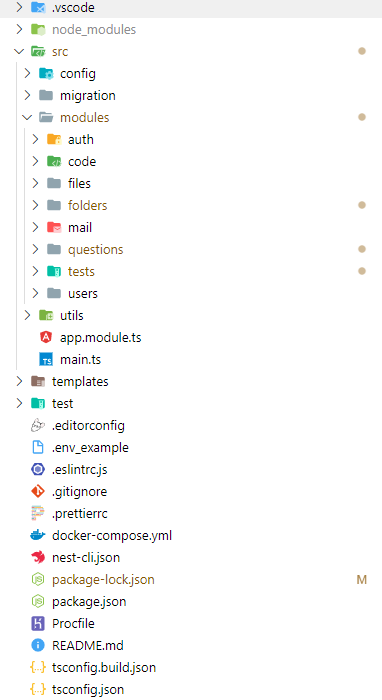


Рисунок 3.1－ Файлова структура back end частини

Для реалізації були використані наступні технології: Nest.js, TypeScript, Node.js, PotgreSQL, JWT. Окрім цього були застосовані засоби розробки: NPM, TSlint, Git, Husky.

Архітектурно, back-end частина складається із мікросервісів, які знаходяться у теці modules: auth, code, files, folders, mail, questions, tests, users. Кожен ізи цих модулів має сервіси, контроллер та моделі у вигляді enum, class та intefaces. Ці моделі також поділяються Оскільки, є код, який повторюється, то його було перенесено у теку utils, яка містить пайпи(pipes). Конвеєр - це клас, котрий коментується декоратором @Injectable (). Pipe повинні реалізовувати інтерфейс PipeTransform.Вони мають два типові випадки використання:

* перетворення: перетворення вхідних даних у бажану форму (наприклад, із рядка в ціле число);
* перевірка: оцініть вхідні дані і, якщо дійсні, просто передає їх без змін; в іншому випадку повертає помилку, коли дані неправильні.[35]

Вони використовуються у всіх модулях, бо кожен модуль має HTTP запити, дані яких мають перевіряються відповідно описаних у попередніх розділах обмежень. Такі перевірки даних відбуваються при запитах HTTP POST та PUT запитах, оскільки такі запити маю тіла із даними, що мають зберігатися чи оновлюватися у базі даних. Для того, щоб сервер не падав від помилок взаємодії із базою даних.

Такі запити описані у контролерах, а методи для взаємодії з даними БД у сервісах. Усі методи супроводжуються типізацією вхідних та результативних даних роботи методу. Вхідні дані - параметри, результативні - те, що повертає функція.

* 1. **Опис клієнтської частини**

Файлова структура front end частини також відповідає тій, що описана у другому розділі. Частково вона була перейнята від бекенд файлової системи і окрім модульної архітектурі, була реалізована з використанням шаблону MVC для роботи із користувацьким інтерфейсом.

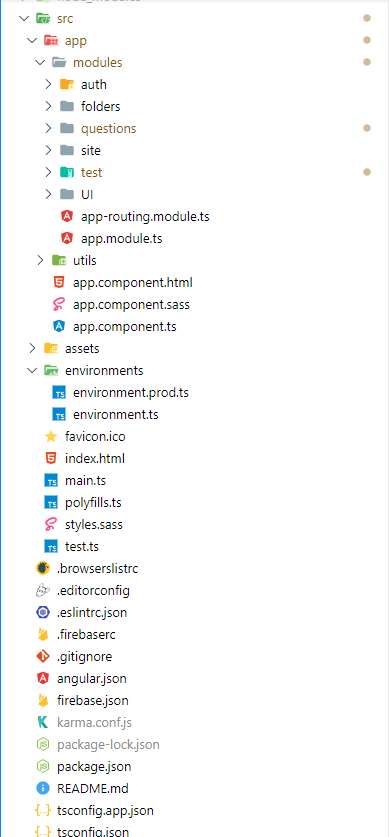


Рисунок 3.2－ Файлова система front end частини

Така файлова система дозволяє масштабувати проект. Згідно описаному шаблону проектування було реалізовано частину системи тестування Quick Test. Для наглядної демонстрації використання шаблону, нижче описаний алгоритм взаємодії елементів компонентів: component-name.html, component-name.ts, component-name.class.ts. component-name.html - view, включає у себе відображення інформації компонента, який у цьому випадку виступає контролером. component-name.ts - контролер, де описана вся бізнес логіка, яка стосується сторінки component-name.html. component-name.class.ts включає у себе клас із полями, які будуть оброблені та/чи показані у компонентах і відображені на сторінках відповідних компонентам.

Так як проект розроблений з використанням фреймворку Angular CLI, то мова програмування там використовується TypeScript і у файлах .html реалізований фукнціонал для відображення даних компонентів за допомогою синтаксису: {{ property-name }}. Компоненти фреймворку, згідно шаблону програмування, є контролером. Для того щоб такий синтаксис спрацював у файлі компонента використовується директива @Component.

Схематично взаємодія складових програми виглядає так:

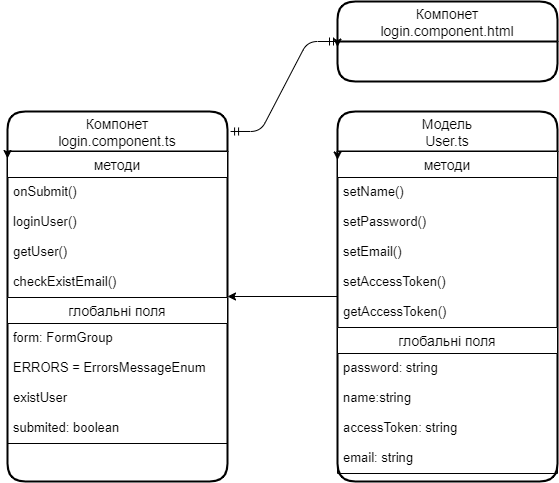


Рисунок 3.3－Схема взаємодії складових програми

Реалізована модуль користувача являє собою клас із полями password, name, accessToken та email. Для кожного поля був прописаний метод get та set. Перший записує у полі дані, а другий витягує:

export class User {

password: string;

name: string;

accessToken?: string;

email: string;

public setName(name: string): void {

this.name = name;

}

public setPassword(password: string): void {

this.password = password;

}

public setEmail(email: string): void {

this.email = email;

}

public setAccessToken(accessToken: string): void {

this.accessToken = accessToken;

}

public getAccessToken(): string {

return this.accessToken;

}

}

Модель використовується у контролері, але так як в Angular CLI ця роль поділена між компонентом і сервісом, то було реалізовано LoginComponent та AuthService. Згідно описаним вимогам до архітектури із другого розділу, компонент відповідає за взаємодію із UI елементами та подіями з клієнту(браузеру), у той час як сервіс включає у себе HTTP запити на серверну частину системи.

