Галицький коледж імені В’ячеслава Чорновола

відділення комп’ютерних та видавничих технологій

циклова комісія інформатики та комп’ютерних дисциплін

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач відділенням комп’ютерних та видавничих технологій

Чубей О.О. /_____/
підпис

«___» _____ 2020

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проєкту

освітньо-кваліфікаційного рівня «молодший спеціаліст»

зі спеціальності 122 «Комп’ютерні науки та інформаційні технології»

на тему: «Інформаційна система тестування знань учнів з метою підготовки до ЗНО»

Студент групи К-47 Безруков О.О. _____
(підпис)

Керівник проєкту Павлюс В.П. _____
(підпис)

Консультанти:

з техніко-економічного обґрунтування Меленчук Л.І. _____
(підпис)

нормоконтролер Кульчинська Н.З. _____
(підпис)

Тернопіль – 2020

Галицький коледж імені В’ячеслава Чорновола
відділення комп’ютерних та видавничих технологій
циклова комісія інформатики та комп’ютерних дисциплін

ЗАТВЕРДЖУЮ
Завідувач відділенням комп’ютерних та видавничих технологій
Чубей О.О. /_____/
підпис
«___» _____ 2020 р.

ЗАВДАННЯ

на дипломне проєктування
на здобуття освітньо-кваліфікаційного рівня «молодший спеціаліст»

студенту _____
(прізвище, ім’я та по-батькові студента)

1. Тема проєкту _____
_____
_____
_____

затверджена наказом по коледжу від “___”_____2019 р., №_____

2. Термін здачі студентом завершеного проєкту “___” _____2020 р.
3. Вихідні дані до проєкту _____
_____
_____
_____

4. Перелік питань, які повинні бути розроблені в проєкті:
а) основна частина _____
_____
_____
_____

б) техніко-економічне обґрунтування _____
_____
_____
_____
_____

5. Перелік графічного матеріалу _____
_____
_____

6. Консультанти проєкту: _____

Розділ	Консультанти	Підпис, дата	
		Завдання видано	Завдання прийнято
з техніко-економічного обґрунтування	_____ _____ (вчена ступень, звання П.І.Б. _____ _____ консультанта)		

КАЛЕНДАРНИЙ ПЛАН
дипломного проєктування

п/п	Найменування етапу	Терміни	
		початку	завершення

7. Дата видачі завдання "___" _____ 2019 р.
Керівник _____/

Завдання прийняв до виконання _____/

Реферат

Дипломний проєкт. Тема: «Інформаційна система з тестування знань учнів з метою підготовки до ЗНО» містить сторінок - 71, рисунків - 41, таблиць - 2, джерел - 5, додатків - 0, блок-схеми - 2.

Метою проєкту є дослідження теоретичних та практичних основ сервісів для тестування з різних предметів.

Об’єктом дослідження для подальшої розробки є застосунки для підготовки та тестування учнів.

Головним завданням є вибір необхідних технологій, розробка архітектури та бізнес логіки сервісу для тестування.

Результатом дипломного проєктування став повноцінний сервіс для проходження тестів з метою визначення знань учнів з подальшою підготовкою до ЗНО.

Для коректної розробки було використано систему контролю версій git для зручності реалізації, яка надає можливість зберігати усі зміни та при необхідності повертати до попереднього стану. Середовищем розробки стало Visual Studio Code, яке містить головні інструменти для зручного проєктування та розробки ПЗ:

- Панель інструментів.
- Файловий провідник.
- Редактор коду.
- Консоль.

Під час розробки програмного застосунку було використано мову програмування TypeScript, яка розширює можливості JavaScript завдяки статичній типізації.

TYPESCRIPT, VSCODE IDE, СИСТЕМА, КОНТРОЛЬ ЯКОСТІ, ТЕСТУВАННЯ ЗНАНЬ, ДЕРЖАВНІ ЕКЗАМЕНИ, ЗНО.

Abstract

The diploma project topic: "information system for testing students knowledge with the help of external evaluation" contains a 71 pages, 41 figures, 2 tables, 5 sources, 0 applications, 2 flowcharts.

The purpose of project is studying the theoretical and practical basic services for testing various subjects.

The object of research for extended mailings, it is kept for teaching and learning students.

The main task is the selection of the necessary technologies, development of architecture and business logos for testing.

The results of the diploma design provide a dedicated service for the promotion of those who use the knowledge of students with external evaluation.

For correct mailings, a system was used that reliably uses git for ease of use, which allows you to save all changes and, if necessary, return to the previous state. Color boxes directly from visual studio code contain the main tools for quickly creating and sending software:

- toolbar.
- file explorer.
- code editor.
- console.

During the development of the program used, the typescript program was used, which extends javascript, which is a statistical typing.

TYPE SCRIPT, VSCODE IDE, SYSTEM, QUALITY CONTROL, KNOWLEDGE TESTING, STATE EXAMS, EXAMINATION.

ЗМІСТ

Вступ 7
1 Аналіз Існуючих Рішень Та Постановка Завдання 8
1.1 Обґрунтування доцільності створення системи 8
1.2 Огляд існуючих рішень 11
1.3 Постановка задачі 13
2 Проєктування Системи 14
2.1 Формалізація вимог до системи 14
2.2 Проєктування структури системи 16
2.3 Проєктування алгоритму роботи системи 17
3 Реалізація Та Тестування Системи 18
3.1 Поняття, принципи реалізації та тестування систем 18
3.2 Вибір засобів реалізації 20
3.3 Вибір способу захисту користувачів 33
3.4 Реалізація клієнтської частини 41
3.5 Реалізація серверної частини 52
3.6 Тестування 55
4 Техніко-Економічне Обґрунтування 64
4.1 Аналіз ринку 64
4.2 Розрахунок витрат на проєктування 65
4.3 Обґрунтування необхідності розробки 69
Висновки 70
Перелік джерел посилання 71

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – База Даних

ЗНО – Зовнішнє Незалежне Оцінювання

ДПА – Державна Підсумкова Атестація

СКБД – Система Керування Базами Даних

HTML – HyperText Markup Language

JS – JavaScript

TS – TypeScript

CSS – Cascading Style Sheets

API – Application Programming Interface

AJAX – Asynchronous Javascript And XML

XML – Extensible Markup Language

DHTML – Dynamic HTML

DOM – Document Object Model

JSX – JavaScript XML

JSS – JavaScript Style Sheets

HTTP – HyperText Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

SSL – Secure Sockets Layer

TLS – Transport Layer Security

AWS – Amazon Web Services

SWT – Simple Web Token

JWT – JSON Web Token

JSON – JavaScript Object Notation

ВСТУП

Одним із особистих прав кожної людини є право на здобуття вищої освіти. Здійснення цього права не має залежати від певних прав та привілеїв, майнових можливостей родини потенційного здобувача вищої освіти. Держава має створити рівні умови кожному своєму громадянину для її здобуття.

Однією з форм забезпечення доступності до вищої освіти є зовнішнє незалежне оцінювання (далі – ЗНО). В Україні створено чимало можливостей отримання знань для успішної здачі ЗНО, але нами пропонується власний сервіс для тестування та підготовки до ЗНО майбутніх абітурієнтів, учнів шкіл та професійно-технічних закладів, студентів коледжів.

Сервіс, який нами розроблено представляє з себе платформу, яка надає можливість майбутнім абітурієнтам підготуватися до ЗНО. Він буде мати можливість проходження попередніх сесій, як це реалізовано у стандартному застосунку та окрім цього, буде надаватися можливість підготовки з предметів з можливістю вибору теми, а також готових варіантів, які є наближеними до ЗНО. Даний сервіс буде реалізовано у вигляді веб-застосунку.

Метою дипломної роботи є створення сервісу у вигляді веб-застосунку, який оптимізує процес підготовки учнів загальноосвітніх шкіл до здачі ЗНО.

Основними завданнями дипломної роботи є:

- обґрунтування доцільності створення системи;
- проведення аналізу існуючих рішень;
- опрацювання необхідних джерел з наступних фреймворків, платформ та мов програмування: CSS, HTML, TypeScript, Node JS, React та Fastify;
- визначити функціональні вимоги до інформаційної системи;
- здійснити проєктування та розробку веб-застосунку;
- провести техніко-економічне обґрунтування рішення;
- розглянути питання організації умов праці.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Обґрунтування доцільності створення системи

Зовнішнє незалежне оцінювання – одна з найпоширеніших та найефективніших у світі систем оцінювання освітніх досягнень учнів. Вона дає можливість провести як підсумкову атестацію, так і селекцію для закладів вищої освіти. Результати ЗНО є основою для проєктування освітньої стратегії і тактики, оцінки роботи освітніх систем в цілому та закладів освіти зокрема.

У сучасному світі проблема якості освіти набуває особливої актуальності для всіх держав. Суспільства повною мірою усвідомлюють важливість набуття якісної освіти для випереджального розвитку країни.

Основними передумовами запровадження ЗНО в Україні було проголошення Національною доктриною розвитку освіти принципу доступності до якісної освіти для усіх громадян України. Визначення головною метою української системи освіти – створення сприятливих умов для розвитку і самореалізації кожної особистості, формування поколінь, які здатні навчатися впродовж життя, створювати й розвивати цінності громадянського суспільства. Одним із пріоритетів державної політики в розвитку освіти визначається створення однакових можливостей для дітей і молоді у здобутті якісної освіті [1, с. 5].

Сучасне інформаційне суспільство ставить перед освітою глобальну проблему – збільшення кількості та підвищення якості навчальної інформації при інваріантному навчальному часі, протягом якого має бути засвоєна ця інформація.

ЗНО – це дуже важливий етап у житті кожного випускника, адже від нього залежить, у якому закладі вищої освіти він зможе продовжити своє подальше навчання. Успішно здає ЗНО лише той, хто уміє опрацьовувати великі обсяги інформації, запам’ятовуючи при цьому найнеобхідніше і відкидаючи зайве або спірне. Така підготовка завжди вимагає немало часу та сил.

Саме тому більшість підлітків при підготовці до головного екзамену у своєму житті користуються послугами репетиторів. Але таке «задоволення» не з дешевих: ціна одного заняття з репетитором, яке, як правило, триває одну-дві години, може сягати 200 гривень (в залежності від кваліфікації та досвіду самого репетитора). Такі індивідуальні заняття можуть тривати від одного до двох років. Як наслідок, витрати на репетиторів можуть сягати десятків тисяч гривень. Звичайно, при правильній мотивації учня та допомозі з боку батьків – це будуть не марно витрачені кошти. Але досить часто репетиторство не дає тих результати, яких від нього очікують.

Чи можна якось зменшити витрати і при цьому якісно підготуватися до проходження ЗНО? Це питання хвилює значну кількість учнів та їх батьків у нашій країні. Очевидно, що так. По-перше, якщо учень чи студент добре та наполегливо навчається, то кількість занять з репетитором можна значно зменшити (або ж зовсім не використовувати їх). По-друге, і умовах інформаційно-комунікативних технологій існують різноманітні онлайн-курси, які, як правило, коштують значно менше, але дають можливість отримати таку ж кількість знань, як і репетитор.

Зазвичай, такі онлайн-курси проводяться у формі вебінарів.

Для повного розуміння давайте розглянемо, що ж таке вебінар. Вебінар – спосіб організації зустрічей онлайн, формат проведення семінарів, тренінгів та інших заходів за допомогою Інтернету [2].

Це неологізм, який утворений шляхом поєднання слів «веб» та «семінар». Щоб організувати вебінар необхідно використовувати технології відео-конференції, інтернет-телефонії тощо. Донедавна вебінари були досить поширеними виключно у діловому середовищі. Однак усе частіше вони набувають популярності і у дистанційній освіті. Яскравим прикладом цього може бути онлайн навчання під час карантину COVID-19.

У перші роки появи Інтернету терміном «веб-конференція» часто називали гілку форуму або дошки оголошень. Пізніше термін набув значення спілкування саме в режимі реального часу.

Вебінари можуть бути спільними і включати в себе сеанси голосувань і опитувань, що забезпечує повну взаємодію між аудиторією та ведучим. У деяких випадках ведучий може коментувати інформацію, що відображається на екрані, а слухачі можуть йому відповідати, переважно з використанням аудіо гарнітури.

Проте, потрібно врахувати той факт, що цей варіант підходить більше для мотивованих учнів, які вже мають хороше розуміння дисциплін, з яких вони вирішили здавати екзамени та бажають просто закріпити свої поточні знання чи дізнатися певні нюанси у тому чи іншому напрямі.

Давайте підсумуємо переваги та недоліки використання послуг репетиторів та сервісів для онлайн навчання.

Переваги репетитора:

- виділяє весь час на одного учня;
- репетитор вже має певні напрацювання і може дати потрібну базу знань;
- є можливість повернутись до уже пройденої теми, якщо учень чогось не зрозумів, або дещо призабув пройдений матеріал.

Недоліки репетитора:

- як правило, висока вартість заняття;
- не завжди легко знайти необхідну людину у цій сфері.

Переваги онлайн сервісів:

- навчання відбувається онлайн, тобто немає необхідності виходити з дому;
- є можливість спілкування з іншими учасниками занять: поділитися власними знаннями чи запитати, те що цікавить;
- значно менша вартість.

Недоліки онлайн сервісів:

- менше часу приділяється на кожного учня;
- є ризик наткнутися на шахраїв;
- вимагає хорошої базової підготовки, а тому підходить не для всіх.

1.2 Огляд існуючих рішень

Один із існуючих сервісів для підготовки до ЗНО – це zno-osvita.ua. Він представляє собою сайт для проходження тестів ЗНО онлайн та підготовки майбутніх абітурієнтів.

Спосіб виконання всіх тестових завдань у запропонованих на сайті тестах максимально наближений до реальних тестів, а форма надання відповіді відповідає виду, що пропонується абітурієнтам у бланку відповідей під час проходження реальних тестів ЗНО.

Після виконання тестових завдань кожного тесту надаються правильні відповіді на всі завдання та розраховується результат у тестових та рейтингових балах, також визначається час витрачений на виконання тесту.

На головній сторінці сайту знаходиться панель вибору дисципліни, з якої учень чи студент бажає пройти ЗНО попередніх років, а також містить короткий опис самого сайту. На рисунку 1.1 зображено головну сторінку сайту.

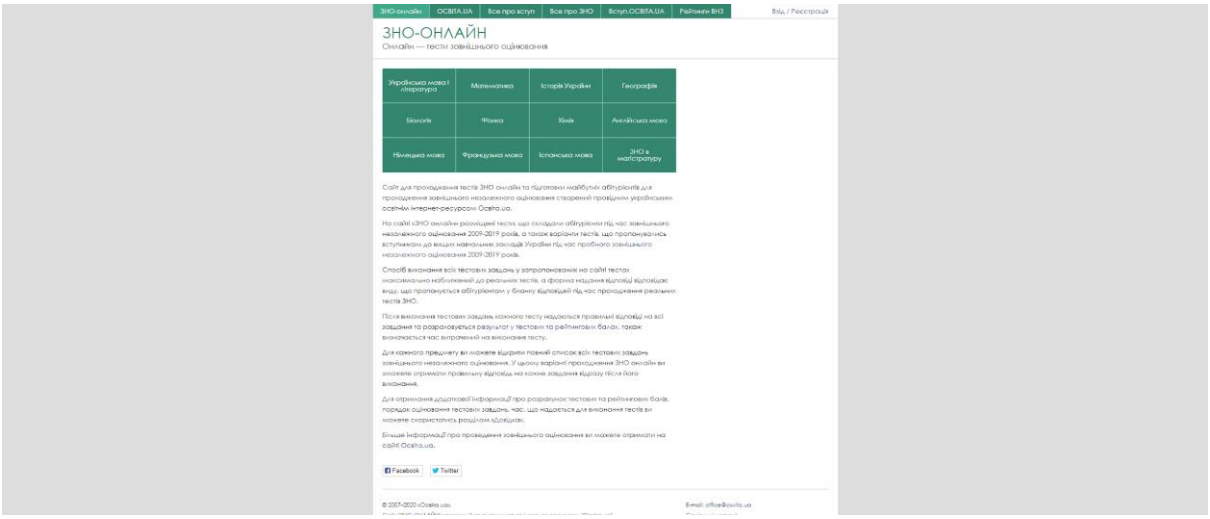

Рисунок 1.1 – Головна сторінка сайту zno-osvita.ua

Після вибору бажаного предмету перед користувачем відкривається можливість вибору сесії ЗНО по роках, а також по типах сесій: основна, додаткова та пробний тест. На рисунку 1.2 зображено сторінку вибору сесії.

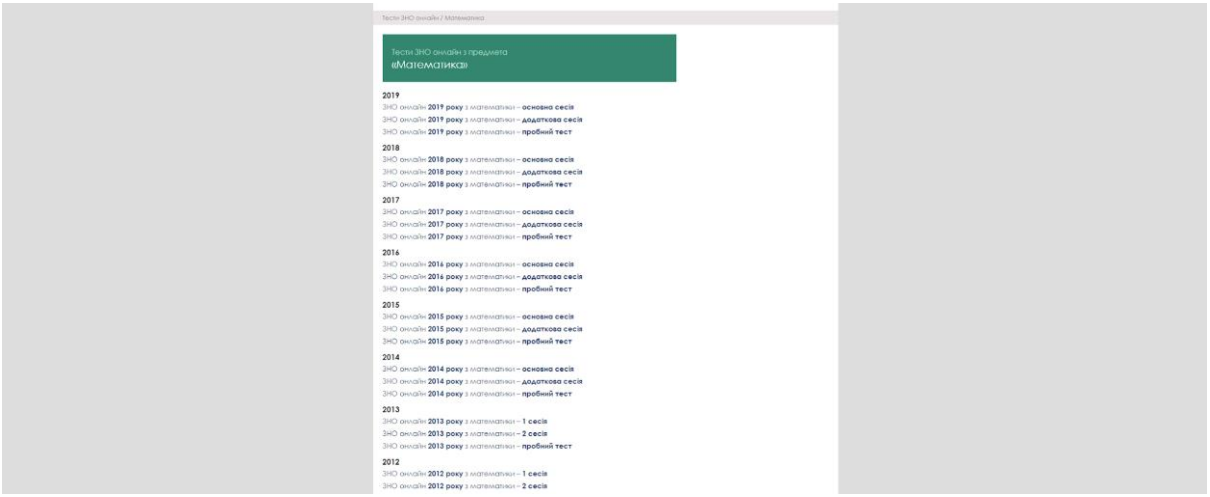

Рисунок 1.2 – Сторінка вибору сесії

Після вибору сесії з’являється користувацький інтерфейс, на якому відображаються головні необхідні елементи для тестування, а саме: умова завдання, панель вибору відповіді, індекс завдань та кнопка «відповісти», яка з’являється тільки після вибору відповіді до завдання. На рисунку 1.3 зображено сторінку тестування.