Демонстративний код LoginComponent:

@Component({

selector: 'app-login',

templateUrl: './login.component.html',

styleUrls: []

})

export class LoginComponent {

form: FormGroup;

ERRORS = ErrorsMessages;

existEmail: boolean;

existUser: boolean;

submited: boolean;

constructor(

private formBuilder: FormBuilder,

private authService: AuthService,

private router: Router) {

this.form = this.formBuilder.group({

email: ['', [

Validators.required,

Validators.pattern(Patterns.EMAIL\_PATTERN)]

],

password: ['', [

Validators.required,

Validators.minLength(8),

Validators.maxLength(20),

Validators.pattern(Patterns.PASSWORD\_PATTERN)]

]

});

}

onSubmit(): void {

this.submited = true;

const email = this.form.controls.email.value;

this.checkExistingEmail(email);

}

loginUser(user: User): void {

setTimeout(() => {

this.authService.login(user)

.subscribe(

(res) => {

this.existUser = true;

user.setAccessToken(res.accessToken);

this.authService.setLoginStatus(true);

localStorage.setItem('token', res.accessToken);

console.log(res.accessToken);

this.router.navigate(['/home/', ROOT\_FOLDER\_NAME]);

},

(error) => +error.status === 400 ? this.existUser = false : console.error('error')

);

}, 1000);

}

getUser(): User {

const user = new User();

user.setEmail(this.form.controls.email.value);

user.setPassword(this.form.controls.password.value);

return user;

}

checkExistingEmail(email: string): void {

setTimeout(() => {

this.authService.checkEmail(email)

.subscribe(

() => this.existEmail = false,

(error) => {

+error.status === 400 ? this.existEmail = true : console.log(error);

// пошта існує в базі, вхід доступний

const user = this.getUser();

this.loginUser(user);

},

);

}, 1000);

}

}

Параметри у конструкторі створюють посилання на об’єкт використаних модулів та їх складових, серед них:

* router - для взаємодії із вбудованим в Angular маршрутизатором;
* authService - для взаємодії із сервісом, де описані усі запити для реєстрації, входу в обліковий запис та перевірки існування користувача у БД;
* formBuilder - необхідне для реалізації форми на базі реактивної форми фреймворку.

У тілі конструктора описано реалізацію очкуваних полів форми із обмеженням(Validators), де згідно назвам параметрам реалізовується перевірка даних. Дані підключаються вбудованими атрибутами: formGroup, formControlName, ngSubmit. Перший підключає поле form до форми в html. Другий підключає поля форми до полів вводу. Третій викликає метод, який має спрацювати при події submit форму.

Після натиснення на кнопку типу submit викликається метод onSubmit, який змінює змінну submited форми і викликає метод перевірки існування електронної пошти в базі(checkExistingEmail). Метод нічого не повертає, містить у собі алгоритм, де у JavaScript функції setTimeout викликається метод checkEmail сервісу, де із затримкою в одну секунду в поле existEmail класу компонента записуються дані типу boolean. Якщо поле рівне true, то це означає, що пошта існує і допускаються наступні дії перевірки даних. Якщо рівне false, то це означає, що пошти у БД не має і як наслідок, користувача не існує, повідомлення про даний результат відображається користувачу на екрані форми.

Наступні перевірка включає у себе перевірку правильності введених даних до тих, що вже є. Тим самим відбувається перевірка коректності введеного пароля на відповідність введеного пароля до того, що збережений. Якщо паролі не рівні, то відображається повідомлення “Невірні дані”. Усі повідомлення до форм збережені в enum, де кожна назва оформлена як константа.

Код сервісу:

@Injectable({

providedIn: 'root'

})

export class AuthService {

AUTH\_API\_URL = `${environment.api}/auth`;

loginStatus = false;

constructor(private http: HttpClient) { }

setLoginStatus(status: boolean): void {

this.loginStatus = status;

}

getLoginStatus(): boolean {

return this.loginStatus;

}

checkEmail(email: string): Observable<object> {

return this.http.head(`${this.AUTH\_API\_URL}/email/${email}`);

}

register(user: User): Observable<User> {

return this.http.post<User>(`${this.AUTH\_API\_URL}/registration`, user);

}

login(user: User): Observable<User> {

return this.http.post<User>(`${this.AUTH\_API\_URL}/login`, user);

}

}

Даний код містить у собі HTTP запити на сервер для перевірки існування пошти в базі, методи входу та реєстрації користувача. Окрім цього є метод для отримання статусу авторизації, який необхідний для перевірки доступу до сторінок панелі ментора.

View - має назву login.component.html. Для підключення форми, до тегу form було додано атрибут обрамлений квадратними дужками, такий ситаксис показує фреймворку, що цей тег підключається до поля form. Щоб спрацювала реактивність при натисненні на кнопку “Увійти”, до цього ж тегу form у круглих дужки виклик методу onSubmit. Реактивність дозволяє обробляти дані без оновлення сторінки. Поля вводу мають атрибут без дужок formControlName. Для показу помилок був створений компонент модуля UI - ErrorMessage. Цей компонент приймає message, яке може бути тільки типу string. Щоб використати компонент використовується селектор app-error-message. Для керування відображенням використовується директива \*ngIf, яка виконує ту саму функцію, що й умовний оператор TypeScript тільки в середині .html.

Код компоненту форми входу:

<form class="form" [formGroup]="form" (ngSubmit)="onSubmit()">

<label class="label" for="email">E-mail \*</label>

<input formControlName="email"

[ngClass]="{'input\_text--error': this.form.controls.email.errors

&& this.submited === true}"

class="input\_text"

id="email"

type="email"

name="email"

placeholder="pavloivanovich@mail.com">

<div \*ngIf="this.submited === true && this.form.controls.email.status === 'INVALID'" class="form\_errors-c">

<app-error-message

\*ngIf="this.form.controls.email.errors.required"

message="{{ ERRORS.ERROR\_MESSAGE\_FOR\_REQUIRED\_FIELD }}"></app-error-message>

<app-error-message

\*ngIf="this.form.controls.email.errors.pattern"

message="{{ ERRORS.ERROR\_MESSAGE\_FOR\_INVALID\_EMAIL }}"></app-error-message>

</div>

<label class="label" for="password">Пароль \*</label>

<input formControlName="password"

[ngClass]="{'input\_text--error': this.form.controls.password.status === 'INVALID' && this.submited === true}"

id="password"

class="input\_text"

type="password"

placeholder="Пароль">

<div \*ngIf="this.submited === true && this.form.controls.password.status === 'INVALID'"

class="form\_errors-c">

<app-error-message

\*ngIf="this.form.controls.password.errors.required"

message="{{ ERRORS.ERROR\_MESSAGE\_FOR\_REQUIRED\_FIELD }}"></app-error-message>

<app-error-message

\*ngIf="this.form.controls.password.errors.pattern"

message="{{ ERRORS.ERROR\_MASSAGE\_FOR\_INVALID\_PASSWORD }}"></app-error-message>

<app-error-message

\*ngIf="this.form.controls.password.errors.minlength"

message="{{ ERRORS.ERROR\_MESSAGE\_FOR\_SO\_SHOT\_PASSWORD }}"></app-error-message>

<app-error-message

\*ngIf="this.form.controls.password.errors.maxlength"

message="{{ ERRORS.ERROR\_MESSAGE\_FOR\_SO\_LONG\_PASSWORD }}"></app-error-message>

</div>

<div>

<!-- common errors -->

<app-error-message

\*ngIf="this.existEmail === false"

message="{{ ERRORS.ERROR\_MESSAGE\_FOR\_NOT\_EXISTED\_EMAIL }}"></app-error-message>

<app-error-message

\*ngIf="this.existUser === false"

message="{{ ERRORS.ERROR\_MESSAGE\_FOR\_LOGIN\_FORM }}"></app-error-message>

</div>

<button class="btn--submit" type="submit">Увійти</button>

</form>

Для підключення каскадних таблиць стилізації, був реалізований файл form-elements.sass у теці utils/styles. Цей файл використовується у всіх компонентах, які мають форму, оскільки, усі форми мають спільну стилізацію.

Аналогічно цьому алгоритму, було реалізовано модулі теки, тесту та запитання.

* 1. **Алгоритм роботи фактичної системи**

При першому вході на сайт системи швидкого тестування коритсувач потрапляє на головну сторінку офіційного сайту.(див. рисунок 3.4)

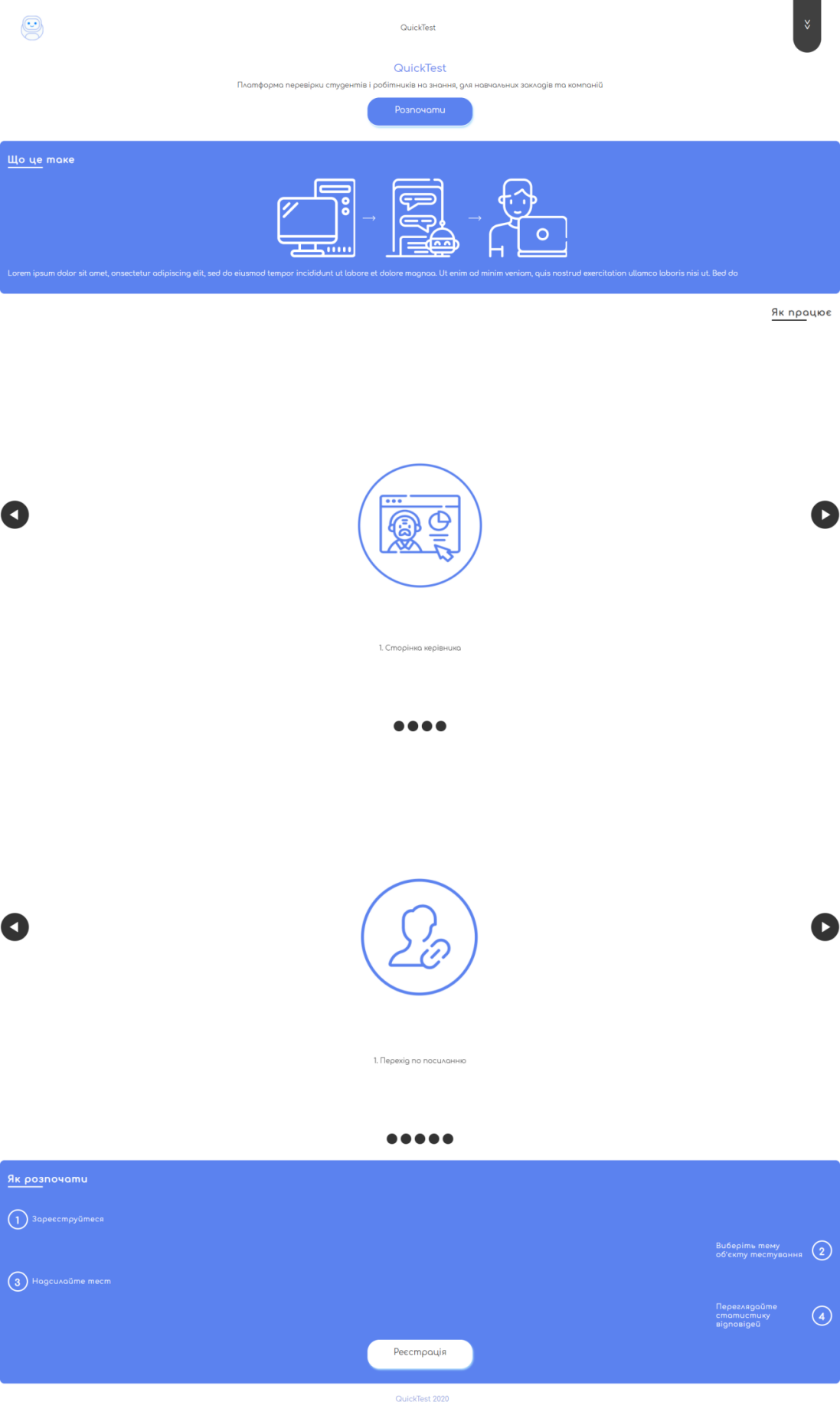


Рисунок 3.4－ Реалізована головна сторінка

Далі користувач можу перейти на сторінку допомоги, де він може більш детально ознайомитися із функціоналом системи(див. рисунок 3.5)



Рисунок 3.5 － Реалізована сторінка допомоги

У будь-якому випадку, для доступу до описаних можливостей, користувачу в статусі гості потрібно увійти чи зареєструвати новий обліковий запис. Тому він має натиснути сіру кнопку у правому верхньому кутку. По натисненню відкривається сторінка входу(див. рисунок 3.6), де можна увійти в обліковий запис, якщо дані були введені відповідно до вимог, і користувач є в системі, то його дані для доступу до сторінок панелі ментора та відбудуться перехід на головну сторінку панелі. Якщо дані було введено не коректно, то буде відображено повідомлення про помилку відповідно до типу помилки та його сенсу.(див. рисунок 3.7)

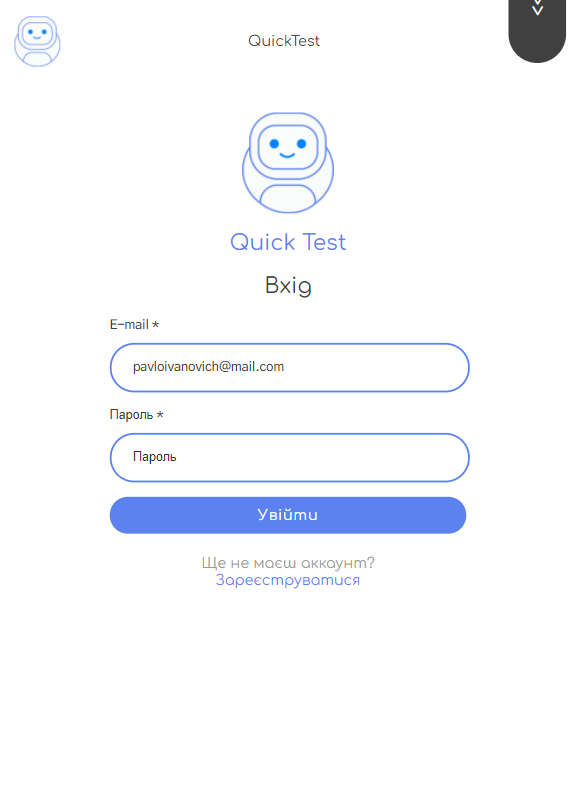


Рисунок 3.6 － Сторінка входу



Рисунок 3.7 － Реалізована сторінка реєстрації

Якщо користувач не має облікового запису, то може перейти на відповідну сторінку за писиланням, що знаходиться внизу. Тоді він опиниться на сторінці з формою створення акаунта.(див. рисунок 3.8) Там йому необхідно ввести ім’я формату “Прізвище Ім’я”, електронну пошту та двічі ввести пароль.