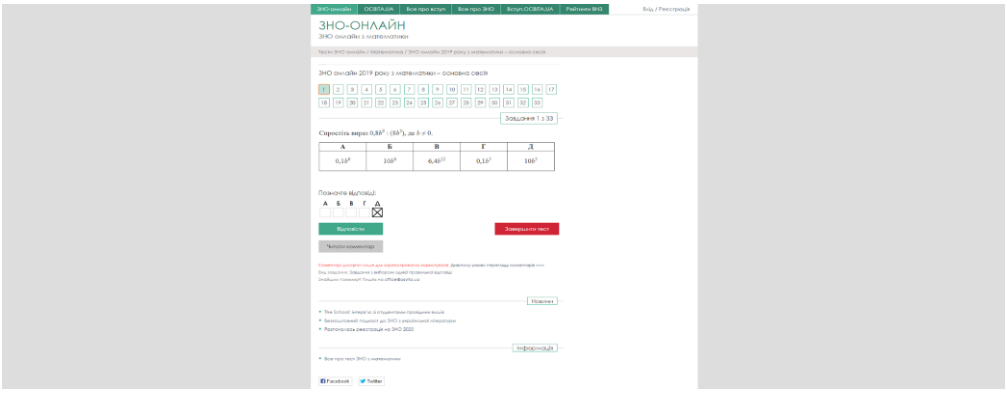

Рисунок 1.3 – Сторінка тестування

1.3 Постановка задачі

Основним завданням даного дипломного проєкту є розборка веб-додатку, який надаватиме можливість майбутнім абітурієнтам проходити попередні сесії ЗНО та буде містити тести для підготовки з різних дисциплін у вигляді оцифрованих підручників для підготовки до ЗНО.

Програмний продукт повинен містити наступний функціонал:

- авторизація та аутентифікація користувачів;
- можливість налаштування бажаного тесту;
- відображення вірних відповідей;
- відображення невірних відповідей;
- шкала прогресу завдань, на які користувач дав відповідь;
- шкала прогресу завдань, на які користувач дав правильну відповідь;
- шкала прогресу завдань, в яких користувач обрав відповідь і затвердив її;
- шкала прогресу набраних балів;
- таймер для тестів;
- можливість створення нових тестів.

Для забезпечення продуктивної роботи веб-сервіс повинен мати адаптивний та інтуїтивно зрозумілий користувацький інтерфейс, а також працювати з різними настільними та мобільними пристроями.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Формалізація вимог до системи

Вимоги до програмного забезпечення (далі – ПЗ) – це сукупність тверджень щодо атрибутів, властивостей або якостей програмної системи, яка підлягає реалізації. Вимоги можуть виражатися у вигляді текстових тверджень і графічних моделей.

У класичному технічному підході сукупність вимог використовується на стадії проєктування ПЗ. Вимоги також використовуються в процесі перевірки ПЗ, оскільки тести ґрунтуються на певних вимогах.

Етапу розробки вимог може передувати техніко-економічне обґрунтування або концептуальна фаза аналізу проєкту. Фаза розробки вимог може бути розбита на виявлення вимог (збір, розуміння, розгляд та з'ясування потреб зацікавлених осіб), аналіз (перевірка цілісності і закінченості), специфікація (документування вимог) і перевірка правильності.

Автоматизована система управління технологічним процесом (далі – АСУ ТП) – це група рішень технічних і програмних засобів, призначених для автоматизації управління технологічним обладнанням на промислових підприємствах. Може мати зв'язок з більш загальною автоматизованою системою управління підприємством (АСУП).

Під АСУ ТП зазвичай розуміється цілісне рішення, що забезпечує автоматизацію основних операцій технологічного процесу на виробництві в цілому або якійсь його ділянці, що виготовляє відносно завершений виріб.

Веб-застосунок – це клієнт-серверний застосунок, у якому клієнт взаємодіє з веб-сервером за допомогою браузера. Логіка веб-застосунка розподілена між сервером і клієнтом, зберігання даних здійснюється, переважно, на сервері, обмін інформацією відбувається по мережі. Однією з переваг такого підходу є той факт, що клієнти не залежать від конкретної операційної системи користувача, тому веб-застосунки є багатоплатформними службами.

Багатоплатформність (кросплатформність, мультиплатформність) – це властивість ПЗ працювати більш ніж на одній програмній (в тому числі операційній системі) або апаратній платформі. Кросплатформність дозволяє суттєво скоротити витрати на розробку нового або адаптацію існуючого ПЗ.

Перш ніж розпочати проєктування системи необхідно визначити вхідні та вихідні дані для функціонування системи.

Вхідними даними системи є:

- дані користувача;
- дані конфігурації бажаного тесту;
- дані вибору відповідей, під час проходження тесту;
- зображення, при створенні нового тесту;
- набір правильних відповідей до завдань, при створенні нового тесту;
- конфігурація тесту, при створенні, для подальшої можливості коректного вибору на стороні користувача.

Вихідними даними системи є:

- результат виконання тесту;
- новий тест, при створенні нового.

Детальна схема функціонування системи зображена на рисунку 2.1.

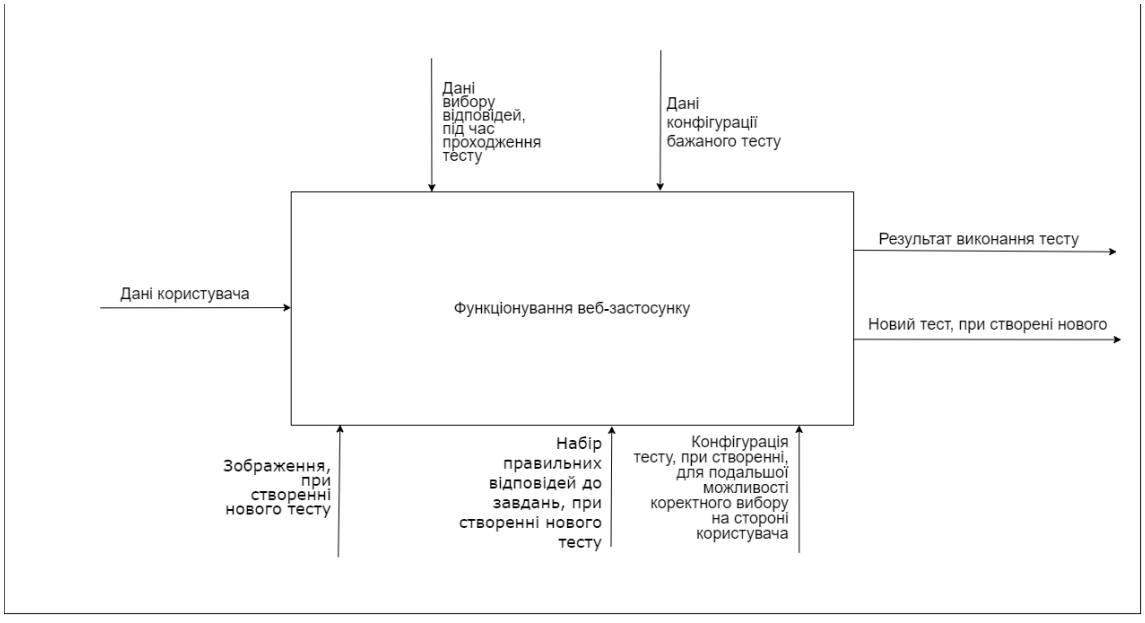

Рисунок 2.1 – Функціональна схема системи

2.2 Проєктування структури системи

Проєктування – це процес визначення архітектури, компонентів, інтерфейсів та інших характеристик системи або її частини. Результатом проєктування є проєкт – цілісна сукупність моделей, властивостей або характеристик, описаних у формі, придатній для реалізації системи.

Проєктування, поряд з аналізом вимог, є частиною великої стадії життєвого циклу системи, званої визначенням системи. Результати цієї стадії є вхідною інформацією для стадії реалізації системи.

Проєктування системи направлене на уявлення системи, відповідно до передбачено мети, принципів та задумів; воно включає оцінку і прийняття рішень по вибору таких компонентів системи, які відповідають її архітектурі і укладаються в запропоновані обмеження.

Результатом роботи є веб-продукт, який надає можливість майбутнім абітурієнтам проходити онлайн тестування з будь-якого пристрою за наявності інтернет-підключення.

В результаті аналізу вимог до продукту було визначено, що система повинна складатися з наступних модулів:

- база даних (далі – БД) – зберігатиме основну інформацію, а саме: дані користувачів, назви предметів, інформацію про тести (відповіді, посилання на зображення тощо);
- сервер для взаємодії з БД – опрацьовуватиме запити до БД, для зменшення навантаження на основний сервер, виступатиме як посередник між БД та сервером;
- сервер – опрацьовуватиме запити до Web API та буде зберігати зображення;
- адміністративна панель – буде взаємодіяти з сервером для створення нових тестів;
- клієнт – буде отримувати дані з сервера та демонструвати їх користувачам (учням) з допомогою графічного інтерфейсу.

Для кращого розуміння роботи системи використовують структурну схему. Структурна схема – це сукупність елементарних ланок об’єкта і зв’язків між ними, один з видів графічної моделі. Під елементарною ланкою мається на увазі частина об’єкта, системи управління тощо, яка реалізує елементарну функцію. Структурну схему системи зображено на рисунку 2.2.

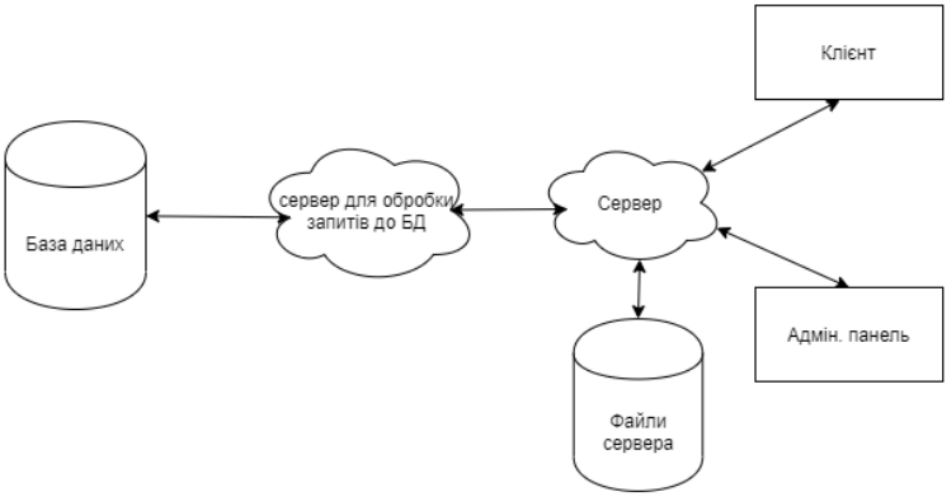

Рисунок 2.2 – Структурна схема системи

2.3 Проєктування алгоритму роботи системи

Алгоритм – це кінцева сукупність точно заданих правил рішення довільного класу задач або набір інструкцій, що описують порядок дій виконавця для вирішення деякої задачі.

Будь-який проєкт повинен мати власний алгоритм виконання для коректної роботи без неочікуваних помилок. Для цього потрібно врахувати вхідні та вихідні дані (рисунок 2.1), які описують роботу системи.

Результатом роботи повинна бути веб-платформа, яка надаватиме можливість реєстрації та авторизації користувачів, після чого вони зможуть обирати бажані предмети для навчання, здійснювати їхню конфігурацію та проходити онлайн-тести. В процесі тестування користувачам надаватиметься можливість відслідковувати прогрес кількості правильних відповідей та кількості набраних балів.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Поняття, принципи реалізації та тестування систем

Тестування – це процес дослідження, який призначений для того, аби зібрати дані про якість продукту в загальному та в контексті окремих модулів. До основних цілей тестування також можна віднести процес пошуку помилок та інших дефектів, а також випробування системи з метою її оцінки. Водночас завдяки цьому процесу можна перевірити наскільки програмний продукт відповідає заявленим до продукту вимогам і реально реалізованого функціоналу, завдяки спостереженню його роботи в штучно створених ситуаціях з специфічно для конкретного продукту обраним набором тестів.

Не виявляється можливим проведення тестування ПЗ без виконуваного коду, тобто для цього процесу потрібно мати частково або повністю завершений код, який вже відображає певний процес розробки та функціонал продукту. Зазвичай умови тестування визначаються в процесі розробки. Наприклад, при поетапному процесі, перевірка більшості тестів відбувається лише після визначення системних вимог до ПЗ після цього набір тестів можна реалізувати в тестових програмах. Проте існує діаметрально-протилежна гнучка методологія розробки ПЗ, в якій впроваджений стандарт при якому програмування та тестування відбувається водночас.

Саме тестування являє собою одну з технік контролю якості, що включає в себе наступне:

- планування робіт;
- проєктування тестів;
- виконання тестування;
- аналіз отриманих результатів.

Верифікація є процесом оцінки системи, її компонентів та модулів з метою визначення задоволення результатів поточного етапу у порівнянні з поточним етапом розробки умов, сформованими на початку цього етапу. Валідація – процес визначення відповідності ПЗ методом порівняння

очікування результату продукту та потребами користувача.

План тестування представляє з себе документ, що описує весь обсяг робіт з тестування.

Тест дизайн – етап процесу тестування, при якому розробляється певний набір тестів для перевірки бажаних штучно створених ситуацій, відповідно до раніше визначених критерій якості та цілей тестування.

Тестовий випадок – документ, в якому описана певна сукупність кроків у чітко вираженій послідовності, набір конкретних умов та параметрів, необхідних для перевірки реалізації всього ПЗ або його окремих частин.

Тестове покриття – один із параметрів оцінки якості тестування, представляє з себе щільність покриття коду чи частин додатку тестами. Переважно відображається у відсотках, потрібен для аналізу протестованого функціоналу програми чи її модулів.

Реалізація ПЗ - процес розробки, який являє собою сукупність послідовних дій, спрямованих на розробку програмного забезпечення, а саме:

- аналіз вимог;
- проєктування програмного забезпечення;
- програмування;
- тестування програмного забезпечення;
- системна інтеграція;
- впровадження програмного забезпечення;
- супровід.

Методологія розробки ПЗ – це набір чітко визначених методів, що застосовуються на різних стадіях житяного циклу розробки ПЗ, вони мають спільний філософський підхід та, в залежності від самого підходу дозволяють забезпечити максимальну ефективність процесів розробки конкретного продукту.

3.2 Вибір засобів реалізації

Після детального аналізу етапів та послідовностей дій під час розробки програмного забезпечення, наступним кроком буде вибір технологій, платформ та мов програмування, з допомогою яких буде розроблено кінцевий продукт.

Першим етапом є вибір платформи та мови програмування. Для розробки веб-застосунків використовують мову розмітки – HTML, мову каскадних стилів – CSS та в якості мови програмування – JavaScript.

HTML – мова тегів, яка надає можливість написання гіпертекстових документів для мережі Інтернет. Коли користувач переходить на конкретний сайт, веб-браузер отримує його з веб-сервера або з локальної пам’яті. Елементи HTML є «будівельними блоками» гіпертекстових сторінок. За допомогою різноманітних конструкцій тегів, зображення, інтерактивні форми та інше можуть бути легко вбудовані у візуалізовану сторінку.

HTML може вбудовувати програми, написані на мовах сценаріїв, таких як JavaScript, що дозволяє динамічно впливати на поведінку та вміст веб-сторінок, а підключення CSS дозволяє змінювати візуальний вигляд та компонувати вміст документу.

CSS – спеціалізована мова стилю сторінок, за допомогою якої описується опис їхнього зовнішнього вигляду. CSS має різні рівні та профілі, кожен з яких створюється на основі попередніх, при цьому додається новий функціонал або розширюється наявний. Дана мова стилів використовується розробниками та відвідувачами веб-сторінок для визначення основних параметрів стилів, таких як: шрифти, кольори, верстка, розміщення елементів у документів та інші.

JavaScript – мова для створення сценаріїв веб-сторінок, яка надає можливість стороні клієнта взаємодіяти з відвідувачем сайту, обмінюватися даними з сервером, керувати браузером та динамічно змінювати зовнішній вигляд сайту. Цю мову програмування класифікують як прототипну, скриптову мову програмування з динамічною типізацією.

Водночас її використовують для наступних завдань:

- написання сценаріїв для веб-сторінок, щоб надати їм інтерактивність;
- програмування на стороні сервера, з використанням платформи Node JS;
- розробка стаціонарних застосунків на базі Electron та NW.js;
- розробка мобільних застосунків;
- всередині PDF-документів.

Незважаючи на те, що мова JavaScript є досить легкою в освоєні, в дипломному проєкті її було використано як результат компіляції мови TypeScript.

TypeScript також відноситься до скриптових мов програмування, проте напряму його неможливо виконати у браузері. Перед цим його потрібно скомпілювати у JavaScript код.

Чому TypeScript? Він має наступні переваги над JS:

- статична типізація – можливість явного визначення типів;
- підтримка використання повноцінних класів;
- підтримка підключення модулів;

Незважаючи на те, що він збільшує кількість коду програми на етапі розробки, водночас він зменшує час входу нових розробників в існуючий проєкт завдяки статичній типізації та полегшує команду розробку великих проєктів.

Що потрібно зробити, аби отримати готовий JavaScript код з TypeScript? Для цього використовують різноманітні платформи та пакети для компіляції та оптимізації коду. Наприклад, платформа Node JS та, якщо брати TypeScript - це пакет «tsc» та «webpack». Для встановлення пакетів використовують системи керування пакетами. Одні з популярних на сьогодні це npm та yarn.

Системи керування пакетами – це ПЗ, яке дозволяє керувати процесом установки, налаштування, видалення та оновлення різних компонентів продукту. Пакети представляють з себе бібліотеки коду та функціоналу, які

знаходяться у відкритому доступі. Окрім дистрибутива ПЗ вони також містять набір певних метаданих, які містять в собі наступне:

- повне ім’я пакета;
- номер версії;
- опис пакета;
- ім’я розробника;
- контрольну суму;
- відношення з іншими пакетами.

Node.js – платформа для виконання високопродуктивних застосунків написаних на мові програмування JavaScript. Платформа надала можливість виконувати JS скріпти на стороні сервера та відправляти користувачеві результат їх виконання. Вона перетворила мову JS на мову загального використання з великою спільнотою розробників. Ця платформа має наступні властивості:

- асинхронна одно-нитева модель виконання запитів;
- системи модулів CommonJS;
- не блокуючий ввід/вивід;
- рушій JavaScript V8.

Для керування модулями використовують пакетний менеджер npm або yarn.