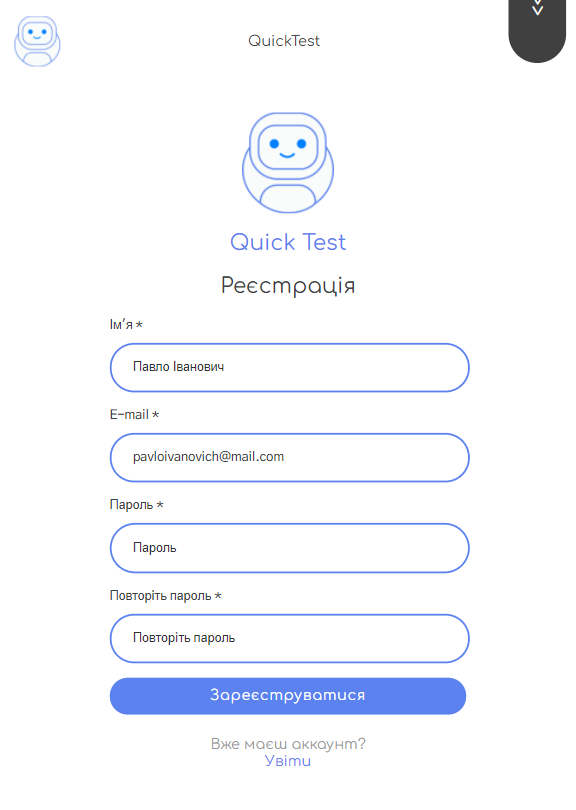


Рисунок 3.8 − Реалізована форма реєстрації

Якщо дані валідні, то відбудеться запит на серверну частину системи, де відбудеться валідація даних, дані збережуться у базі даних і на клієнт повернеться токен доступу.

Якщо дані не валідні, то під кожним полем, яке має проблеми із введеними даними, буде відображено повідомлення про помилку і рама поля змінить колір на червоний.(див. рисунок 3.9)

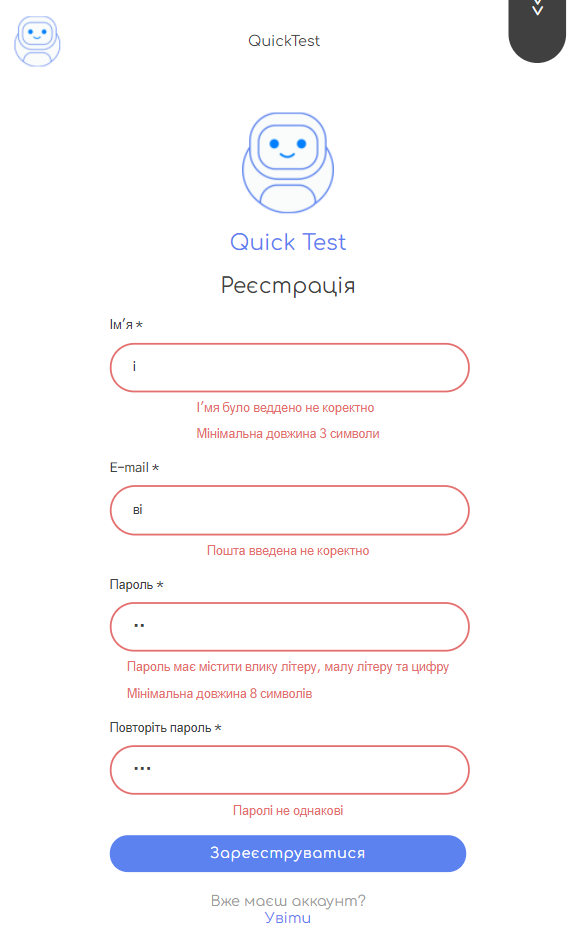


Рисунок 3.9－ Реалізовані сповіщення про помилки при спробі зареєструватися з некоректними даними

Після отримання токену, тим самим заміни статусу користувача на авторизованого, користувач потрапляє на головну сторінку панелі ментора. Вона включає у себе переліком файлів і посиланнями для створення сутностей тест, тека і запитання.(див. рисунок 3.10)

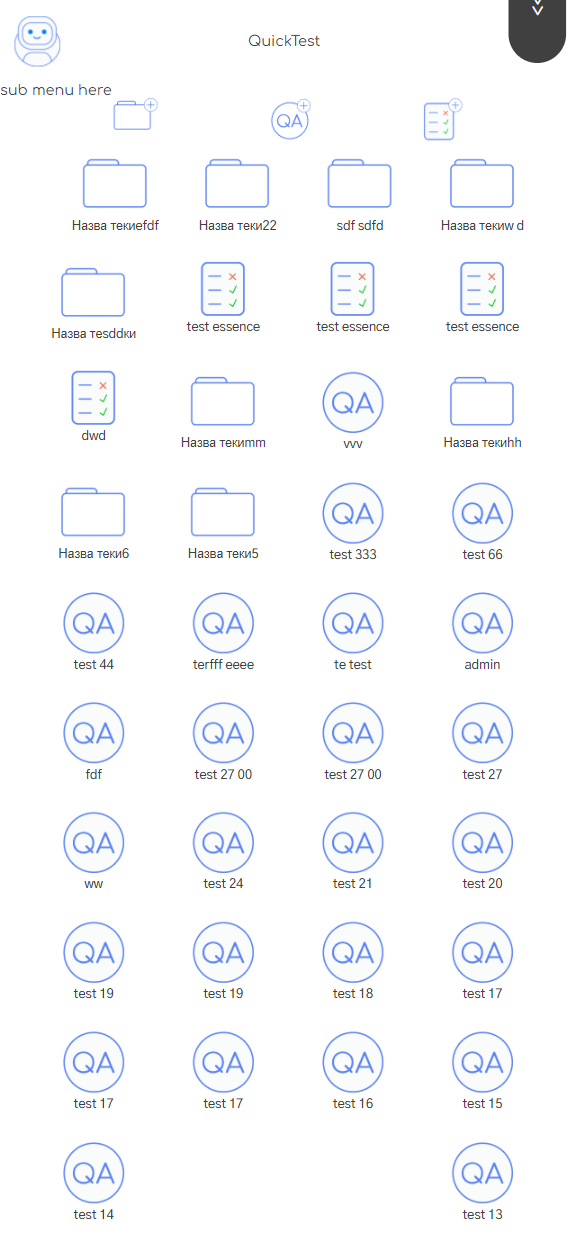


Рисунок 3.10 - Реалізована головна сторінка панелі ментора

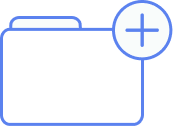


Рисунок 3.11 − Іконка посилання на форму для створення теки в середині поточної

Для створення теки необхідно натиснути на іконку як на рисунку 3.11. Форма для створення теки відображається на цій самій сторінці на місці як на рисунку 3.12.

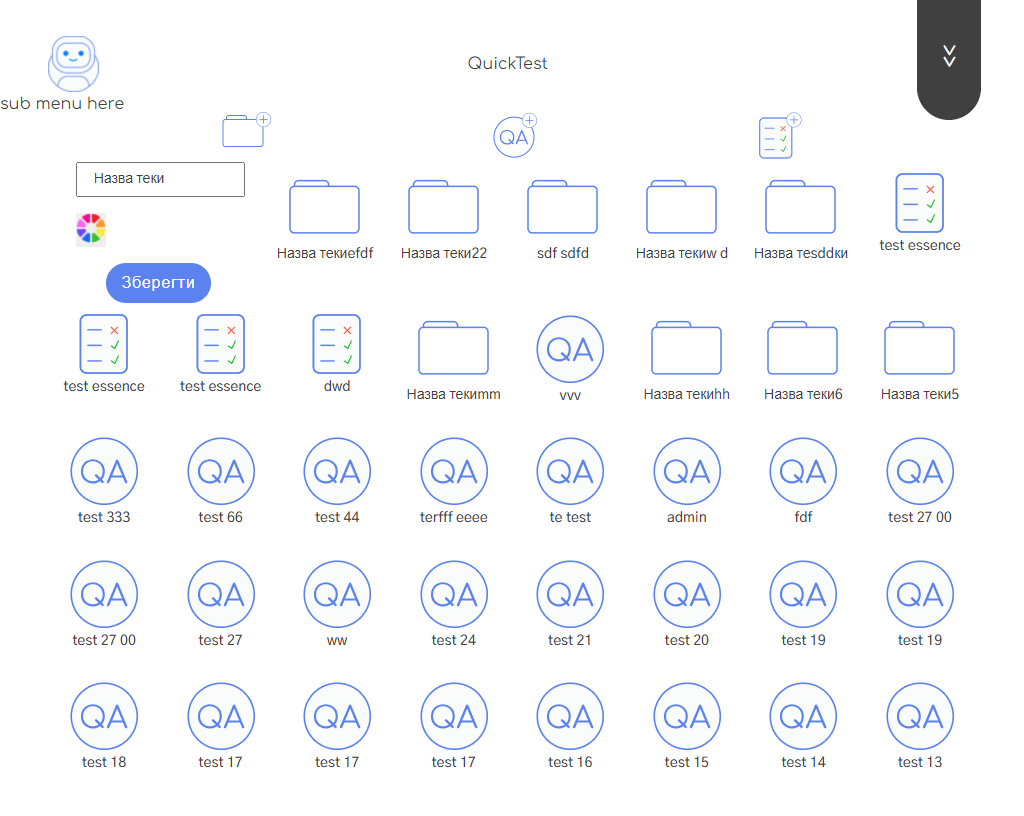


Рисунок 3.12 － Реалізована головна сторінка панелі ментора

Після натиснення на кнопку з вибором кольору, поряд з’являється перелік кольорів.(див. рисунок 3.13) Якщо поле з назвою буде пустим і користувач натисне на кнопку “Зберегти”, то форма закриється і дані не будуть збереженими.

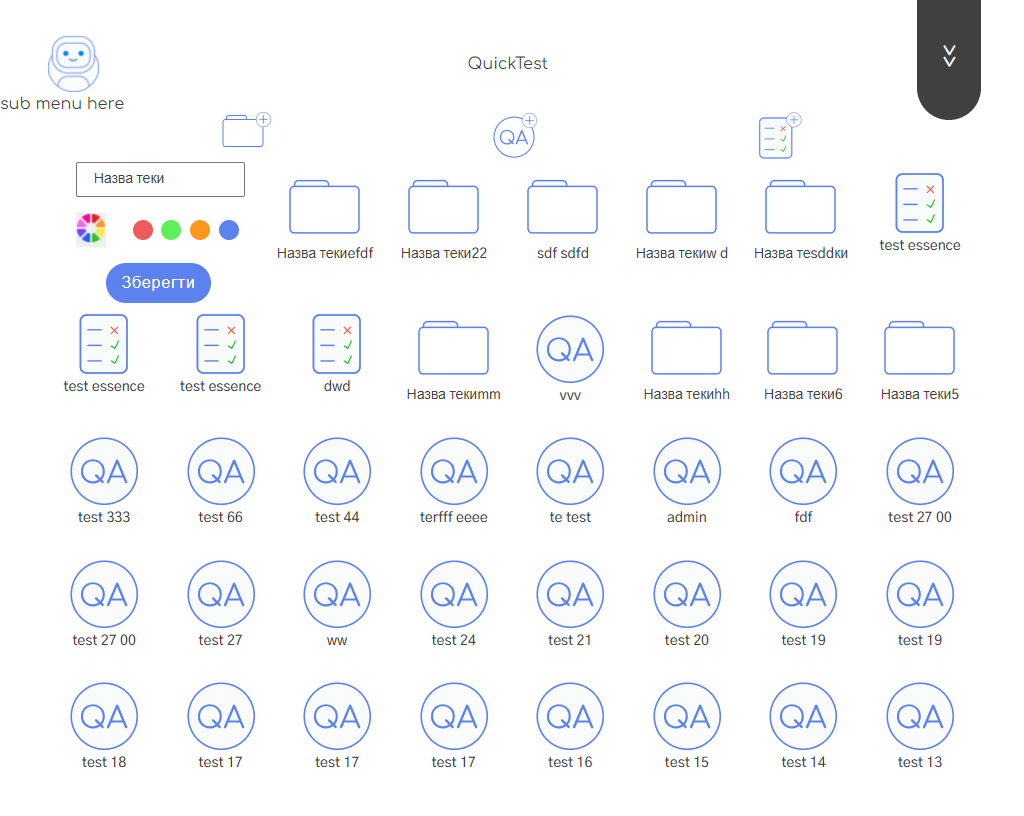


Рисунок 3.13 － Реалізована сторінка панелі ментора із формою створення теки та вибором кольору

Після користувач може натиснути на кнопку створення запитання і тоді він перейде на сторінку із формою створення. Вона включає у себе поле для вводу назви запитання, велике текстове поле для вводу тексту запитання, два випадні списки: для вибору шаблону відповіді та для вибору часу на надання відповіді; також є кнопку для збереження даних.(див. рисунок 3.14)



Рисунок 3.14 － Реалізована сторінка створення запитання

Поки поле із текстом запитання не буде заповнене і користувач не обере шаблон тесту, то запит надісланим не буде. І ці поля будуть підсвічені червоним та з’явиться повідомлення “Поле обов’язкове”.(див. рисунок 3.15)



Рисунок 3.15 － Реалізована форма стоворення запитання із повідомленням про помилку