Суть одно-нитевої моделі у тому, аби усі вхідні запити та процеси їх обробки надсилати та опрацьовувати в один потік в асинхронному режимі. Якщо порівнювати останню з мовою програмування PHP, в якій кожен запит надсилається в новий потік та, яку також використовують для розробки серверної частини, одно-нитева модель збільшує швидкість обробки запитів (а саме це одна з переваг Node.js.). При необхідності існує можливість запустити Node.js сервер у багато-поточному режимі, що у рази збільшить продуктивність.

Для реалізації серверної частини використовують різні фреймворки за допомогою яких має місце спрощення та пришвидшення написання об’ємних

сервісів та модулів.

Фреймворк – комплексна структура програмних рішень, що використовується для полегшення розробки складних систем. Цю структуру можна вважати бібліотекою, але при цьому вона має своєрідні обмеження, що задають правила написання коду та організацію структури проєкту. Одна з переваг фреймворків полягає у тому, що такі програми мають стандартну структуру. Завдяки їм набагато простіше створювати засоби для автоматичного створення інтерфейсів користувача, оскільки структура реалізації коду є відомою заздалегідь.

На сьогодні один з найпопулярніших фреймворків для розробки серверної частини – Express. Він спроєктований для легкої розробки веб-застосунків та API з можливістю легкого підключення плагінів та модулів. Проте він не є достатньо оптимізованим; саме тому було обрано фреймворк Fastify.

Після вибору платформи та фреймворку для серверної частини наступним етапом буде вибір бази даних. Існує безліч видів БД, якщо класифікувати їх за моделлю організації даних можна вирізнити наступні:

- ієрархічна. Суть цієї моделі у тому, що вона представляє дані як дерево, яке складається з об’єктів різних рівнів. Відношення між об’єктами мають тип «предок-нащадок». Можливі ситуації, коли об’єкт не має нащадку або має два, а то й більше. Варто зауважити, що нащадок в такій моделі може мати тільки одного предка;
- мережна. Ця модель БД дуже подібна до ієрархічної, але вона має певну особливість, яка полягає у тому, що нащадок може мати більше одного предка;
- реляційна. Усі дані у такій БД представлені у вигляді таблиць;
- об’єктно-орієнтована. Дані у базах такого виду оформляють у вигляді моделей об’єктів;
- документо-орієнтована. Дані представлені у вигляді ієрархічних структур даних і найчастіше реалізована підходом NoSQL.

Структура дерева починається з головного вузла і може містити необмежену кількість внутрішніх і листових вузлів. Листові вузли в собі містять дані, які при додаванні нового документа заносяться в індекси. Це дозволяє при складній структурі знаходити місце та шлях шуканих даних. Якщо порівнювати зі сховищами типу «ключ-значення», то для вибору великої кількості частин документів немає необхідності завантажувати всі документи в оперативну пам'ять. Документи можуть бути організовані в колекції. Це дуже віддалений аналог таблиць реляційних СКБД, проте у випадку документо-орієнтованої БД колекції в собі можуть містити інші колекції.

Зважаючи на наведені вище типи БД, було обрано документо-орієнтовану, оскільки вона є досить легкою у розумінні та дозволяє оперувати об’єктами та масивами даних в комірках документів. Такий тип найчастіше використовується у системах керування вмістом, документальному пошуку, видавничій справі тощо. До СКБД даного типу відносяться – CouchDBm Couchbase, MarkLogic, MongoDB, eXist. Серед цих усіх було обрано MongoDB, так як вона нелише легка у розумінні, а й надає послуги хостингу БД, що дозволяє отримувати доступ до існуючої БД з будь-якої точки світу та застосунку. Також вона має свій клієнт для роботи з нею, в якому присутній легкий та зрозумілий API. Приклад архітектури документо-орієнтованої БД зображено на рисунку 3.1.

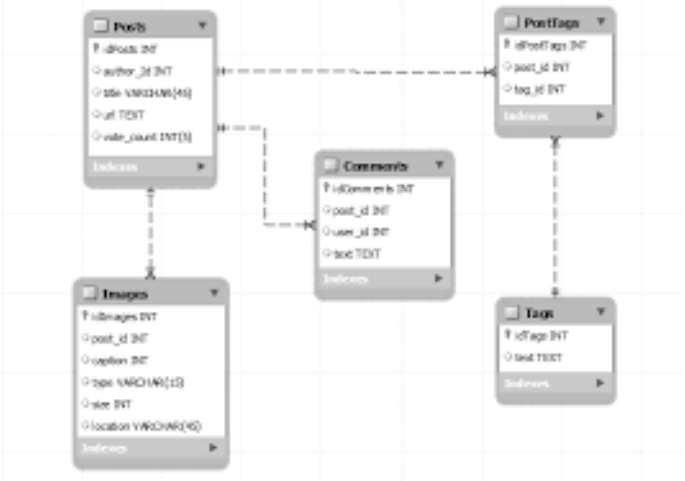

Рисунок 3.1 – Приклад архітектури документо-орієнтованої БД

API – це набір чітко визначених функцій, протоколів взаємодії та засобів для розробки ПЗ. Він використовується для веб-базованих систем, баз даних, апаратного забезпечення, бібліотек, модулів, фреймворків та іншого.

Після цього настав час вибору фреймворку чи бібліотеки для розробки клієнтської частини. Одна з найпопулярніших та найстаріших бібліотек - це jQuery. W3Techs провели дослідження і показали, що ця бібліотека використовується більше ніж половиною від мільйона найбільш відвідуваних сайтів. Основа мета jQuery – спростити та зробити зручнішим орієнтування у навігації завдяки зручному вибору елементів DOM, обробці подій, створенню анімацій, та розробці AJAX застосунків.

AJAX – це підхід для побудови клієнтських частин, при якому веб-сторінка без перезавантаження надсилає запити на сервер у фоновому режимі і при необхідності оновлює вигляд, відштовхуючись від отриманих даних. AJAX – це концепція використання декількох технологій, вона комбінує наступні методи та прийоми:

- DHTML – для динамічної зміни вебсторінки клієнта;
- XMLHttpRequest - для надсилання запитів до сервера у фоновому режимі без повного перезавантаження сторінки;
- динамічне створення дочірніх фремів.

Використання цих підходів надає можливість розробляти набагато зручніші веб-інтерфейси на тих сторінках сайту, де потребується активна взаємодія з користувачем сайту. Оскільки AJAX асинхронний, то користувач може переглядати сайт, поки клієнтська частина у фоновому режимі здійснює запит та обробляє відповідь від сервера.

Якщо порівнювати класичний підхід та AJAX, то різниця полягає у наступному: за класичного підходу при надсиланні запиту користувачем сервер обробляє запит та відсилає нову згенеровану сторінку клієнту, він в свою чергу повністю оновлює документ. Також існують випадки, за яких сервер може відправити тільки певну частину сторінки, яку потрібно оновити. У випадку AJAX, сервер віддає тільки ту частину, яка змінилась і при цьому надає

можливість не оновлювати документ повністю. Приклад AJAX запиту зображено на рисунку 3.2.

```
$.ajax({
  type: "POST",
  url: "some.php",
  data: {name: 'John', location: 'Boston'},
  success: function(msg){
    alert( "Data Saved: " + msg );
  }
});
```

Рисунок 3.2 – Приклад AJAX запиту

Цей приклад заснований на функції бібліотеки jQuery. Можна помітити, що при ініціалізації запиту є можливість вказати тип запиту POST, GET і т.д., url-адресу, на яку буде відіслано запит, дані які знаходяться у тілі запиту, а також вказати «колбек», який спрацює у випадку, коли запит повертає успішну відповідь.

Колбек - це функція, яка передається як параметр у іншу функцію і має бути викликана при певній умові. Є два типи колбеків – блокуючі та відкладені, вони відрізняються способом контролю потоків даних. У той час як перші – синхронні викликаються перед тим, як функція поверне результат свого виконання, асинхронні можуть виконуватись після повернення результату.

На рисунку 3.3 буде зображено візуальне порівняння класичного та AJAX підходів.

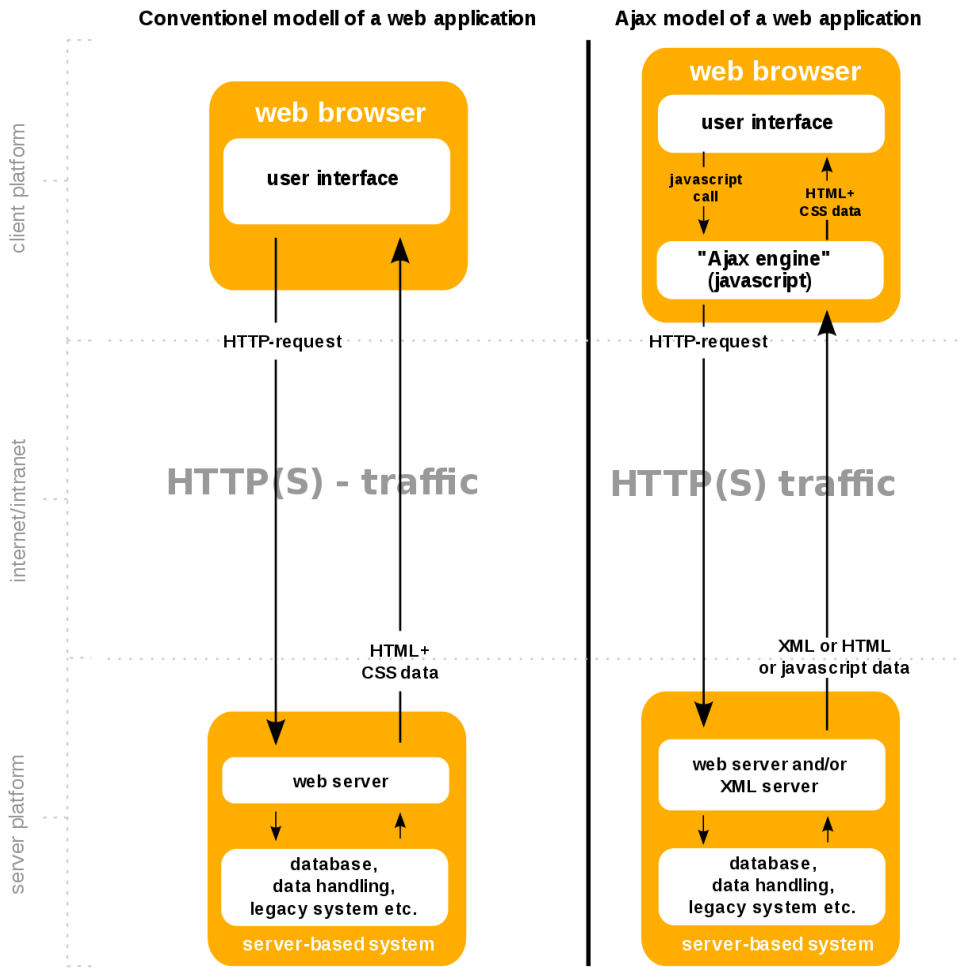

Рисунок 3.3 – Порівняння класичного та AJAX підходів

Після того як було розглянуто бібліотеку jQuery та технологію AJAX, було вирішено відмовитися від першого, оскільки вона надає недостатньо інструментів для роботи з DOM на сьогоднішній день. Для нових проєктів вона вважається нерентабельною через проблеми з підтримкою та зручністю розробки великих веб-застосунків. Проте технологія AJAX є дуже популярною і зручною у використані. З кожним днем вона набирає обертів, а тому і вибір було зроблено в її користь, хоча було використано інший клієнт для ініціалізації таких запитів. Про нього буде йти мова після вибору фреймворку для клієнтської частини.

На сьогодні існує декілька найпопулярніших фреймворків для розробки користувацьких інтерфейсів, а саме: React, Vue, Angular. Розглянемо усі почергово.

React розробляється Facebook, а також підтримується Instagram та спільнотою розробників і деяких корпорацій.

Його можна використовувати для розробки односторінкових сайтів та мобільних додатків. Його головна ціль – це надати простоту, високу швидкість та масштабованість. При розробці клієнтської частини також використовують додаткові бібліотеки, такі як MobX, Redux, Effector, GrapgQL.

До особливостей React можна віднести наступне:

- однонаправлена передача даних. При такому підході дані передаються від батьківських компонентів до дочірніх, а для зміни даних використовують колбеки. При цьому властивості є незмінними і для їх модифікації потрібні ті самі колбеки. Такий механізм має назву «властивості вниз, події наверх»;
- віртуальний DOM. При такому підході фреймворк створює кеш-структуру в оперативній пам’яті, що дозволяє вираховувати різницю між попереднім і теперішнім станом інтерфейсу. Програміст при роботі зі сторінкою може вважати, що оновлюється вся, проте React сам вирішує які елементи потрібно оновити;
- JSX – додатковий синтаксис JS, який дозволяє використовувати подібний до HTML синтаксис для опису сторінку структуру користувацького інтерфейсу.
- методи життєвого циклу. Вони дозволяють взаємодіяти з даними компонента на різних стадіях його життя. Наприклад, якщо потрібно викликати функцію, яка відсилає запит на сервер одразу після того як компонент ініціалізувався, використовують метод «componentDidMount»; якщо потрібно відмінити пере рисовку - існує метод «shouldComponentUpdate», в якому потрібно повернути значення false.
- відмальовка в браузері не тільки HTML. Наприклад у Facebook існують динамічні графіки, які підмальовують теги у canvas. Netfilx та PayPal підмальовують ідентичний HTML на сервері та клієнті. Це називають ізоморфними завантаженнями.

На рисунку 3.4 зображено архітектуру React з використанням бібліотеки Redux.

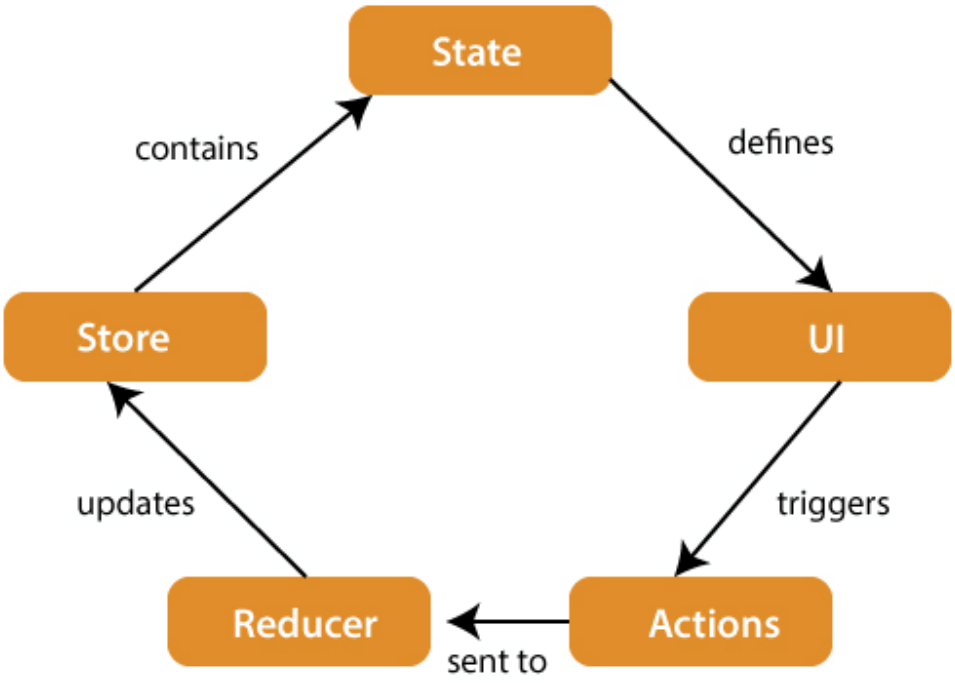

Рисунок 3.4 – Архітектура застосунку на основі React з використанням Redux

Суть архітектури полягає у тому, що існує централізоване сховище даних, до якого можна отримати доступ з будь-якої точки застосунку. Користувацький інтерфейс ініціалізує події, які в свою чергу обробляються в reducer'і; він оновлює все сховище і одразу це впливає на стан компоненту. Після цього користувацький інтерфейс оновлюється. Одним з недоліків такої архітектури є централізоване сховище, яке неможливе оновити частково; воно обов'язково має оновитися повністю і у випадку, якщо веб-застосунок великий це може впливати на його продуктивність. Але це вважається стандартом, який досить добре зарекомендував себе на ринку. Також до мінусів можна віднести написання великої кількості коду, якого можна було б уникнути при використанні інших бібліотек.

Vue.js – JS фреймворк для розробки користувацьких інтерфейсів. Його легко можна інтегрувати в існуючі проєкти, які використовують інші JS-бібліотеки. Використовується він для розробки односторінкових веб-застосунків в реактивному стилі.

Що ж таке реактивний стиль? Це парадигма програмування, яка є орієнтованою на потоки даних і розповсюдження даних. При такому підході існує можливість легко виражати статичні та динамічні потоки даних.

Якщо порівнювати імперативний та реактивний підходи, то такий вираз «a = b + c» присвоїть змінній «a» результат «b + c» і при подальших змінах останніх змінних вони не вплинуть на значення «a», в той час як в реактивному стилі, при зміні «b та c» значення «a» буде автоматично перераховане.

Архітектура клієнтської частини з використанням vuex зображено на рисунку 3.5.