Після обрання шаблону, залежно від обраного варіанту поля будуть заповнені відповідно до назви. Тобто якщо користувач обирає шаблон “так/ні”, то на екрані над кнопкною “Зберегти” з’явиться два поля із заповненими полями, де перше міститиме текст “так”, а друге “ні”.(див. рисунок 3.16)



Рисунок 3.16 - Реалізована форма стоврення запитання із обраним шаблоном “так/ні”

У процесі створення запитання шаблон можна змінювати, і навіть якщо буде обрано спочатку “Власний” шаблон, а потім інший, то дані будуть замінені і кількість полів варіантів відповіді буде рівна двом. Якщо при попередньому шаблону створити варіантів більше, ніж 2, то напроти поля для вводу тексту з’явиться кнопка видалення у вигляді картинка повернутого хреста.(див. рисунок 3.17)

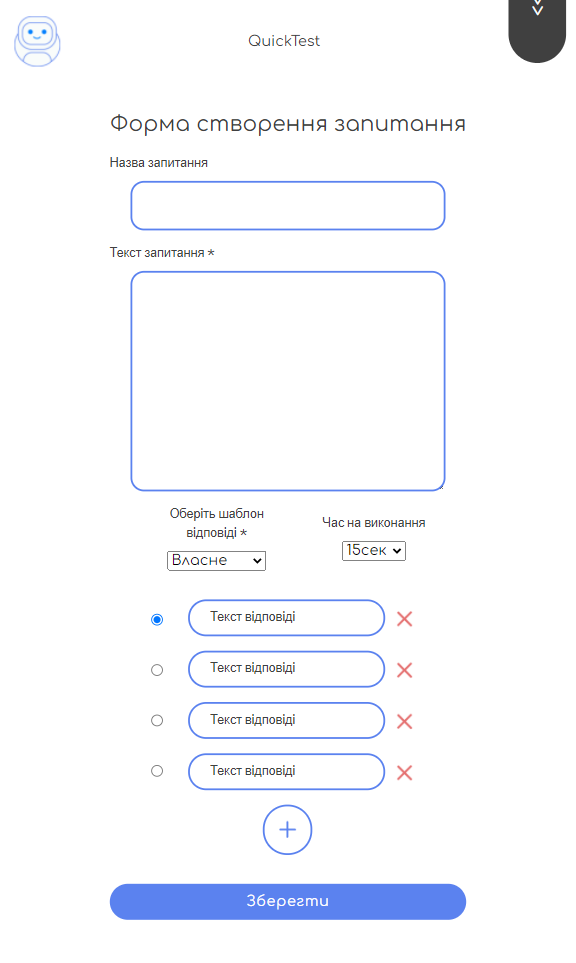
****

Рисунок 3.17 - Реалізована форма створення запитання із обраним шаблоном “Власне” із 4ма варіантами

Який би шаблон не був обраний, користувач може обрати той варіант відповіді, який має бути правильним, за допомогою поля вводу типу radio.

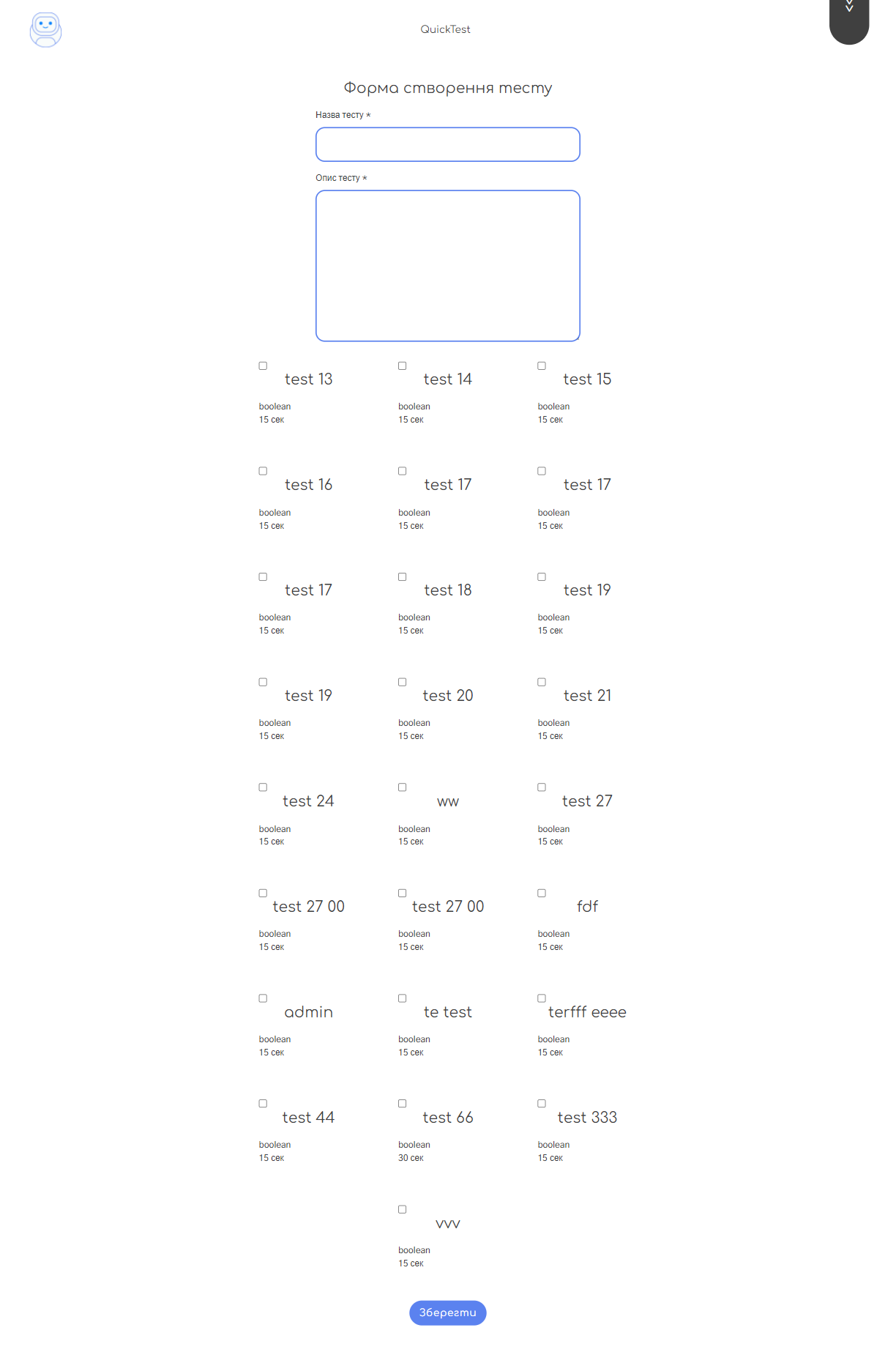


Рисунок 3.18 - Реалізована форма створення тесту

Якщо користувач ввів усі дані коректно і натиснув кнопку “Зберегти”, то надсилається запит на серверну частину після чого відбувається автоматичний перехід на головну сторінку панелі ментора.

Далі користувач має перейти на сторінку створення тесту, натиснувши на відповідне посилання. Через обмеження у часі, форма реалізації тесту має поля для вводу назви тесту, його опису і список карток запитань, які можна обрати для додавання їх до тесту.(див. рисунок 3.18)

Поля для вводу опису тесту є обов’язковим. Також не коректним буде вважатися спроба створити тест без запитань. Помилки будуть показані аналогічно формі створення запитання.

Якщо дані було введено, то відбувається надсилання даних на серверну частину проекту та перехід на головну сторінку.

Вся інша логіка, пов’язана із редагуванням, видаленням переглядом вмісту сутностей, убла реалізована без реалізації відображення у вигляді html сторінок. Тестування цих тригерів відбувається за допомогою допоміжного ПЗ - Swagger.(див. рисунок 3.19)

Swagger дозволяє описати структуру API, щоб складові системи могли їх читати. Здатність API описувати власну структуру - корінь Swagger та протестувати запити.[36]

Тестування чат боту також включено у тестування запитів, проте до нього доступ має тільки розробник серверної частини.

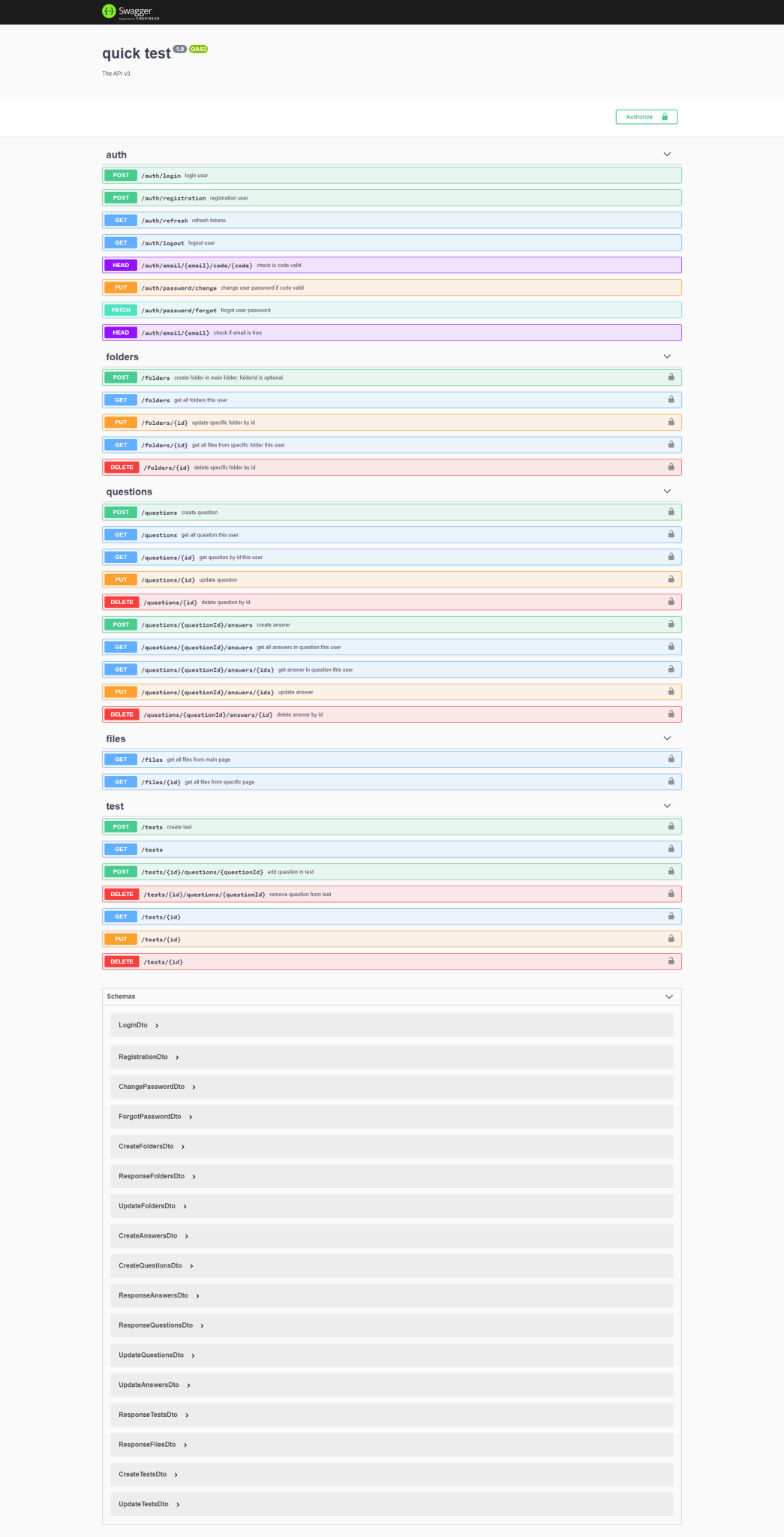


Рисунок 3.19 - Результат тестування тригерів

**Висновок третього розділу**

Таким чином було реалізовано базу даних мовою SQL, а саме на базі PostgreSQL, серверну та клієнтську частини. Кожна із них включає у себе функціонал для авторизації та автентифікації ментора. Після чого він переходить на головну панель, де знаходяться файли запитань, тестув та інших тек, створені ним. Далі ментор може взаєодіяти із цими елементами відповідно до опису в техінчному завданні, що правда не з усіма на пряму через браузер.

У ході розробки було виокритсано паттерн мікросерсвісний програмування в серверній частині та MVC із мікросервіснами в клієнтській. Парша частина реалізовна за жопомогою технілогій Nest.js, TypeScript, Node.js, PostgreSQL, JWT. Друга частина(клієнтська) з використанням AngularCLI, TypeScript, Webpack, Sass. Для підтримки коду у єдиному стилі, було використано наступні засоби розробки: TSlint, Husky. Для організації роботи із пакетами цих бібліотек використовувався NPM, а для організації роботи у команді над спільними теками - Git.

Було реалізовано стоірнки офійіного сацту, входу та реєстрації, головної стоірнки панелі ментора, форми створення тесту, запитання та теки згідно вимогам ТЗ та дизайну із певними лопущеннями, де це було можливо.

Весь інший фукціонал не має представлення із UI складовою, і всі запити тестуються за допомогою Swagger. Доступ до чат боту заблокований, тому описати його роботу не було можливим.

ВИСНОВКИ

Отже, було реалізовано систему швидкого тестування. Першими кроками до реалізації були: аналіз ринку, формування вимог до системи та вибір засобів реалізації. На основі цих даних було створено план реалізації проекту, який включав у себе таблицю із детальним описом строків на виконання робіт та обумовлення шаблону проектування Scrum. На основі аналізу аналогів, було описано основні сутності з описом взаємодії користувача над ними. Згідно цьому опису було спроектовано програмні модулі та користувацький інтерфейс з подальшим проектуванням дій системи з точки зору різних користувачів. Далі було розроблено користувацький інтерфейс, щоб зрозуміти які дані необхідно зберігати, окрім тих, що були описані на попередніх етапах. І вже на основі зібраних даних було спроектовано базу даних з описом усіх її таблиць з вказівкою на тип поля та його обмеження. Після було спроектовано архітектурні вимоги, які включають у себе розміщення файлів, файлову систему, представлення як елементи мають взаємодіяти між собою та опис паттернів програмування, які мають бути реалізовані.