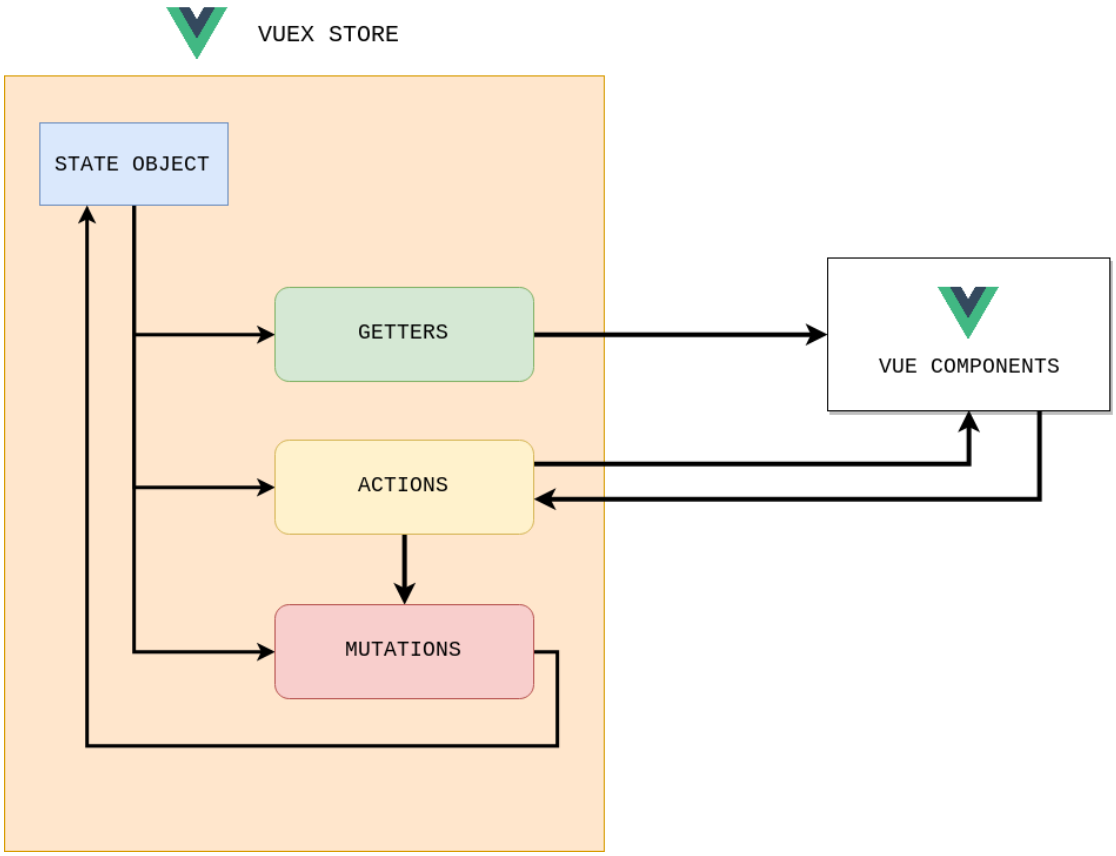

Рисунок 3.5 – Архітектура з використанням бібліотеки vuex

Vuex – аналог redux, пристосований для фреймворку vue, тому поведінка та архітектура дуже подібна.

Наступним буде розглянуто AngularJS. Він також має відкритий код, а розробником цього фреймворку є компанія Google. Призначений для розробки односторінкових сайтів, які складаються з одного HTML документу, CSS та JavaScript. Основна ціль – розширення браузерних застосунків на підставі шаблону «модель-вид-контролер». Водночас він спрощує їх тестування та розробку. Фреймворк працює з HTML - документом, який містить додаткові атрибути і при цьому пов’язує області для вводу чи виводу з моделлю, які зазвичай є стандартними змінними у JS. За даними службами аналізу AngularJS використовую наступні сайти: Wolfram, NBC, Intel, Sprint, Walgreens. Цей фреймворк заснований на переконанні, що для створення користувацьких інтерфейсів слід використовувати декларативне програмування, а для бізнес-логіки додатку більше прийнятним буде імперативне.

AngularJS притаманні такі особливості:

- відокремлення DOM-маніпуляцій від бізнес-логіки, що в свою чергу впливає на архітектуру на структуру проєкту;
- розподілення клієнтської та серверної частини для зручності розробки.

З мінусів можна виділити високий поріг входження. AngularJS має велике середовище, яке потрібно розуміти та знати повністю при розробці веб-застосунків. На рисунку 3.6 зображено архітектуру клієнтської частини, яка заснована на фреймворку AngularJS.

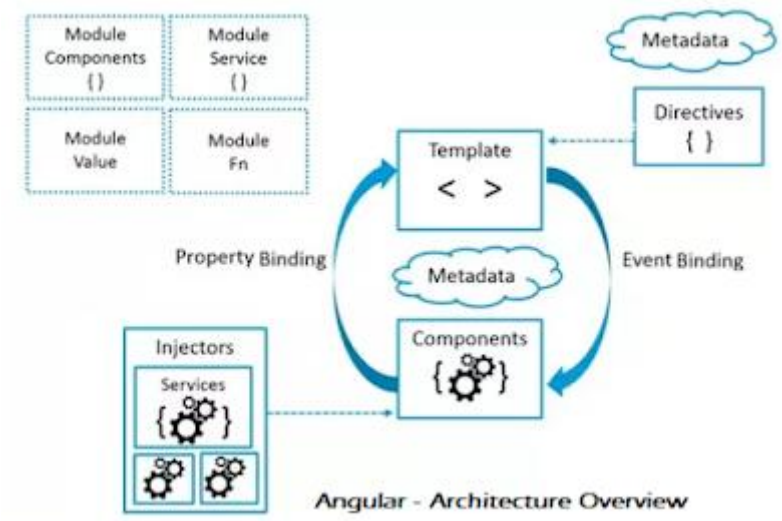

Рисунок 3.6 – Архітектура з використанням AngularJS

Виходячи з опису трьох фреймворків нами було обрано React, оскільки він є більш зручний в розробці, дозволяє швидко розуміти структуру та архітектуру існуючих проєктів в ситуаціях, коли підключаються нові розробники. Він не потребує одразу розуміння всього середовища і дає можливість підключати необхідні бібліотеки лише у випадку потреби. В той час як AngularJS заставляє розробників розуміти все і одразу. Це дає можливість його вивчати поступово, а також зменшити розмір вихідного коду, що зменшить швидкість завантаження сторінки, навіть при умові, що розробник не є достатньо досвідчений та має невеликий досвіт робити з цією технологією. Окрім цього він дозволяє писати HTML код в js файлах, а також використовувати технологію CSS-IN-JS (JSSS).

В якості бібліотеки для організацію глобального сховища було використано redux, а саме redux-toolkit, незважаючи на те, що він потребує необхідності написання більшої кількості коду, він дозволяє розробляти продукт з зрозумілою архітектуру як для початкового розробника, так і для тих розробників, які можуть підключатися з часом.

Для надсилання асинхронних запитів (AJAX) було використано бібліотеку axios.

Axios – HTTP клієнт заснований на промісах, що дозволяє зручно надсилати та обробляти запити до сервера у фоновому режимі.

HTTP – протокол передачі даних в комп'ютерних мережах. Його основна ціль передача веб-сторінок з розміткою HTML, проте завдяки ньому успішно передаються і інша інформація.

3.3 Вибір способу захисту користувачів

Платформу та фреймворки для клієнтської і серверної частини уже обрано, тому необхідно обрати спосіб захисту користувачів.

У цій області можна вибрати головні терміни авторизація та автентифікація. Їх часто неправильно розуміють, а тому є необхідність дати визначення цих термінів.

Автентифікація – це процес підтвердження справжності чого-небудь або кого-небудь, у нашому випадку – користувача. Цей процес відбувається шляхом надання певної інформації, завдяки якій можна перевірити, чи цей користувач існує та переконатись, що це саме він. Якщо розглядати область веб-застосунків, то найпопулярніший метод автентифікації це надання емейлу та паролю. При умові що вони вірні користувач вважається автентифікованим.

Авторизація – процес перевірки прав доступу до запрошуваних ресурсів та даних. Також завдяки їй можна визначити чи може користувач взагалі запрошувати бажані дані або виконати певну дію.

Оскільки з автентифікацією усе зрозуміло, варто обрати спосіб авторизації користувачів. Існує безліч способів авторизації, тому потрібно розглянути усі з метою вибору найкращого для конкретного продукту.

Але перед цим потрібне розглянути ще одне поняття – cookie.

Куки – текстові або бінарні дані, які отримує клієнт від сервера. Вони зберігаються на стороні клієнта, тобто в браузері. Водночас ця технологія дозволяє серверу переглядати куки при запитах зі сторони клієнта, оскільки вони будуть прикріплені до кожного запиту, при умові якщо вони існують. В більшості випадків вони створюються за ініціативою сервера, проте є незначна

кількість випадків, за яких клієнт може сам їх записувати. Куки переважно використовують для зберігання уподобань користувачів, статистики, мови користувача, локації тощо.

Окрім наведених вище даних, в куках також зберігають важливі дані, доступ до яких обмежується зі сторони клієнта.

Опції при ініціалізації куків:

- name – назва куки, яка необхідна для подальшої ідентифікації та можливості отримати доступ до значення бажаної куки;
- expires – має формат дати та часу, ідентифікує через скільки часу кука пропаде з браузера, також може приймати значення «Session». При такому значенні кука пропадає після того, як користувач повністю закриває браузер або виключає комп'ютер;
- domain – визначає домен, для якого кука буде дійсною;
- path – визначає підмножини URL, для яких куки будуть дійсними;
- secure – якщо «true», куки будуть використовуватись тільки при HTTPS з'єднані, якщо «false» - при будь якому;
- httpOnly – якщо «true» - обмежується доступ з JS, тим самим унеможливлюється зміна значення конкретної куки зі сторони клієнта, в іншому випадку дозволяється модифікувати значення куки.

HTTPS - це протокол з'єднання який синтаксично ідентичний до HTTP. При використанні першого протоколу використовується інший порт, а саме – 443, окрім цього використовується SSL або TLS протоколи для того щоб дані, які будуть відправлятися клієнтом до сервера і у зворотньому зв'язку були захищені. Єдине, що не шифрується – це адреса веб-хоста та порт.

На рисунку 3.7 можна побачити, що при захищеному з'єднані у випадку спроби хакера перехопити дані під час запиту, він отримає їх у зашифрованому вигляді та нічого не зможе з ними зробити.

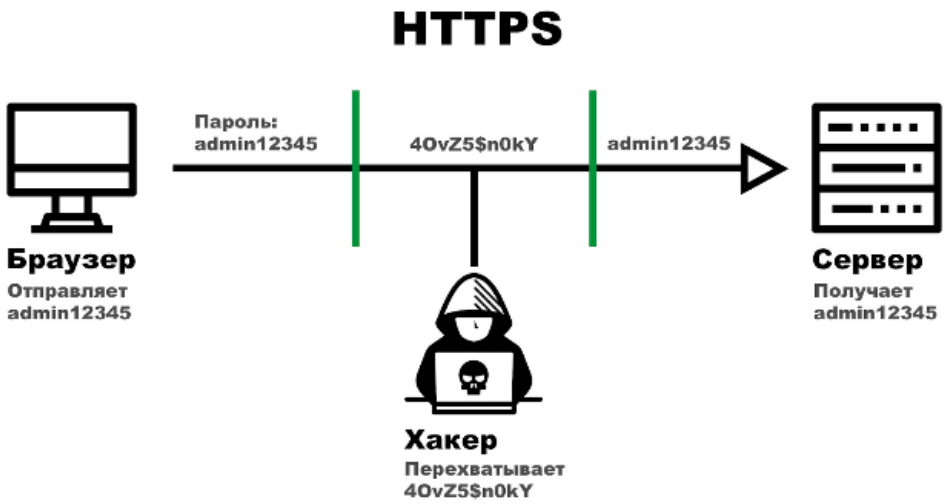

Рисунок 3.7 – Схема перехоплення даних хакером

Далі можна перейти до способів авторизації користувачів. Один з популярних – Forms authentication. Після того як користувач успішно автентифікувався в куках з'являється session token. Після цього при кожному відправлені запиту стороною клієнта, вони надсилаються разом з ним. Сервер у свою чергу перевіряє, чи такий токен існує в БД і після цього може визначити, що це за користувач і які права доступу він має.

Наступний способів автентифікації – за сертифікатами. Він представляє з себе набір даних та атрибутів, які ідентифікують власника цього сертифіката. Окрім цього, сертифікат криптографічно пов'язаний з закритим ключем, який може зберігатися у власника в операційній системі, в браузері, файлі або навіть на окремому фізичному пристрої. Цей механізм добре підтримується браузерами, проте не усі сайти мають такий спосіб автентифікації. На рисунку 3.8 зображено схему використання сертифікатів для автентифікації.

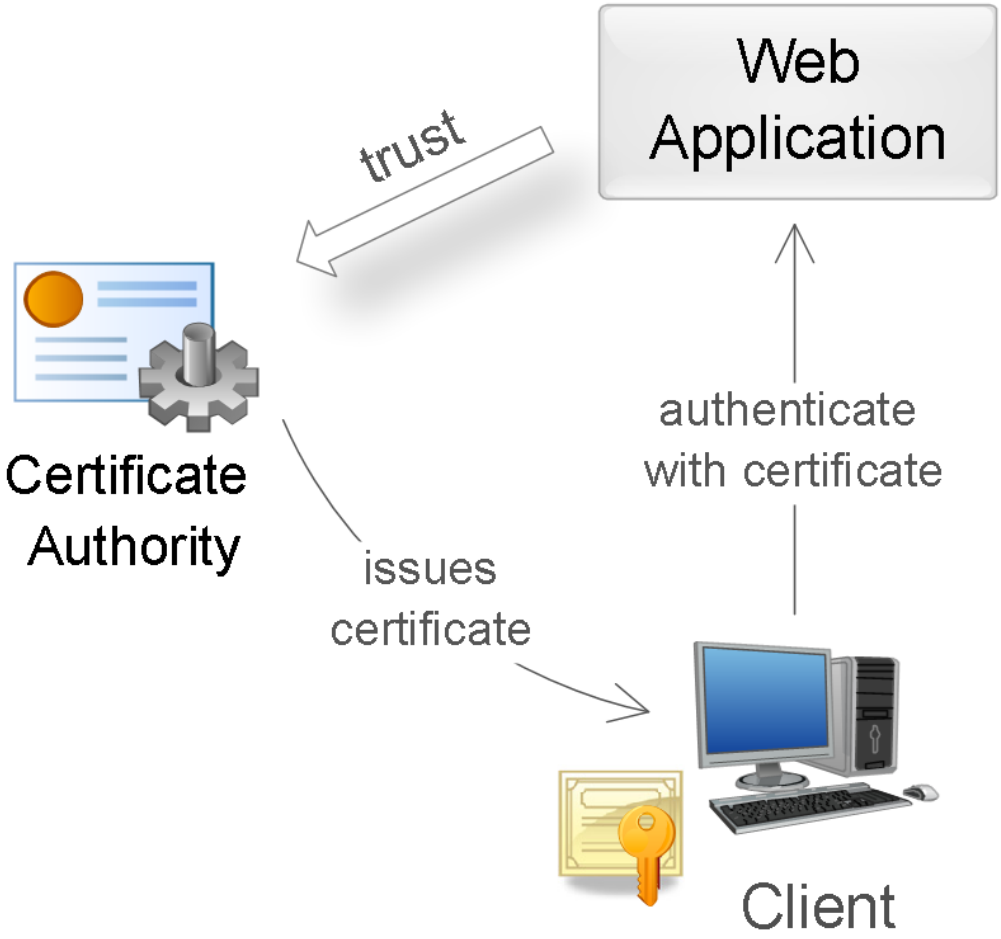

Рисунок 3.8 – Схема використання сертифікатів для автентифікації

Під час автентифікації сервер перевіряє сертифікат на основі наступного:

- сертифікат повинен бути підписаний перевіреним органом сертифікації;
- сертифікат повинен бути дійсний на теперішню дату;
- сертифікат не повинен бути відізваним органом сертифікації, який його підписав.

На рисунку 3.9 зображено приклад X.509 сертифіката.

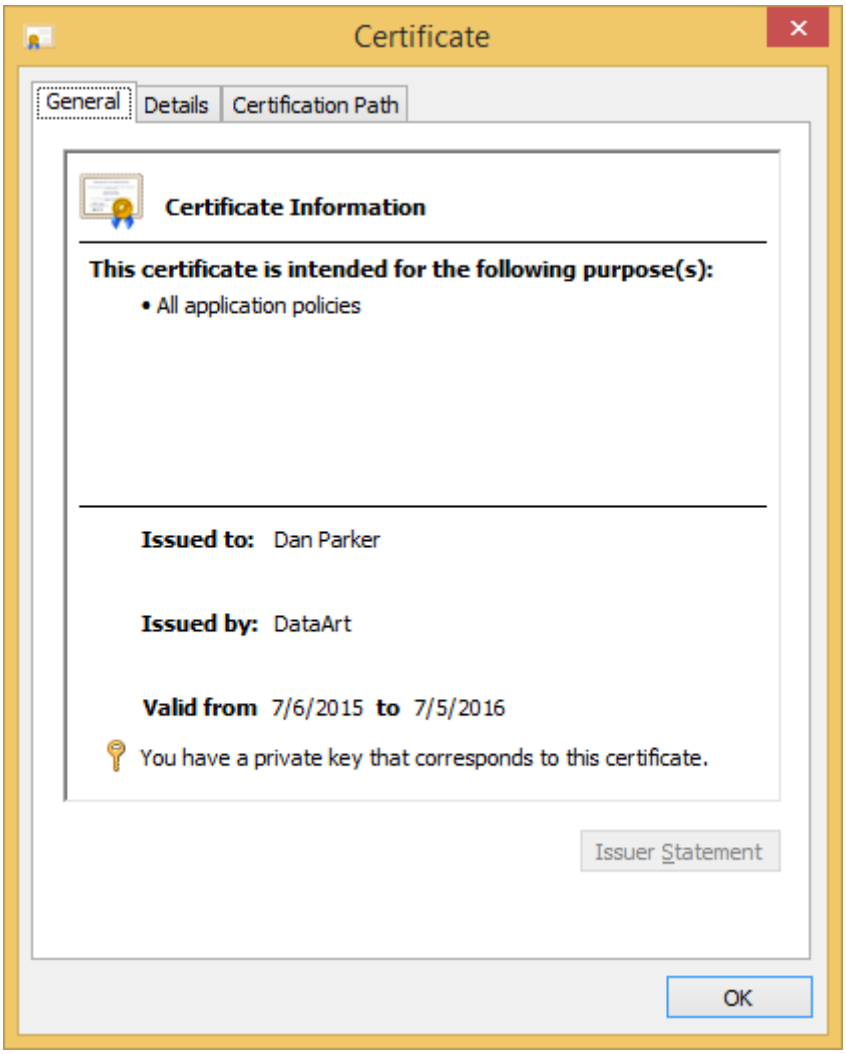

Рисунок 3.9 – приклад X.509 сертифіката.

Автентифікація з допомогою X.509-сертифіката відбувається в момент підключення до сервера і являється частиною протоколу SSL/TLS.

Наступний спосіб авнтентифікація за ключами доступу. Його частіше використовують для автентифікації пристроїв та сервісів при звернені до інших веб-сервісів. В якості секрету використовуються ключі доступу – це довгі унікальні строки з випадкових символів, які по факту заміняють стандартну пару емеїл/пароль. Хороший приклад застосування автентифікації за ключем – хмара Amazon Web Services, яка зображено на рисунку 3.10.

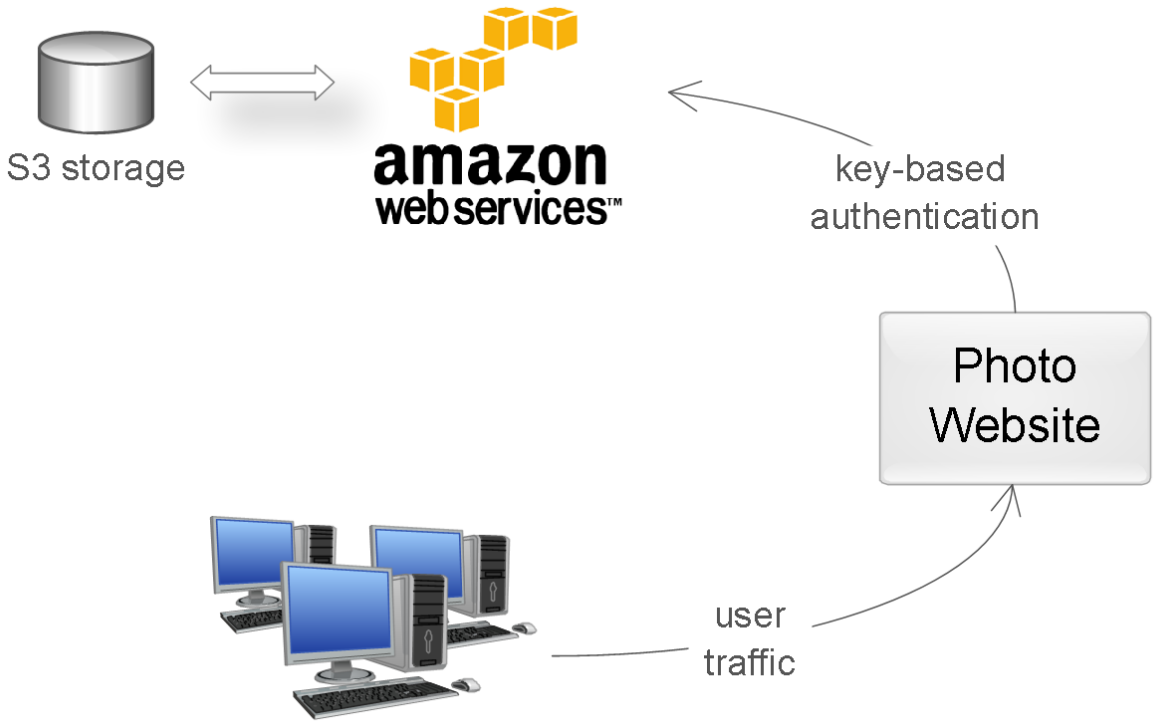

Рисунок 3.10 – Приклад застосування автентифікації по ключу

У даному прикладі користувач через консоль AWS може створити ключ, який має обмежений доступ до хмари: тільки читання/запис в Amazon S3. В результаті створений ключ можна використовувати для автентифікації веб-застосунку в хмарі AWS.

Окрім цього існує можливість передачі ключа доступу разом з HTTP. З технічної точки зору ключ може знаходитися у будь якій частині запиту, проте найбільш оптимальний – HTTP заголовок. Для того щоб запобігти викраденню ключів необхідно, щоб з'єднання було захищено протоколом SSL/TLS. Приклад автентифікації за ключем доступу, який передається в HTTP заголовку зображено на рисунку 3.11.

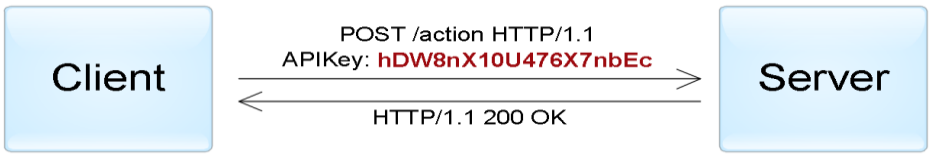

Рисунок 3.11 – Приклад автентифікації по ключу доступу в HTTP заголовку

Наступний спосіб автентифікації по токенам. Такий спосіб частіш усього використовують при побудові розподільних систем, в яких один додаток делегує функцію автентифікації іншому застосунку. На загальному рівні процес виглядає наступним чином:

- клієнт автентифікується в застосунку, який засвідчує особу одним з специфічних способів для кожного користувача. Це може бути, наприклад, пара емеїл/пароль;
- цей оченіунок в свою чергу видає клієнту токен для застосунку, з яким користувач бажає взаємодіяти;
- клієнт автентифікується в бажаному застосунку з допомогою цього очені.

Приклад автентифікації «активного» клієнта за допомогою очені зображено на рисунку 3.12

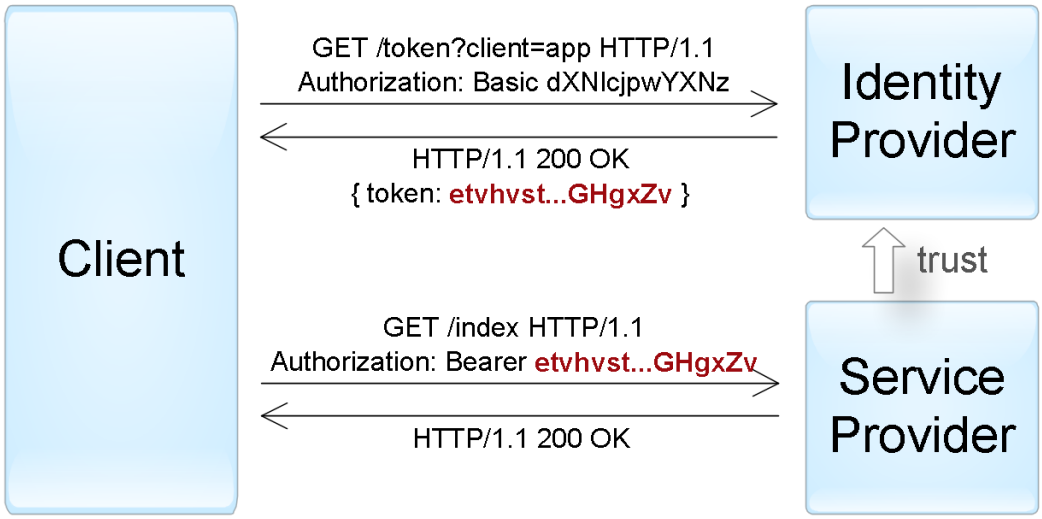

Рисунок 3.12 – Приклад автентифікації «активного» клієнта за допомогою токенів

Існує декілька форматів токенів, які будуть розглянуті нижче.

Simple Web Token – найбільш простий формат, який представляє собою довільний набір пар оч'я/значення в форматі кодування HTML form. Він містить декілька зарезервованих імен: Issuer, Audience, ExpiresOn и

HMACSHA256. Підписується за допомогою симетричного ключа, який означає, що обидва застосунки повинні мати цей ключ. Приклад SWT токену після декодування представлений на рисунку 3.13.

```
Issuer=http://auth.myservice.com&
Audience=http://myservice.com&
ExpiresOn=1435937883&
UserName=John Smith&
UserRole=Admin&
HMACSHA256=KOUQRPSpy64rvT2KnYyQKtFFXUIggnesSpE7ADA4o9w
```

Рисунок 3.13 – Приклад декодованого SWT токену

JSON Web Token – складається з трьох блоків, які розділені точками: заголовок, набір полів та підпис. Перші представляються в вигляді JSON та додатково кодуються в формат base64. JWT також має свій набір зарезервованих полів: iss, aud, exp, iat та інші. Підпис може генеруватися як за допомогою синхронних, так і асинхронних алгоритмів шифрування. Приклад підписаного очені після декодування зображено на рисунку 3.14.

```
{ «alg»: «HS256», «typ»: «JWT» }.
{ «iss»: «auth.myservice.com», «aud»: «myservice.com», «exp»: «1435937883», «userName»: «John
Smith», «userRole»: «Admin» }.
S9Zs/8/uEGGTVVtLggFTizCsMtwOJnRhjaQ2BMUQhcY
```

Рисунок 3.14 – Приклад декодованого JWT токену

Проаналізувавши методи автентифікації та авторизації, було обрано метод автентифікації по токенам з модифікацією того, що оченіунок, який видає токен і оченіунок, з яким взаємодіє користувач один і той самий. Тип очені – JWT. Він буде містити основну інформацію про користувача – id, email, role. Це надасть можливість зменшити кількість звернень до БД, адже основні

дані, які можуть бути потрібні серверу при запиті будуть міститись у самому токені.

3.4 Реалізація клієнтської частини

Нами було обрано фреймворк React, тому розглянемо два концепти методи створення компонентів.

Перший – класовий, в такому випадку використовується наслідування від батьківського класу Component або PureComponent. Такий компонент в обов'язковому порядку повинен мати метод render, який повертає HTML теги для підмальовування бажаного контенту на сторінці користувача. Окрім цього він має методи життя, про деякі з них було описано у розділі 3.2.

Кожен компонент може мати свій стан (state). Для його зміни в батьківському класі є метод setState, який може приймати об'єкт як параметр з парами ключ/значення, які необхідно змінити. Варто зазначити, що стан не оновляється миттєво, це означає, що якщо у методі викликати метод setState і після цього одразу звернутися до стану компонента, ми отримаємо попередню версію цього стану. Приклад створення класового компонента зображено на рисунку 3.15.

```
class Car extends React.Component {
  render() {
    return <h2>Hi, I am a Car!</h2>;
  }
}
```

Рисунок 3.15 – Приклад створення класового компонента

Наступний підхід заснований на функціональних компонентах. Вони мають вигляд звичайних функцій, які можуть приймати параметри (props) та завжди мають повертати HTML теги. Для того, щоб коректно зберігати стан компонента існують так звані хуки, у випадку маніпуляцій зі станом

використовують useState. Цей хук має наступний синтаксис «const [value, setValue] = useState(defaultValue);». Проаналізувавши цей код, можна зрозуміти, що перше значення туплу, який повертає хук - це сама змінна з якої можна читати дані, а друге - це так званий сеттер, який дозволяє змінювати цю змінну. При виклику цього хуки бажано вказувати стандартний, проте це не є обов'язковим.

Окрім цього, існує хука, яка відслідковує зміну певної змінної – useEffect. Вона має наступний синтаксис «useEffect (callback, [variables array])». Перший параметр – це функція, яка викликається при спрацьовувані useEffect, друга – масив змінних, при зміні яких потрібно викликати колбек. Такий набір параметрів дозволяє замінити метод життєвого циклу componentDidUpdate. Проте, якщо забрати масив змінних, цей хук спрацює лише один раз – при ініціалізації компонента, тим самим він заміняє метод componentDidMount.

Проаналізувавши ці два підходи була надана перевага функціональному, через те що, він зменшує кількість написаного коду, а також дозволяє користуватись хуками, які в свою чергу доволі спрощують розробку клієнтської частини.

Наступним, що необхідно нам розглянути – Redux менеджер глобального стану застосунку. Він має декілька головних функцій, які було використані при розробці цього проєкту.

CreateSlice – дозволяє ініціалізувати стан окремої частини глобального стану та описати функції, які будуть впливати на стан бажаної частини застосунку. На рисунку 3.16 зображено приклад використання функції createSlice.

```
/** Create slice */
const errorHandler = createSlice({
    initialState,
    name: 'ErrorHandler',
    reducers: {
        setErrorAction: {
            reducer: (
                state: IErrorHandlerInitialState,
                { payload }: ISetErrorAction,
            ) => ({
                ...state,
                ...payload,
            }),
            prepare: (error?: AxiosError): ISetErrorAction => ({
                payload: error
                    ? {
                        message: error.message,
                        status: error.response
                            ? error.response.statusText
                            : undefined,
                        statusCode: error.response
                            ? error.response.status
                            : undefined,
                    }
                    : {
                        message: '',
                        status: '',
                        statusCode: 200,
                        data: null,
                    },
            }),
        },
    },
});

/** Export actions */
export const {
    setErrorAction,
} = errorHandler.actions;

/** Export reducer */
export default errorHandler.reducer;
```

Рисунок 3.16 – Приклад використання функції createSlice

Ця функція приймає в себе об’єкт як параметр, який містить наступні властивості:

- initialState – початковий стан, який може бути як складний об’єкт так і проста стрічка, число чи нулеве значення;
- name – назва слайсу, яка може знадобитися при відслідковуванні викликаних екшенів;
- reducers – назви функцій та опис їх поведінки з конкретним слайсом. Кожна функція може приймати в себе два параметри: теперішній стан та параметри, якими вона була викликана, якщо такі є.

Ініціалізація головного редюсера відбувається за допомогою функції configureStore, яка приймає один параметр – об’єкт з наступними властивостями:

- middleware – список прошарків, які можуть відслідковувати виклик певних екшенів та впливати на глобальний стан додатку;
- preloadedState – необхідний для того, щоб завантажити користувацький стан інтерфейсу, використовується наприклад для тестування, щоб отримати стан з тестовими даними;
- reducer – приймає один редюсер або декілька об’єднаних за допомогою combineReducers.

Для вибору бажаних даних та їх миттєвого опрацювання з функцією мемоізації (не дозволяє проводити додаткові обрахування, якщо запрошувані дані не змінились) використовують функцію createSelector. Вона може приймати в себе до 11 параметрів, кожен з яких передає результат вибору в наступну функцію, створюючи при цьому певний ланцюжок. Приклад функції createSelector зображено на рисунку 3.17.

```
/**
 * Select ids and names of sub-subjects.
 * If property 'subSubjects' doesn't exist
 * then selector returns null.
 *
 * In other case selector returns array of names(string)
 * if some object doesn't have property 'name'
 * then this object will be skiped.
 */
export const selectSubjectConfigSubSubjectsData = createSelector(
    selectSubjectConfig,
    (subjectConfig) => {
        if (!subjectConfig) {
            return null;
        }

        /** Extract subSubjects from config */
        const { subSubjects } = subjectConfig;

        /** Check is property 'subSubjects' exist or it is an empty array*/
        if (!subSubjects || subSubjects.length === 0) return null;

        return subSubjects.reduce((acc, curr) => {
            /**
             * Return new array from previous array and current value.
             */
            return acc.concat({ name: curr.name, id: curr.id });
        }, []);
    },
);
```

Рисунок 3.17 - Приклад використання функції createSelector

Окрім наведеного вище функціоналу існує також можливість створення екшенів, які виконують асинхронні дії. Таке може бути необхідним для взаємодії з API. Ці екшени приймають як параметри будь-які дані, та повертають функцію параметрами якої є наступні змінні:

- dispatch – штучно викликає екшн для того, щоб змінити глобальний стан застосунку.
- getState – дозволяє отримати стан застосунку на теперішній момент;
- extraArgument – додаткові параметри, які можуть бути додані при ініціалізації стору.

Приклад такого екшену зображено на рисунку 3.18.

```
export const fetchSignUpAction = (credentials: IFetchSignUpActionCredentials) =>
    async (dispatch: Dispatch<any>, _: () => RootState, history: History) => {
        dispatch(signUpLoadingAction(true));

        const invalidData = verifySignUpCredentials(credentials);

        if (!invalidData) {
            return await api.signup({
                email: credentials.email,
                password: credentials.password,
            })
                .then(response => {
                    dispatch(signUpLoadingAction(false));

                    history.push(`/auth/signin?email=${credentials.email}`);
                    return response;
                })
                .catch(error => {
                    if (error.response.data.errors) {
                        const errorData = error.response.data.errors;
                        dispatch(setSignUpErrorFieldsAction(errorData.errorFields));
                        dispatch(setSignUpFieldsMessagesAction(errorData.errorMessages));
                    }
                    dispatch(signUpLoadingAction(false));
                });
        }

        dispatch(signUpLoadingAction(false));
        dispatch(setSignUpErrorFieldsAction(invalidData.invalidFields));
        dispatch(setSignUpFieldsMessagesAction(invalidData.fieldsMessages));
    };
```

Рисунок 3.18 – Приклад асинхронного екшену

Можна помітити, що цей екшн виконує запит до API і після отримання відповіді обробляє його.

Тепер час розглянути реалізацію функціоналу специфічних модулів та компонентів цього проєкту.

Першим буде функціонал який відображає прогрес тесту, який зображено на рисунку 3.19.

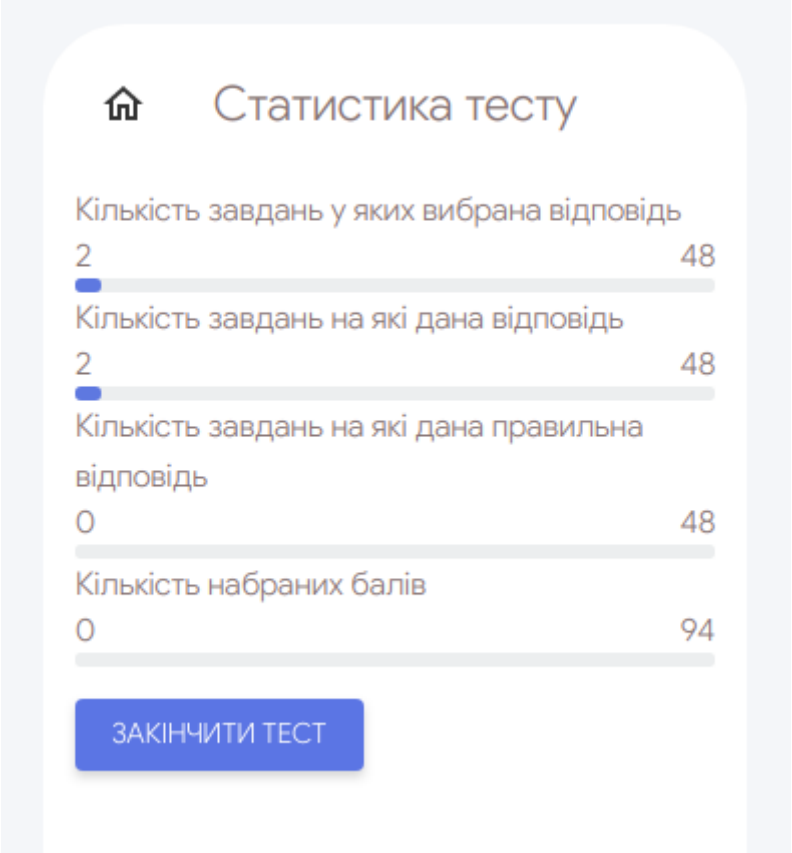

Рисунок 3.19 – Панель прогресу тесту

Компонент рендерить 4 шкали прогресу за наступним шаблоном, який зображено на рисунку 3.20.

```
<Progress
    current={amountOfSelected}
    total={answersAmount}
    label='Кількість завдань у яких вибрана відповідь'
/>
```

Рисунок 3.20 – Шаблон шкали прогресу

Дані компонент отримує завдяки селекторам з централізованого стору, вони в свою чергу розраховують прогрес за певними формулами, відштовхуючись від констант.