Наступним етапом була саме розробка. Так як на попередніх етапах усі неточності були описані, то реалізована система відповідає вимогам. У процесі розробки було використано шаблон проектування Scrum, де на кожному етапі реалізації нового модуля було циклічно. Спочатку було реалізовано модуль авторизації, після теки, запитання, тесту. Кожен із них так чи інакше реалізований не на достатньому рівні, щоб можна було показати це на екрані.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Антикризове управління підприємством (2005) - Бібліотека економіста. [Електронний ресурс]. - Режим доступу: <https://library.if.ua/book/60/4207.html>.
2. Властивості систем - Інформаційний портал stud.com.ua [Електронний ресурс]. - Режим доступу: <https://stud.com.ua/35378/filosofiya/vlastivosti_sistem>
3. Використання платформи «Kahoot!» для дистанційного навчання - Стаття поталу osvita.ua. [Електронний ресурс]. - Режим доступу: <http://osvita.ua/vnz/high_school/73080/>
4. Quizizz - Центр додаткового матеріалу Снейл [Електронний ресурс]. - Режим доступу: <https://www.it-pedagog.ru/quizizz>
5. Переклад інформації про платформу Quizizz - Офіційний сайт Quizizz [Електронний ресурс]. - Режим доступу: <https://quizizz.com/about>
6. FAQ - Офіційний сайт сервісу ClassMarker. [Електронний ресурс]. - Режим доступу: <https://www.classmarker.com/online-testing/faq/>
7. Chatbot: What is a Chatbot? Why are Chatbots Important? -Інформаційний портал expert.ai. [Електронний ресурс]. - Режим доступу: <https://www.expert.ai/blog/chatbot/>
8. CLI Overview and Command Reference - Офіційна документація фреймворку Angular CLI. [Електронний ресурс]. - Режим доступу: <https://angular.io/cli>
9. What is TypeScript? - Офіційний сайт Typescript. [Електронний ресурс]. - Режим доступу: <https://www.typescriptlang.org/>
10. Hello, nest! - Офіційний сайт фреймворку NestJS. [Електронний ресурс]. - Режим доступу: [https://nestjs.org/](https://nextjs.org/)
11. What is PostgreSQL? - Інформаційний портал Amazon. [Електронний ресурс]. - Режим доступу: <https://aws.amazon.com/rds/postgresql/what-is-postgresql/>
12. WebSockets for fun and profit - Стаття блогу компанії Stackoverflow. [Електронний ресурс]. - Режим доступу: <https://stackoverflow.blog/2019/12/18/websockets-for-fun-and-profit/>
13. Documentation - Офіційна документація препроцесору Sass. [Електронний ресурс]. - Режим доступу: <https://sass-lang.com/documentation>
14. What Is Figma? - Навчальний портал elevantoelements. [Електронний ресурс]. - Режим доступу: <https://webdesign.tutsplus.com/articles/what-is-figma--cms-32272>
15. ESLint Find and fix problems in your JavaScript code - Офіційна документація ESLint. [Електронний ресурс]. - Режим доступу: <https://eslint.org/>
16. TSLint An extensible linter for the TypeScript language. - Офіційна документація TSLint. [Електронний ресурс]. - Режим доступу: <https://palantir.github.io/tslint/>
17. Системи контролю версій на прикладі git - Сайт Державного університету телекомунікацій. [Електронний ресурс]. - Режим доступу: <http://www.dut.edu.ua/ua/news-1-626-9170-sistemi-kontrolyu-versiy-na-prikladi-git_kafedra-kompyuternih-nauk-ta-informaciynih-tehnologiy>
18. Husky - Сайт налаштувань конфігурацій npm . [Електронний ресурс]. - Режим доступу: <https://www.npmjs.com/package/husky>
19. About npm - Офіційний сайт npm. [Електронний ресурс]. - Режим доступу: <https://www.npmjs.com/about>
20. Головна сторінка офіційного сайту JWT. [Електронний ресурс]. - Режим доступу: <https://jwt.io/>
21. Token - Інформаційний портал WhatIs.com. [Електронний ресурс].
22. What Is Token-Based Authentication? - Інформаційний портал okta. [Електронний ресурс]. - Режим доступу: <https://www.okta.com/identity-101/what-is-token-based-authentication/#:~:text=Auth%20tokens%20work%20like%20a,or%20server%2Dbased%20authentication%20techniques.>- Режим доступу: <https://whatis.techtarget.com/definition/token>
23. How to write technical task - Encomage [Електронний ресурс]. - Режим доступу: <https://encomage.com/2020/04/27/how-to-write-technical-task/>
24. CRUD - [Електронний ресурс]. - Режим доступу: [https://www.sumologic.com/glossary/crud/](https://www.sumologic.com/glossary/crud/%5d)
25. Менторство в компаніях: як і навіщо впроваджувати - Матеріал електронної системи «Безпека». [Електронний ресурс]. - Режим доступу: - <https://bezpeka.isu.net.ua/sites/default/files/pdf/mentorstvo_v_kompaniyakh_yak_i_n-1831-501340.pdf>
26. WHAT IS SCRUM? [Електронний ресурс]. - Режим доступу: <https://www.scrum.org/resources/what-is-scrum>
27. Аутентифікація і авторизація - Повний посібник по Yii 1.1 [Електронний ресурс]. - Режим доступу: <https://www.yiiframework.com/doc/guide/1.1/uk/topics.auth>
28. Краудсорсинг та пітчинг інноваційних проєктів у сучасному бізнесІ - [Друкований ресурс] - Режим перегляду: Випуск 43, 2020р, c360-36
29. Що таке Патерн? - [https://refactoring.guru/uk/design-patterns/what-is-pattern](https://refactoring.guru/uk/design-patterns/what-is-pattern%5d)
30. Overview [Електронний ресурс]. - Режим доступу: [https://docs.nestjs.com/microservices/basics](https://docs.nestjs.com/microservices/basics%5d)
31. Создание объектов передачи данных (DTO) - <https://docs.microsoft.com/ru-ru/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5>
32. Providers [Електронний ресурс]. - Режим доступу: <https://docs.nestjs.com/providers>
33. Overview [Електронний ресурс]. - Режим доступу: <https://docs.nestjs.com/microservices/basics>
34. Design Patterns - MVC Pattern. [Електронний ресурс]. - Режим доступу: <https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm#:~:text=MVC%20Pattern%20stands%20for%20Model,the%20data%20that%20model%20contains.>
35. Pipes - Документація Nest.JS. [Електронний ресурс]. - Режим доступу: <https://docs.nestjs.com/pipes>
36. What Is Swagger? - Документація Swagger. [Електронний ресурс]. - Режим доступу: https://swagger.io/docs/specification/2-0/what-is-swagger/