Фрагмент коду подано нижче:

```
export const selectCurrentAmountOfPoints = createSelector(
    selectAnswers,
    selectTestSuiteFinished,
    (answers, finished) => answers.reduce((points, answer) => {
        if (answer.type === 'SINGLE') {
            return points + getAmountOfPointsSingle(answer, finished);
        }

        if (answer.type === 'RELATIONS') {
            return points + getAmountOfPointsRelations(answer, finished);
        }

        if (answer.type === 'TEXT') {
            return points + getAmountOfPointsText(answer, finished);
        }

        return points;
    }, 0),
);

export const selectMaxAmountOfPoints = createSelector(
    selectAnswers,
    (answers) => answers.reduce((acc, curr) => {
        if (curr.type === 'SINGLE') return acc + TestSuite.PointsPerSingle;
        if (curr.type === 'RELATIONS') return acc + TestSuite.PointsPerRelati
ons;
        if (curr.type === 'TEXT') return acc + TestSuite.PointsPerText;

        return acc;
    }, 0),
);
```

Наступний момент реалізація каруселі, яка дозволяє вибрати завдання в тесті. Зображення каруселі зображено на рисунку 3.21.

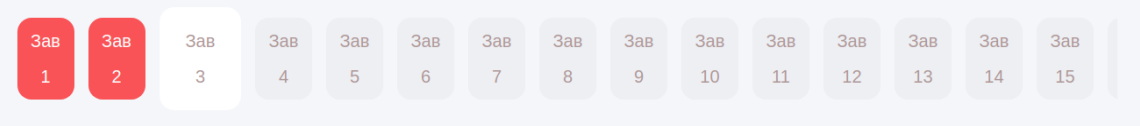

Рисунок 3.21 – Карусель для вибору завдань тесту

На рисунку 3.22 буде зображено елемент, який рендерить компонент, що відповідає за карусель.

```
return (
    <div
        className={classes.root}
        onMouseMove={handleOnMouseMove}
        onMouseDown={handleOnMouseDown}
        onMouseUp={handleOnMouseUp}
        ref={rootContainerRef}
    >
        <Box
            className={classNames(classes.innerContainer, {
                [classes.animateContainer]: animate,
            })}
            left={offsetX}
        >
            {tiles}
        </Box>
    </div>
);
```

Рисунок 3.22 – Рендер компоненту, який повертає компонент відповідаючий за карусель

Можна помітити, що головний div має три лістенера, які відповідають за анімацію каруселі. Код лістенерів подано нижче:

```
const handleOnMouseMove = (event: React.MouseEvent<HTMLElement, MouseEven
t>) => {
    if (mouseDown) {
        const mouseOffsetX = event.movementX / 2;
        const newOffsetX = offsetX + mouseOffsetX;

        handleSetOffset(newOffsetX);
    }
};

const handleOnMouseDown = () => {
    toggleMouseDown(true);
    toggleAnimate(false);
};

const handleOnMouseUp = () => toggleMouseDown(false);
```

Вони відповідають за оновлення змінних, на які реагує хук useEffect про який було розказано вище, крім того фіксується зміна позиції курсора миші.

Код хука useEffect подано нижче:

```
useEffect(() => {
        if (!mouseDown && activeTask !== 0) {
            /**
             * Difference between last tile and new active el.
             * 2 - lastTileIndex offset.
             */
            const lastDiff = getLastTileIndex() - 2 - activeTask;
            /**
             * Difference between first tile and new active el.
             * 2 - firstTileIndex offset.
             */
            const firstDiff = getFirstTileIndex() + 2 - activeTask;

            if (lastDiff < 3 && lastDiff >= 0) {
                toggleAnimate(true);
                handleSetOffset(Math.floor(offsetX - (buttonsAmount -
2) * (tileWidth + tilesSpacing)));
            }

            if (firstDiff < 3 && firstDiff >= 0) {
                toggleAnimate(true);
                handleSetOffset(Math.floor(offsetX + (buttonsAmount -
2) * (tileWidth + tilesSpacing)));
            }
        }
        if (activeTask <= getFirstTileIndex()) {
            const firstDiff = getFirstTileIndex() + 2;

            if (firstDiff !== 0) {
                toggleAnimate(true);
                handleSetOffset(Math.floor(offsetX + (getLastTileIndex() -
2) * (tileWidth + tilesSpacing)));
            }
        }
    }, [activeTask]);
```

Основна роль цього хуку у тому, щоб правильно розрахувати зміщення каруселі вліво чи вправо. Завдяки тому, що він реагує на зміну активного завдання, анімація спрацьовує навіть тоді, коли обране завдання змінюється програмно, а не через взаємодія користувача з інтерфейсом.

Взаємодія з API. Усі операції для надсилання та обробки запитів до сервера відбуваються в асинхронних екшенах, про які йшла мова раніше. Усі кінцеві точки описані в окремому класі, для зручності.

Код класу який зберігає усі кінцеві точки та методи для надсилання запиту до них подано нижче:

```
class Api implements IApi {
    axiosInstance: AxiosInstance;

    constructor() {
        const mode = process.env.NODE_ENV || 'production';
        const baseURL = mode === 'production'
            ? `${process.env.API_ENDPOINT}`
            : 'http://localhost:4000';

        this.axiosInstance = axios.create({
            baseURL,
            timeout: 10000,
        });
    }

    signup = async (credentials: ISignUpCredentials) =>
        await this.axiosInstance.post(
            'api/auth/signup',
            { ...credentials },
            { withCredentials: true },
        )

    signin = async (credentials: ISignInCredentials) =>
        await this.axiosInstance.post(
            'api/auth/signin',
            { ...credentials },
            { withCredentials: true },
        )

    me = async () =>
        await this.axiosInstance.get(
            'api/auth/me',
            { withCredentials: true },
        )

    logout = async () =>
        await this.axiosInstance.post(
            'api/auth/logout',
            {},
            { withCredentials: true },
        )
```

Конструктор цього класу ініцією так званий інстанс бібліотеки axios, при цьому доменне ім’я використовується як поле опцій при ініціалізації, що дозволяє в майбутньому не вказувати повний шлях, а лише стрічку після доменного імені.

3.5 Реалізація серверної частини

Оскільки було обрано фреймворк fastify та авторизацію з використанням JWT токена, будуть описані деякі специфічні моменти, які відносяться до цього проєкту.

По перше варто описати змінні середовища, які необхідні для можливості запуску сервера:

- JWT_SECRET – секретний ключ для кодування JWT токенів;
- JWT_ACCESS_EXPIRES_IN – час життя JWT токена;
- JWT_ACCESS_COOKIES_MAX_AGE – час життя JWT токена в куках;
- JWT_REFRESH_EXIPRES_IN – час життя рефреш токену;
- JWT_REFRESH_COOKIES_MAX_AGE – час перебування рефреш токена в куках;
- JWT_SESSION_EXPIRES_IN – час життя токена при умові, що користувач бажає бути авторизованим лише протягом сесії;
- CLIENT_ENDPOINT – кінцева точка клієнтської частини для ПК;
- CLIENT_MOBILE_ENDPOINT – кінцева точка клієнтської частини для мобільних пристроїв та планшетів;
- ADMIN_ENDPOINT – кінцева точка панелі адміністратора;
- PORT – порт, на якому буде запущено сервер;
- HOST – доменне ім’я, на якому буде запущено сервер;
- PROTOCOL – протокол з’єднання.

Після цього необхідно ініціалізувати головну змінну, з якої буде запускатися сервер зі всіма зареєстрованими плагінами та роутами. Реєстрація усіх необхідних модулів зображена на рисунку 3.23.

```
instance
    .register(fastifyCors, {
        origin: [
            clientEnpoint,
            clientMobileEndpoint,
            adminEndpoint,
        ],
        credentials: true,
    })
    .register(fastifyStatic, {
        root: path.join(__dirname, uploadsPath),
        prefix: '/uploads/',
    })
    .register(require('fastify-env'), { schema })
    .register(fp(errorHandler))
    .register(fp(authenticator))
    .register(fp(decorateFastifyInstance))
    .register(fastifyFormBody)
    .register(fastifyCookie)
    .register(require('fastify-file-upload'))
    .register(authController, { prefix: 'api/auth/' })
    .register(adminAuthController, { prefix: 'api/auth/admin' })
    .register(subjectController, { prefix: 'api/' })
    .register(subjectConfigController, { prefix: 'api/' })
    .register(testSuiteController, { prefix: 'api/' });

export { instance };
```

Рисунок 3.23 – Реєстрація модулів для коректної роботи сервера

У списку можна побачити fastifyCors. Він необхідний для того, щоб вказати список сайтів, які можуть взаємодіяти з даним сервером та його API.

FastifyStatic потрібно для того, щоб сервер міг взаємодіяти з файлами та надати можливість отримати ці файли звертаючись за певними url-адресами.

FastfiEnv призначене для того, щоб отримати змінні середовища при запуску сервера.

FastifyCookie – для зручної роботи з куками.

FastifyFormBody – для підтримки запитів типу form-data та xxx-url-encoded.

Кодування та декодування JWT токена відбувається завдяки плагіну fastify-jwt. Клас, який відповідає на генерацію та верифікацію токена зображено на рисунку 3.24.

```
class AccessService {
    instance!: FastifyInstance;

    constructor(fastify: FastifyInstance) {
        this.instance = fastify;
    }

    async generateToken(userProfile: UserProfile, session: boolean = false): Promise<string> {
        const expiresIn = session
            ? this.instance.config.JWT_SESSION_EXPIRES_IN
            : this.instance.config.JWT_ACCESS_EXPIRES_IN;

        const token = this.instance.jwt.sign(_.omit(userProfile, ['iat', 'exp']), {
            expiresIn: Number(expiresIn),
        });

        return token;
    }

    async verifyToken(token: string): Promise<UserProfile> {
        const userProfile: UserProfile = this.instance.jwt.verify(token);

        if (!userProfile) {
            throw new HttpErrors.Unauthorized('Профіль користувача відсутній.');
        }

        return userProfile;
    }
}
```

Рисунок 3.24 – Код класу який генерує та верифікує jwt-токени

Генерація рефреш токена аналогічна звичайному JWT токену, за виключенням того, що він має час життя 30 днів та зберігається в базі даних. Архітектура розроблена таким чином, що створена можливість одночасної автентифікації та авторизації з декількох пристроїв.

При реєстрації нового користувача паролі зберігаються у кодованому вигляді за допомогою bcryptjs. Пароль хешується на основі початкового паролю та «солі», а при спробі автентифікації порівняється пароль введений користувачем та хешований пароль, який зберігається в базі даних.

Це захищає від можливості автентифікації хакером при умові, якщо інформація з БД була викрадена. Клас який відповідає за хешування та порівняння паролів зображено на рисунку 3.25.

```
class BcryptHasher implements PasswordHasher {
    async hashPassword(password: string): Promise<string> {
        const salt = await genSalt(BCRYPT_HASHER.BCRYPT_ROUNDS);
        return hash(password, salt);
    }

    async comparePasswords(providedPass: string, storedPass: string): Promise<boolean> {
        const isMatched = await compare(providedPass, storedPass);
        return isMatched;
    }
}
```

Рисунок 3.25 – Код класу для хешуваня та верифікації паролів користувачів

3.6 Тестування

Тестування дуже важливий етап розробки. Він необхідний для того, щоб звірити чи розроблений застосунок відповідає умовам, які були визначені раніше. У нашому випадку головним буде тестування користувацького інтерфейсу.

При реєстрації нового користувача є правила заповнення полів, а саме:

- email повинен мати правильну структуру виду xxx@xxx;
- пароль повинен бути не коротший за 8 символів;
- повторний пароль має співпадати з паролем введеним вище.

Для того щоб перевірити ці умови, варто лише ввести некоректні дані. Після цього інтерфейс одразу позначить поля, які заповнено неправильно. Результат тестування з неправильними даними форми реєстрації зображено на рисунку 3.26.

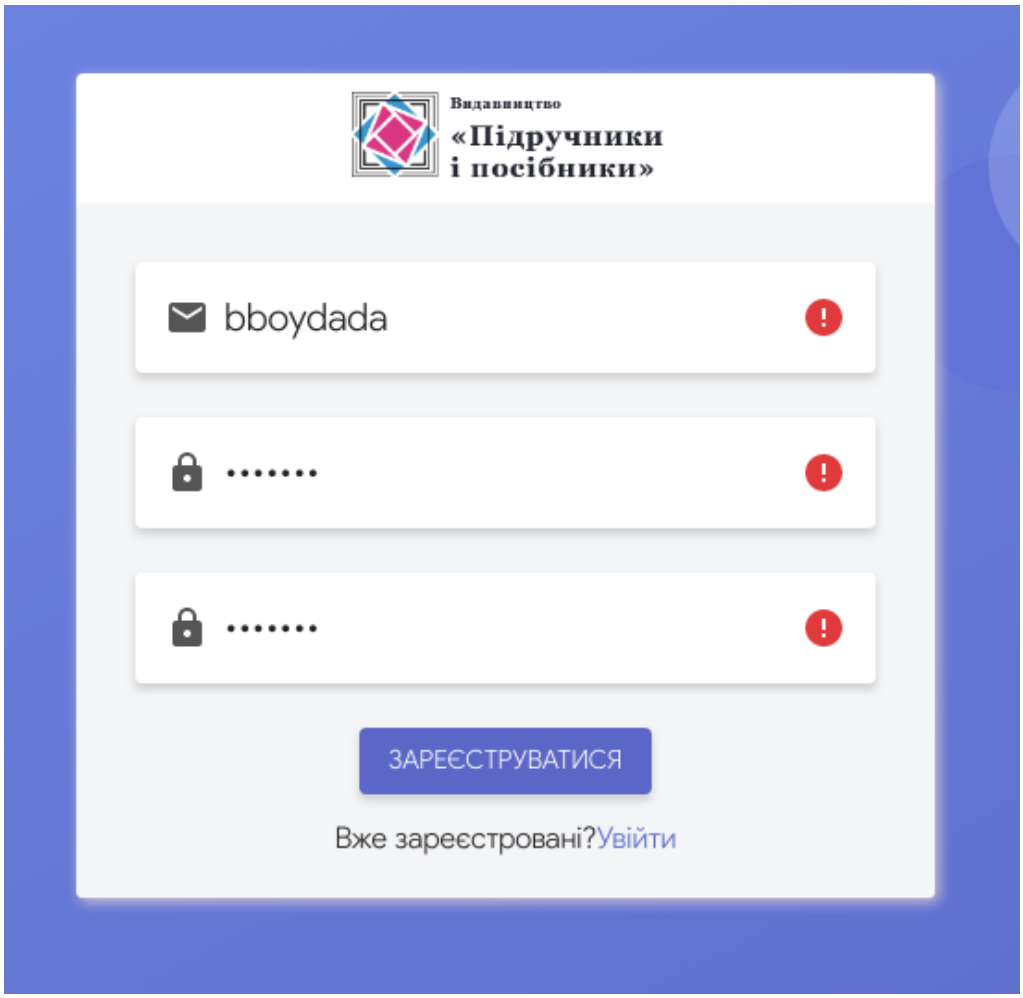

Рисунок 3.26 – Результат тестування форми для реєстрація з некоректними даними

При умові, що усі дані введені правильно і користувач успішно зареєструвався, він має бути автоматично переадресований на сторінку автентифікації. При цьому, емеїл який було використано під час реєстрації автоматично підставляється у відповідне поле форми для верифікації.

Форма верифікації також має перевірку на правильність емейлу та довжину пароля, проте це вже було протестовано у формі реєстрації і оскільки алгоритм аналогічний, немає сенсу для тестування. Більш важливим у цьому випадку буде перевірка при введені даних неіснуючого користувача або існуючого але з неправильним паролем.

Спочатку потрібно протестувати як зреагує система при вводі даних неіснуючого користувача. Результат тестування буде зображено на рисунку 3.27.

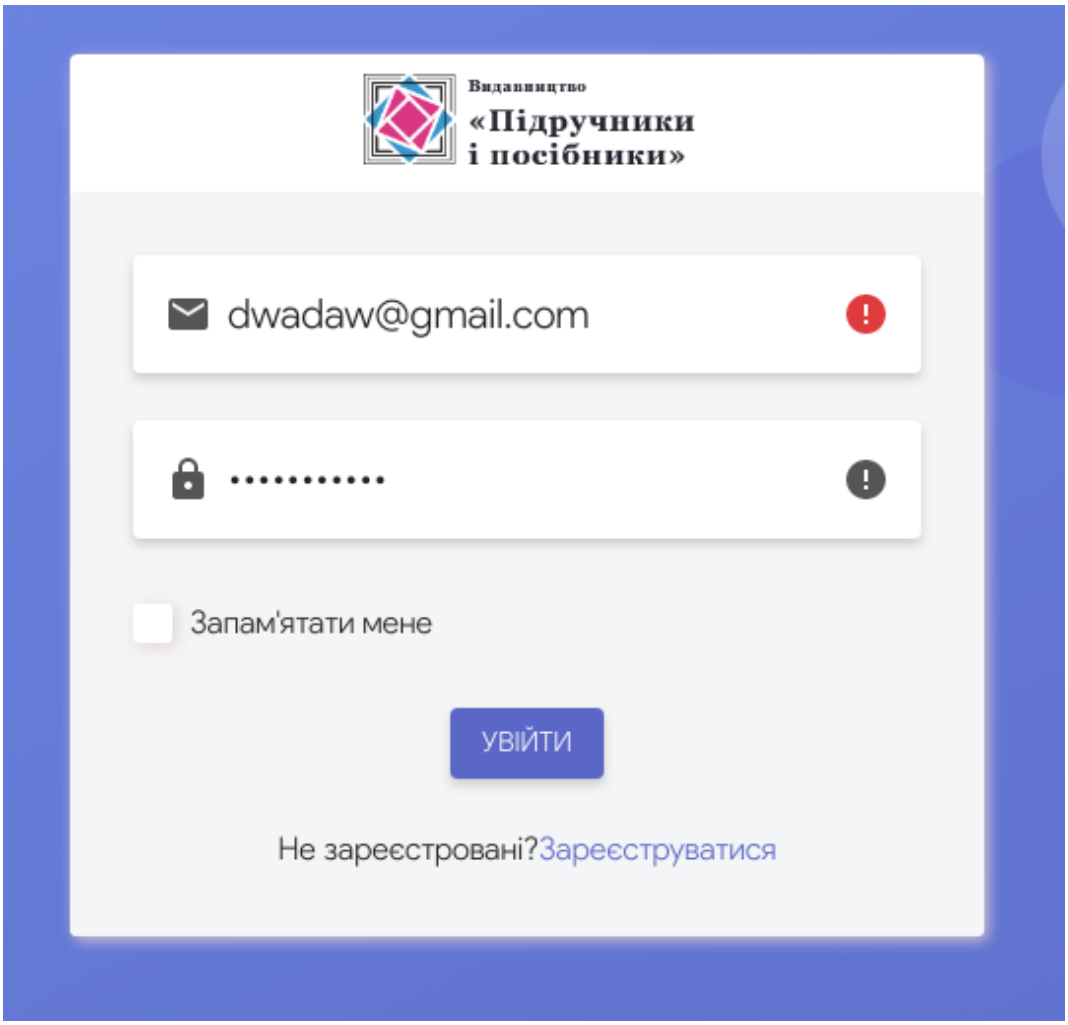

Рисунок 3.27 – Результат тестування автентифікації з даними неіснуючого користувача

Після цього є необхідність протестувати дані існуючого користувача (email), але з неправильним паролем. Результат тестування буде зображено на рисунку 3.28.

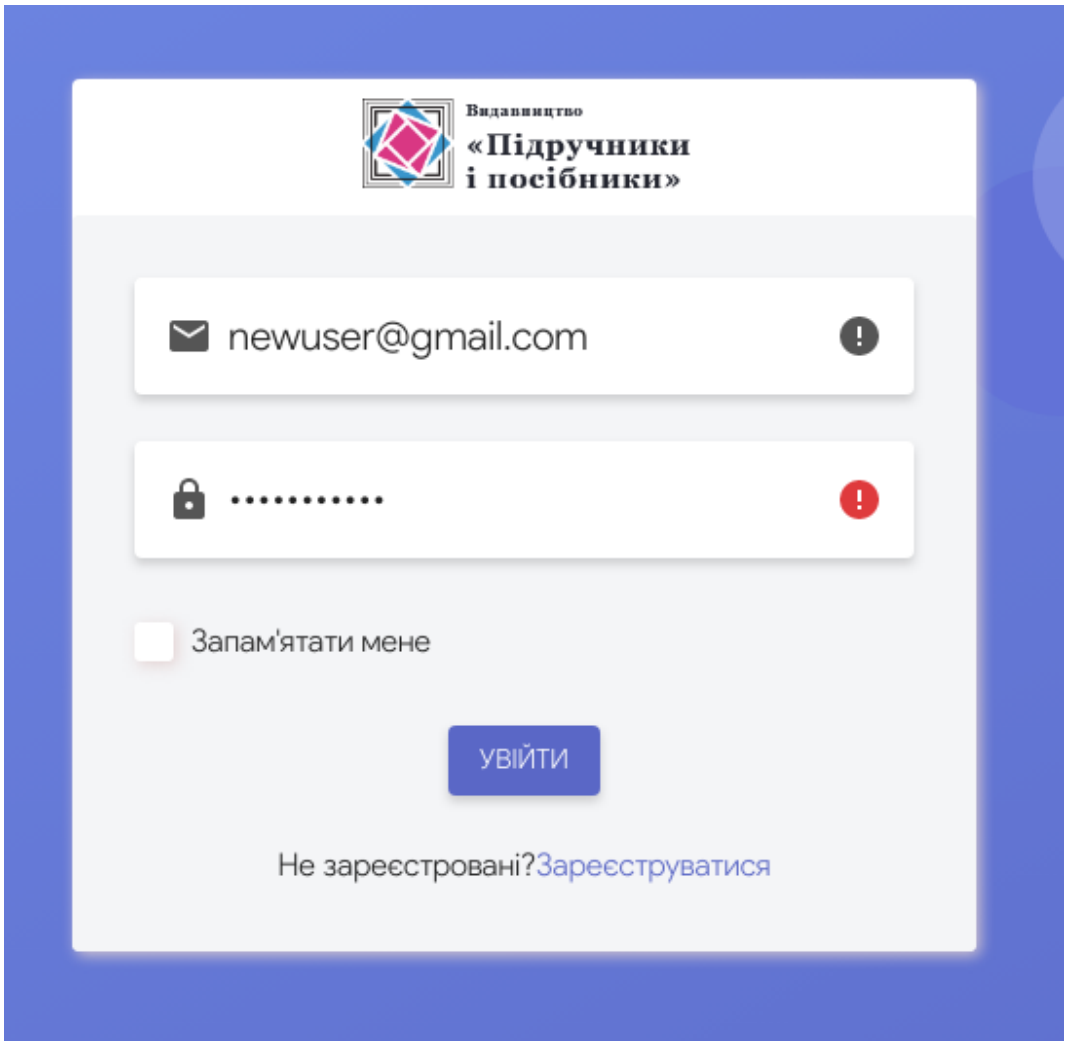

Рисунок 3.28 – Результат тестування автентифікації з неправильним паролем

В обох випадках система повинна показати, які поля були заповнені неправильно.

Окрім цього, є необхідність тестування реєстрації при умові, що користувач з таким емейлом вже зареєстрований. У такому випадку система має показати, що саме поле емейлу некоректне, та при наведені на іконку показати помилку. Результат тестування цього випадку буде зображено на рисунку 3.29.

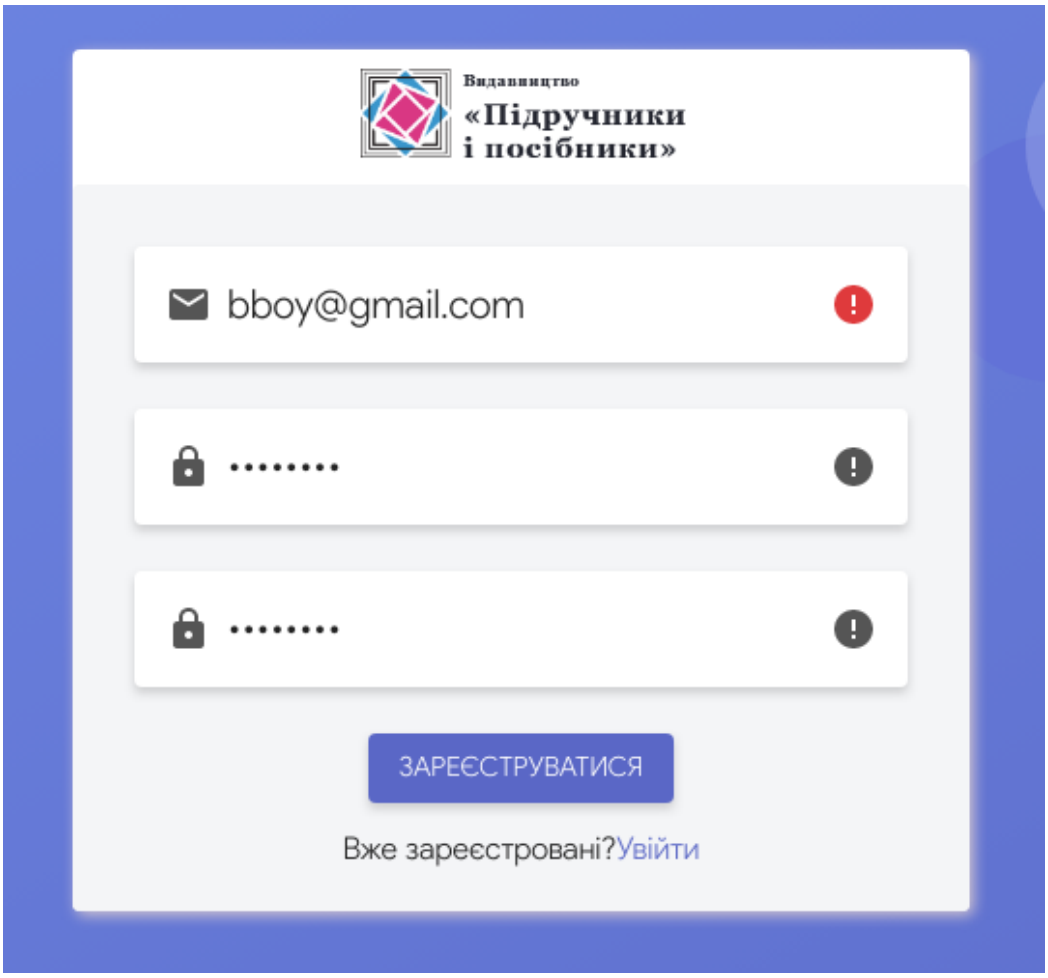

Рисунок 3.29 – Результат тестування реєстрації з вже зареєстрованим емейлом

Оскільки головний етап, а саме реєстрація та автентифікація були успішно протестовані, виникає необхідність протестувати наступне:

- вибір предмету;
- налаштування тесту;
- вибір відповідей;
- відображення статистики тесту.

Тестувати потрібно почергового, тобто почати з вибору предмету. Після того як користувач обирає бажаний предмет, він має бути переадресований на сторінку налаштування тесту. Результат тестування зображено на рисунку 3.30.

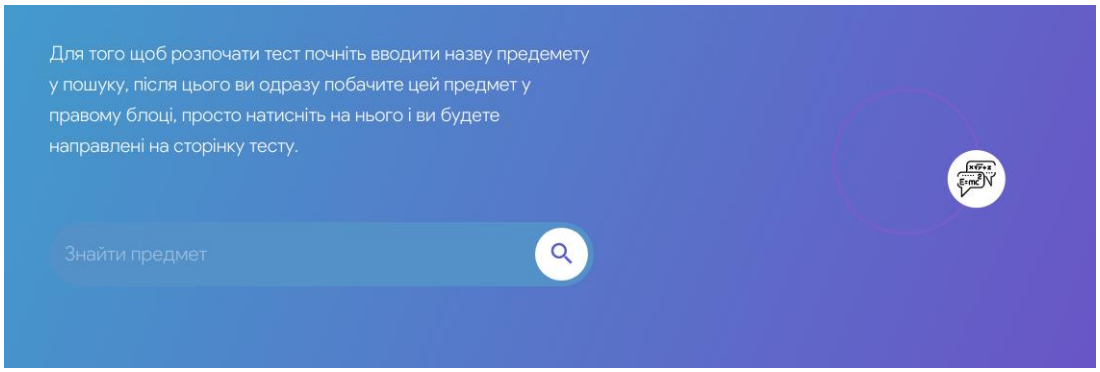

Рисунок 3.30 - Вибір предмету

Після цього виникає необхідність тестування налаштувань тесту, оскільки є велика кількість можливостей кастомізації, зокрема:

- вибір похідного предмету, як наприклад алгебра у математиці;
- вибір тесту по темам;
- вибір тестів наближених до ЗНО;
- похідний від попереднього – варіанти наближені до ЗНО;
- похідний від попереднього – варіанти попередніх сесій ЗНО;
- відображення правильних відповідей під час тесту;
- обмеження часу.

Для тестування селекції бажаного тесту варто протестувати лише можливість вибору та факт того, що при переході до тесту відображається ім’я відповідно до обраного раніше. На рисунку 3.31 зображено можливість конфігурації, завдяки випадаючим спискам на кнопках-перемикачах.

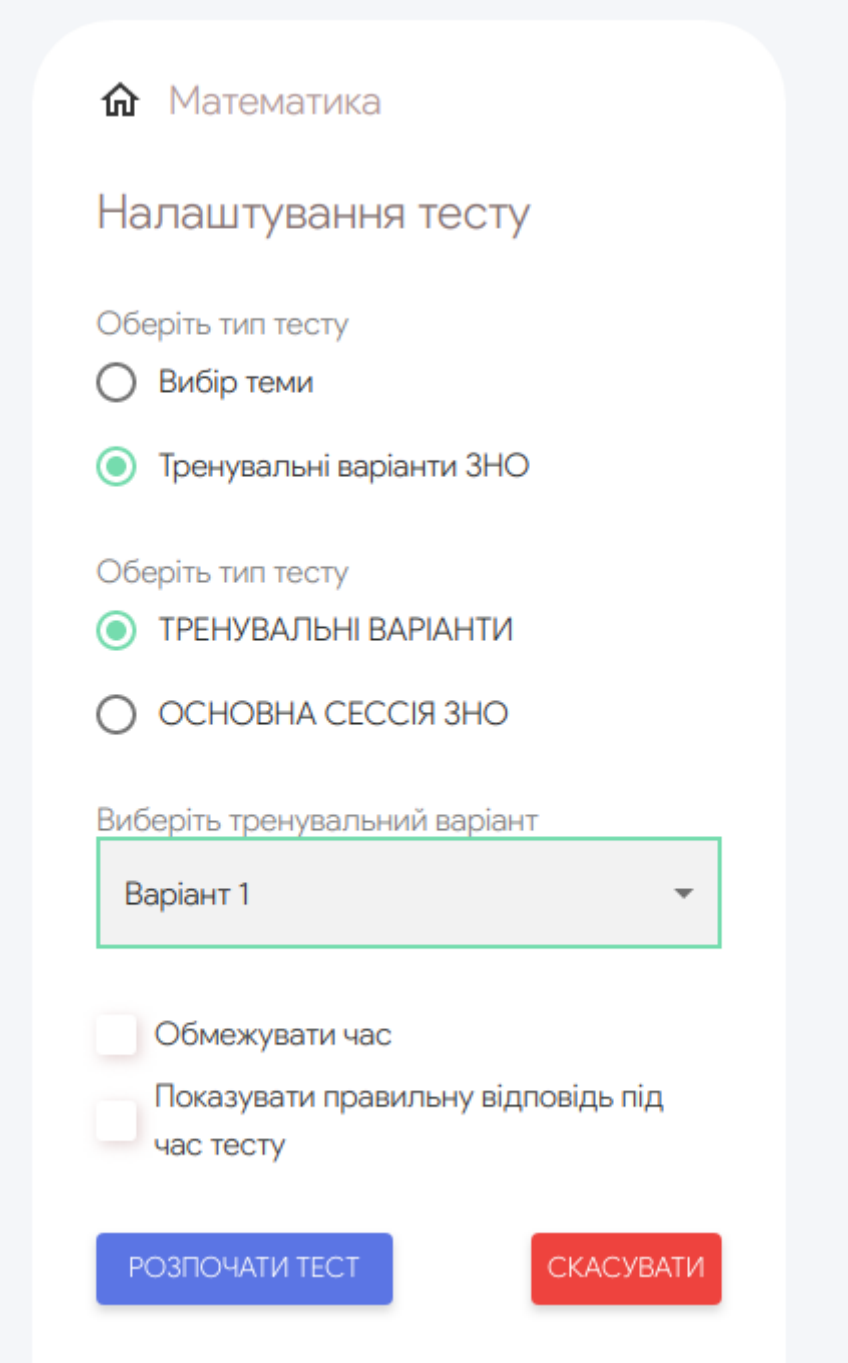

Рисунок 3.31 – Тестування конфігурації випадаючих списків

Після цього варто перевірити опції, які відповідають за відображення правильних та некоректних відповідей під час тесту та за обмеження часу.

Почнемо з перевірки поведінки сервісу, якщо опція відображення правильної відповіді не відображена. Для цього необхідно вибрати відповідь та натиснути кнопку «відповісти». Повинна бути зображена лише обрана відповідь, а квадрат з індексом завдання позначений зеленим кольором.

Результат тестування зображено на рисунку 3.32.

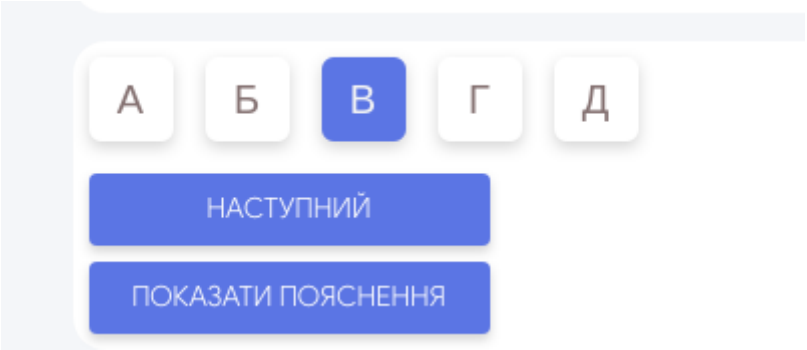

Рисунок 3.32 – Тестування вибору відповіді без опції відображення правильної відповіді під час тесту

Наступним буде перевірка з умовою, що опція відображення правильної відповіді обрана. Поведінка повинна бути наступною: після вибору відповіді користувачем та кліком на кнопку «відповісти» одразу зеленим буде показано правильну відповідь. У випадку, якщо обрана користувачем відповідь неправильна - вона буде підсвічена червоним кольором і квадрат з індексом обраного завдання бути аналогічного кольору. Тому потрібно протестувати два випадкт.

Результат тестування з правильною відповіддю зображено на рисунку 3.33.

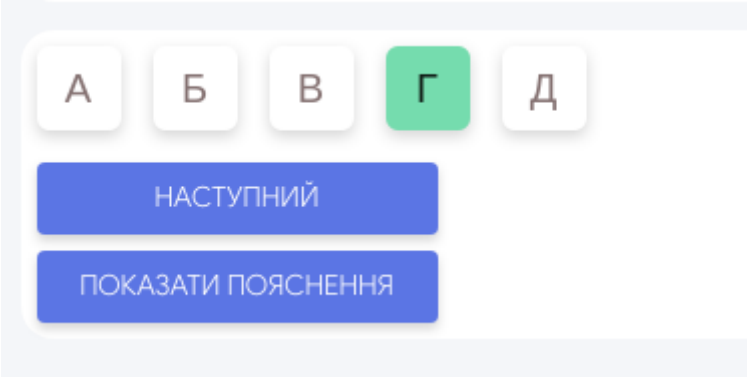

Рисунок 3.33 – Результат тестування з обраною опцією відображення правильної відповіді під час тесту з умовою, що користувач обрав правильну відповідь

Результат тестування з некоректною відповіддю зображено на рисунку 3.34.

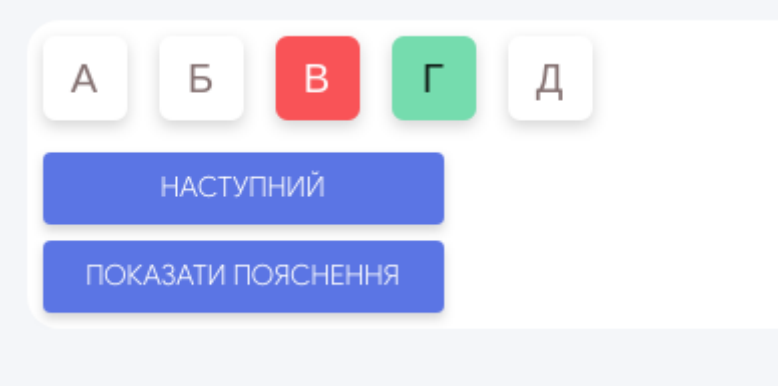

Рисунок 3.34 - Результат тестування з обраною опцією відображення правильної відповіді під час тесту з умовою, що користувач обрав некоректну відповідь

Наступним крок - тестування опції обмеження часу. Якщо опція позначена як не обрана, таймер не повинен бути відображений в інтерфейсі користувача. У протилежному випадку він повинен бути відображений напроти назви тесту, який проходить користувач.

Результат тестування без обраної опції обмеження часу зображено на рисунку 3.35.

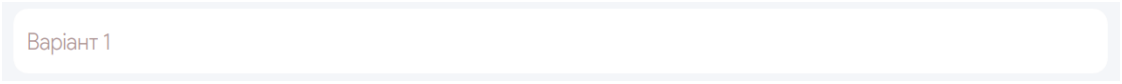

Рисунок 3.35 – Результат тестування без обраної опції обмеження часу

Результат тестування з обраною опцією обмеження часу зображено на рисунку 3.36.

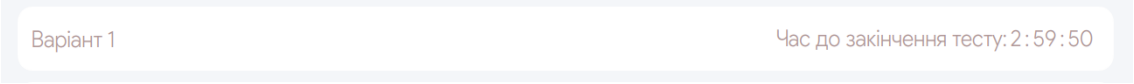

Рисунок 3.36 – Результат тестування з обраною опцією обмеження часу

4 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

4.1 Аналіз ринку

Для переходу між рівнями освіти в Україні існують певні іспити. Їх метою є перевірка рівня знань учнів шкіл, закладів вищої освіти та майбутніх абітурієнтів. Існує два основних державних екзамени - Державна підсумкова атестація (далі – ДПА) та ЗНО [3]. Перший учні здають наприкінці дев’ятого класу для фіксації знань та подальшого допуску до навчання у 10-11 класах. ДПА – це форма контролю відповідності освітнього рівня випускників загальноосвітніх навчальних закладів І-ІІІ ступенів, коледжів та професійно-технічних закладів, що надають загальну середню освіту.

Наступний важливий екзамен - це ЗНО. Система ЗНО в Україні створювалася та вдосконалювалася протягом багатьох років. Основне її завдання - забезпечити кожному рівні умови доступу до вищої освіти. Ця система мінімізує корупційні ризики і дає можливість талановитим дітям реалізувати свій потенціал.

Оскільки ЗНО – це дуже важливий рубіж у житті учнів для подальшого навчання, його потрібно успішно здати. Для цього необхідно підготуватися усіма можливими способами. Один з найпопулярніших у нашому суспільстві є особистий репетитор. Цей спосіб підготовки є досить ефективним, оскільки наставник приділяє весь час заняття одному учню, завдяки чому останній отримує максимальну кількість знань та має можливість зворотного зв’язку. Однак, знайти професійного репетитора не так легко та й вартість такого навчання доволі висока.

У наш час все більшої популярності набуває онлайн підготовка. Є чимало сервісів, які надають послуги онлайн-уроків за певну оплату, але при цьому набирається уже група осіб і замість того, щоб один учень був закріплений за одним викладачем, має місце ситуація, коли уже 5-10 учнів закріплені за викладачем, що відповідно зменшує виділений час на кожного учня, а тому зменшує кількість отриманих знань.

Одним із фаворитів є сайт zno-osvita.ua. Він представляє з себе застосунок для проходження тестів ЗНО онлайн та підготовки майбутніх абітурієнтів для проходження оцінювання. Спосіб виконання всіх тестових завдань на сайті максимально наближений до реальних тестів, а форма надання відповіді відповідає виду, який пропонується абітурієнтам у бланку відповідей під час проходження тестів ЗНО. Одним з його недоліків є те, що він обмежений лише сесіями ЗНО, тобто не має можливості підготовки з певних дисциплін та тем.

4.2 Розрахунок витрат на проєктування

Продуктивність та мотивація праці насамперед забезпечуються формуванням матеріальних стимулів, головною формою яких є оплата праці.

Оплата праці - це будь-який заробіток, який обчислюється у грошовому виразі, який власник або уповноважений ним орган виплачує працівникові за виконану роботу чи надані послуги відповідно до умов трудового договору.

За законодавством України (ст. 1 Закону України «Про оплату праці») заробітна плата — це винагорода, обчислена, як правило, у грошовому виразі, яку за трудовим договором роботодавець виплачує працівнику за виконану ним роботу [4].

Ознаками заробітної плати є:

- це винагорода за виконання працівником трудових обов'язків, розмір якої залежить від складності та умов виконуваної роботи, професійних та ділових якостей працівника, результатів його роботи, господарської діяльності підприємства в цілому;
- це винагорода, розмір якої визначається за наперед встановленими нормами і розцінками;
- це винагорода, що має гарантований характер.

Заробітна плата має регулярно виплачуватися у строки, встановлені у колективному договорі. Розмір її не може бути нижчим ніж мінімальний розмір оплати праці, визначений законодавством.

Мінімальна заробітна плата — це законодавчо встановлений розмір

заробітної плати за просту, некваліфіковану працю, нижче якого не може встановлюватися оплата за виконану працівником місячну норму робіт Згідно п.5 ст.38 Бюджетного кодексу України [5] розмір мінімальної заробітної плати визначається в Законі про Державний бюджет на відповідний рік. Наразі місячна мінімальна зарплата в Україні складає 4723 грн.

Водночас при нарахуванні заробітної плати враховується і таке поняття як мінімальний посадовий оклад (тарифна ставка), який встановлюється у розмірі, не меншому за прожитковий рівень, визначений для працездатних осіб на 1 січня календарного року. Таке правило задекларованe у ч. 6 ст. 6 Закону України «Про оплату праці»

У 2020 році величина мінімального посадового окладу складає 2102 грн., оскільки саме такий розмір прожиткового мінімуму для працездатних осіб мав місце станом на 01.01.2020 р.

Але, якщо працівнику, який виконав місячну норму праці, нарахували зарплату в меншому розмірі, ніж мінімальна заробітна плата, роботодавець проводить доплату до її рівня, яку виплачує разом із зарплатою (ч. 1 ст. 3^1 Закону України «Про оплату праці»)

Розрізняють основну, додаткову заробітну плату та інші заохочувальні і компенсаційні виплати.

За положеннями ст. 2 Закону України «Про оплату праці» основна заробітна плата це винагорода за виконану роботу відповідно до встановлених норм праці. Вона може встановлюватися у вигляді відрядних розцінок, тарифних ставок або посадових окладів.

У свою чергу, додаткова заробітна плата являє собою винагороду за працю понад встановлені законодавством, колективним чи трудовим договором норми праці за трудові успіхи та винахідливість, а також за роботу в особливих умовах праці.

Інші заохочувальні, компенсаційні виплати можуть встановлюватися у формі винагород за підсумками роботи за рік, премій за спеціальними системами і положеннями, компенсаційних та інших грошових і матеріальних

виплат, які не передбачені актами чинного законодавства або які виплачуються понад встановлені норми. Це можуть бути: надбавки і доплати, які не передбачені законодавством; оплата простоїв не з вини працівника; суми наданих підприємством трудових і соціальних пільг; винагорода за підсумками роботи за рік; одноразові заохочення; матеріальна допомога тощо.

Джерелами виплати заробітної плати для підприємств є частина доходу, для установ та організацій, які фінансуються з бюджету — кошти відповідних бюджетів, а також частина доходу, одержаного внаслідок господарської діяльності.

До створення сервісу у вигляді веб-застосунку було залучено два працівники за різними посадами і кваліфікацією. Розрахунок їх заробітної плати наведений нижче.

Працівнику №1 нараховані за повний відпрацьований місяць 6456 грн. Податкова соціальна пільга до такої заробітної плати не застосовується, оскільки вона більша за граничний розмір доходу, який дає право на податкову соціальну пільгу.

1) Рахуємо податок на доходи фізичних осіб: 6456 х 18% (ставка податку на доходи фізичних осіб) = 1162,08грн.

2) Рахуємо військовий збір: 6456 х 1,5% (ставка військового збору) = 96,84 грн.

3) Рахуємо єдиний внесок: 6456 х 22 % (ставка ЄСВ) = 1420,32 грн.

4) Утримання – 1258,92 грн. (1162,08 грн. + 96,84 грн.)

5) До виплати працівникові – 5100,48 грн. (6456,24 грн. – 1258,92 грн. – 96,84 грн.)

Працівникові №2 нараховані за повний відпрацьований місяць 5047 грн. Податкова соціальна пільга не застосовується, ідентично першому співробітнику.

1) Рахуємо податок на доходи фізичних осіб: 5047 х 18% (ставка податку на доходи фізичних осіб) = 908,46 грн.

2) Рахуємо військовий збір: 5047 х 1,5% (ставка військового збору) = 75,71 грн.

3) Рахуємо єдиний внесок: 5047 х 22 % (ставка ЄСВ) = 1110,34 грн.

4) Утримання – 984,17 грн. (908,46 грн. + 75,71 грн.)

5) До виплати працівникові – 4063,73 грн. (5047,90 грн. – 908,46 грн. – 95,71 грн.)

Плата за працю робітникам зображена у таблиці 4.1

Таблиця 4.1 – Розрахування заробітної плати

Посада	Оклад	Відрахування	Кількість		Сума
Інженер І категорії	6456 грн./міс.	1258 грн./міс.	1 чол.	1 міс.	6456 грн.
Технік	5047 грн./міс.	984 грн./міс.	1 чол.	1 міс.	5047 грн.
Усього зарплати:			11503 грн.		

Контрагентські роботи становлять 11503 х 11% = 1265,33 грн.

Співробітники не були у відрядженні, тому витрат на відрядження немає.

Інші прямі витрати за місяць становлять – 4601 грн. (11503 грн. х 0,4).

Усього прямих витрат за місяць 19611 грн. (11503 грн.+2242 грн.+ 1265 грн.+ 4601 грн.).

Накладні витрати за місяць становлять 6863 грн. (19611 грн. х 0,35).

Планові накопичення визначається у відсотках (20-30%) від суми прямих і накладних витрат. Планові накопичення за місяць становлять 4842 грн. ((4601 грн.+ 19611 грн.) х 0,2).

Усього кошторисна вартість проєкту 31316 грн. (19611 грн.+ 6863 грн.+ 4842 грн.).

Податок на додану вартість становить 6263 грн. (31316 грн. х 0,2)

Договірна ціна становить 37579 грн. (31316 грн.+ 6263 грн.)

Загальний кошторис витрат на проєктування наведено у таблиці 4.2.

Таблиця 4.2 - Кошторис витрат на проєктування

Найменування статей витрат	Сума, грн
1. Зарплата проєктувальників.	11503
2. Відрахування на соціальні потреби.	2242
3. Контрагентські роботи і послуги.	1265
4. Витрати на відрядження.	0
5. Інші прямі витрати.	4601
6. Усього прямих витрат.	19611
7. Накладні витрати.	6863
8. Планові накопичення.	4842
9.Усього, кошторисна вартість проєкту.	31316
10. Податок на додану вартість.	6263
11. Загалом, договірна ціна розробки	37579

4.3 Обґрунтування необхідності розробки

У наші дні, питання тестування та підготовки учнів та майбутніх абітурієнтів до ЗНО дуже важливе, адже воно визначає їх майбутнє.

Після аналізу ринку онлайн сервісів тестування, було визначено, що головні критерії для зручності користування, це простота інтерфейсу, його інтуїтивна зрозумілість, а також можливість доступ з будь якої точки світи при наявності підключення до мережі Інтернет.

Одні з головних недоліків існуючих сервісів, це їх мала кількість та примітивність. Один з найбільш популярних обмежується тільки існуючими сесіями ЗНО. Цього недостатньо для отримання достатньої кількості знань та впевненості користувача у тому, що він готовий до здачі ЗНО. Тому було вирішено розробити власний сервіс, який окрім попередніх сесій буде містити також додаткові тести для можливості проходження по темам вибраного предмету або пройти подібні до ЗНО унікальні варіанти.

ВИСНОВКИ

В результаті виконання завдань дипломного проєктування було спроєктовано, реалізовано та протестовано інформаційну систему тестування знань учнів з метою підготовки до ЗНО. В процесі роботи було проаналізовано необхідні технологій проєктування та обрано засоби реалізації програмного забезпечення, а також досліджено питання захисту персональних даних користувачів інформаційної системи. Завдяки звіту можна детально розібрати всі етапи реалізації онлайн сервісу.

Перш за все, на етапі розробки було проаналізовано існуючі рішення та здійснено постановку завдання. Крім цього було визначено необхідність та актуальність теми дипломного проєктування, її значення та ціль.

На наступному кроці головним завданням була формалізація вимог до майбутнього програмного забезпечення, спроєктовано його архітектуру, загальну концепцію та алгоритми роботи.

На третьому етапі було детально проаналізовано можливі технології для реалізації сервісу та вибрано ті, які найкраще підходять до поставленого завдання. Окрім цього було представлено рівні реалізації серверної та клієнтської частинами.

На останньому етапі було сформоване техніко-економічне обґрунтування, в якому проаналізовано ринок, проведено розрахунок витрати та необхідність розробки продукту.

В майбутньому планується покращення системи шляхом впровадження нового функціоналу, такого як, наприклад, особистий кабінет користувача та ведення загальної статистики.

Дана тема та її дослідження були доведені до відома під час виступу на «Днях Науки – 2020» на базі Галицького коледжу імені В’ячеслава Чорновола. Також стаття на дану тему опублікована у матеріалах міжнародної студентської наукової конференції «Модернізація та сучасні українські та світові наукові дослідження» (29 травня 2020р. м. Львів).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Зовнішнє незалежне оцінювання в освіті України. Курс лекцій: навч. посіб. / Г.С. Кашина, В.П. Сергієнко. Луцьк, 2010. 115 с. URL: http://moodle.ndu.edu.ua/pluginfile.php/889/mod_page/content/1/ZNO.pdf. (дата звернення: 12.05.2020)
2. І. В. Ставицька. Особливості організації та проведення вебінарів у технічному університеті. URL: http://www.kamts1.kpi.ua/node/1575. (дата звернення: 20.05.2020)
3. Державна підсумкова атестація (2020). URL: https://mon.gov.ua/ua/tag/derzhavna-pidsumkova-atestatsiya (дата звернення: 20.05.2020)
4. Закон України «Про оплату праці» від 24 березня 995 року, №108/95-ВР. URL: https://zakon.rada.gov.ua/laws/show/108/95-%D0%B2%D1%80 (дата звернення: 01.06.2020)
5. Бюджетний кодекс України від 8 липня 2010 року, №2456-VI. URL: https://zakon.rada.gov.ua/laws/show/2456-17 (дата звернення: 09.06.2020)
6. Документація Node.JS. URL: https://nodejs.org/dist/latest-v12.x/docs/api/ (дата звернення: 25.05.2020)
7. Документація Fastify. URL: https://www.fastify.io/docs/latest/ (дата звернення: 24.05.2020)
8. Документація TypeScript. URL: https://www.typescriptlang.org/docs/home.html (дата зверненя: 20.05.2020)

ВІДГУК
на дипломний проект
освітньо-кваліфікаційного рівня «молодший спеціаліст»
зі спеціальності 5.05010101 «Комп'ютерні науки та інформаційні технології»

студента

Галицького коледжу імені В'ячеслава Чорновола

Безрукова Олександра

на тему

«Інформаційна система з тестування знань учнів з метою підготовки до ЗНО»

Дипломний проект присвячений вирішенню практичної задачі – реалізація онлайн-системи з тестування знань учнів з метою підготовки до ЗНО. Реалізація поставленої задачі дозволить учням та абітурієнтам краще підготуватися до проходження ЗНО чи інших видів тестування.

У процесі роботи над дипломним проектом автор добре розібрався у сучасних підходах, технологіях та засобах реалізації онлайн-систем, проаналізував типові платформи з метою розробки архітектури та бізнес-логіки сервісу для тестування знань, реалізував власну інформаційну систему з тестування знань учнів. Вчасно і достатньо самостійно виконував план дипломного проектування. Зарекомендував себе як хороший спеціаліст у галузі розробки веб-базованих систем.

Під час виконання плану дипломного проекту студент продемонстрував вміння використовувати інформаційні джерела, вести пошук інформації в Інтернеті, ставити та вирішувати фахові завдання.

У цілому дипломний проект виконаний на належному для присвоєння відповідної фахової кваліфікації рівні.

Керівник дипломного проекту:

Павлюс В.П.,
викладач ЦК професійної та
практичної підготовки
спеціальності «Обслуговування
програмних систем і комплексів»

РЕЦЕНЗІЯ
на дипломний проект
освітньо-кваліфікаційного рівня «молодший спеціаліст»
зі спеціальності 122 «Комп'ютерні науки та інформаційні технології»

студента
Галицького коледжу імені В'ячеслава Чорновола
Безрукова Олександа Олександровича

на тему ***«Інформаційна система тестування знань учнів з метою підготовки до ЗНО»***

Дипломний проект, представлений на рецензію, містить завдання на дипломне проектування, пояснювальну записку на 71 сторінці, схеми та таблиці на 2 додатках.

Зміст дипломного проекту повністю відповідає назві та поставленому завданню. Розкрито актуальність теми, продемонстрована практична значущість.

При реалізації дипломного проекту обґрунтовано вибрані технології та програмні засоби реалізації інформаційної системи.

Пояснювальна записка написана чітко та ясно. Автор мотивує своє рішення щодо архітектури, демонструє високу програмістську кваліфікацію.

До недоліків можна віднести недостатній перелік використаної літератури: відсутні посилання на електронні ресурси та матеріали наукових досліджень даної тематики. У цілому дипломний проект виконаний на високому рівні і заслуговує високої оцінки.

Рецензент: *Глинська М.Л.,*
викладач ЦК інформатики та
комп'ютерних дисциплін

UNICHECK

Ім'я користувача:
Наталя Кульчинська

ID перевірки:
1004112668

Дата перевірки:
18.06.2020 09:05:47 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
16.02.2021 10:18:34 EET

ID користувача:
100004382

Назва документа: **ДП_Безруков_К47_перевірка**

Кількість сторінок: **72** Кількість слів: **9894** Кількість символів: **74496** Розмір файлу: **2.28 MB** ID файлу: **1004125423**

5.58%
Схожість

Найбільша схожість: **0.93%** з Інтернет-джерелом (https://zno.osvita.ua)

5.58% Джерела з Інтернету	122	Сторінка 74

Не знайдено джерел з Бібліотеки

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

5.53%
Вилучень

Деякі джерела вилучено автоматично (фільтри вилучення: кількість знайдених слів є меншою за **8 слів** та **0%**)

4.08% Вилучення з Інтернету	282	Сторінка 75
2.4% Вилученого тексту з Бібліотеки	10	Сторінка 76

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	1