МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Львівський національний університет імені Івана Франка

Факультет електроніки та комп'ютерних технологій

Кафедра системного проектування

Допустити до захисту
Завідувач кафедри
_____ доц. Шувар Р.Я.
«___» _____2023 р.

Кваліфікаційна робота

Бакалавр
(освітній ступінь)

Розробка застосунку для тренування швидкого мислення з використанням інтерактивного опитування

Виконала:

студентка групи ФЕП – 41

спеціальності 121 – Інженерія програмного забезпечення

_____ **Андрейків І.В.**

Науковий керівник:

_____ **доц. Демків Л.С.**

«____» _____2023 р.

Рецензент:

_____ **проф. Монастирський Л.С.**

Львів 2023

АНОТАЦІЯ

Дана бакалаврська робота присвячена розробці застосунку, спрямованого на тренування навичок швидкого мислення, за допомогою інтерактивного опитування. Основна ідея полягає у створенні цікавої та зручної для користувача платформи, яка сприяє розвитку здібностей до швидкого мислення. Додаток поєднує елементи інтерактивного опитування зі зручним інтерфейсом, сприяючи покращенню критичного мислення та здатності приймати швидкі та обґрунтовані рішення.

Дослідження, проведені для цього проєкту, включали вивчення когнітивних процесів і технік, а також огляд і аналіз існуючих програм та додатків, які сприяють розвитку швидкого мислення. Крім цього, було проаналізовано сучасні технології, з метою оптимального вибору ефективних інструментів. На основі отриманих даних було розроблено концепцію та функціональні вимоги, використовуючи сучасні веб-технології, такі як HTML, CSS і JavaScript, для створення інтерфейсу користувача, а також фреймворки та бібліотеки, зокрема React і Express.js, для реалізації функціональності застосунку.

Результати цієї роботи мають різноманітне застосування, зокрема в навчальних закладах для підвищення академічної успішності та удосконалення когнітивних навичок. Крім того, додаток може бути корисним для людей, які хочуть покращити психологічну стійкість до стресових ситуацій.

В цілому, ця кваліфікаційна робота підкреслює важливість розробки навчальних застосунків, та звертає увагу на необхідність створення ефективних інструментів для тренування швидкого мислення.

ABSTRACT

This bachelor's work is dedicated to developing an application, dedicated to training quick thinking skills by completing an interactive quiz. The main goal is website creation, which will be interesting and practical for users wanting to train quick thinking skills. The application presents an interactive quiz with a convenient interface, improving the development of users critical thinking and ability to accept quick and informed decisions.

Researches conducted for this project aim to learn cognitive processes and techniques and review already existing applications, which develop quick thinking skills. Furthermore, an analysis was performed in order to come up with optimal selection of both effective and modern tools and technologies. Based on this data, concepts and functional requirements were conducted which resulted in using modern web technologies, such as HTML, CSS, and JavaScript for creating a user interface and frameworks as well as libraries, mainly React and Express.js, for implementation of application functionality.

This work results can have different usage, in particular, they can be used in educational institutions to improve academic performance and cognitive skills. Moreover, people, who want to improve their psychological resilience to stressful situations will find this application to be just as useful.

In conclusion, this scientific work makes an emphasis on importance of learning applications development and brings attention to the necessity of creating effective tools used for training quick thinking skills.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ 6
ВСТУП.....7
РОЗДІЛ 1 АНАЛІЗ СТАНУ ПРОБЛЕМНОЇ ОБЛАСТІ.....10
1.1 Користь швидкого мислення 12
1.2 Методи тренування швидкого мислення 13
1.3 Веб-застосунки для розвитку швидкості мислення. Їх переваги та недоліки.15
1.4 Сучасні технології та їх можливості 17
РОЗДІЛ 2 ОГЛЯД ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ.....21
2.1 Середовище розробки – Visual Studio Code.....21
2.2 Мова програмування JavaScript та її особливості 23
2.2.1 Переваги та недоліки JavaScript 24
2.3 Програмна платформа Node.js.....25
2.3.1 Переваги Node.js.....26
2.3.2 Недоліки Node.js.....26
2.4 Express.js 27
2.4.1 Особливості Express.js 28
2.5 Бібліотека React.js 29
2.5.1 React Native.....31
2.5.2 JSX.....32
2.6 База даних MongoDB 33
2.6.1 Переваги MongoDB.....33
2.6.2 Недоліки MongoDB.....35
РОЗДІЛ 3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ.....36
3.1 Верхній рівень стеку MERN – React.JS.....36
3.1.1 Реєстрація та авторизація користувачів 37
3.1.2 Створення сторінок сайту.....38
3.2 Серверна частина – Node.js та Express.js 41
3.2.1 Створення API для сторінок веб-застосунку 42

3.3 База даних MongoDB 47
РОЗДІЛ 4 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ВЕБ-ЗАСТОСУНКУ QUICKI.....49
4.1 Запуск веб-застосунку.....49
4.2 Взаємодія з веб-додатком 49
4.3 Відображення інформації у базі даних 52
ВИСНОВКИ 53
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 54
ДОДАТОК А. FrontCode.....56
ДОДАТОК Б. BackCode.....68

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення;

ШІ – штучний інтелект;

API – програмний прикладний інтерфейс;

CI/CD – неперервна інтеграція/ неперервна доставка;

CRUD – чотири базових функції для роботи із персистентним сховищем даних;

DB – база даних;

DOM – об’єктна модель документа;

HTML – мова розмітки гіпертексту;

HTTP – протокол передачі гіпертексту;

IDE – інтегроване середовище розробки;

IoT – інтернет речей;

JWT – веб токен;

MVC – модель, вигляд, контролер;

QA – забезпечення якості;

REST – передача репрезентативного стану;

SEO – оптимізація для пошукових систем;

SQL – мова структурованих запитів;

UI – призначений для користувача інтерфейс;

URL – уніфікований локатор ресурсу;

VCS – система керування версіями;

ВСТУП

У сучасному швидкоплинному та інформаційно насиченому світі здатність швидко мислити та приймати ефективні рішення є надзвичайно важливою. Розвиток когнітивних навичок, зокрема швидкого мислення, привертає значну увагу, оскільки люди прагнуть не відставати від вимог особистого та професійного життя.

Актуальність цієї роботи очевидна через потребу в збільшенні кількості навчальних інструментів, які б розвивали таку специфічну навичку, як швидке мислення. Оскільки люди щодня стикаються з різного роду проблемами, пов'язаними з прийняттям рішень, то здатність швидко аналізувати ситуації та розглядати їх з різних сторін, має вирішальне значення для досягнення успіху в різних сферах, включаючи освіту, бізнес та особисте життя.

Мета роботи – зробити свій внесок у сферу навчання, зосередившись на розробці застосунку, який пропонує цікаву та просту у використанні платформу для тренування навичок швидкого мислення. Поєднуючи елементи інтерактивного опитування та зручного інтерфейсу, додаток має на меті покращити критичне мислення та забезпечити зручність взаємодії користувача з програмою, що сприяє позитивному досвіду навчання та підвищує мотивацію до тренувань.

Об'єктом дослідження є розробка застосунку для тренування швидкого мислення. Основна увага приділяється проєктуванню та оцінці додатка з огляду на його зручність, ефективність та задоволеність користувачів.

Предметом дослідження є вивчення когнітивних процесів та технік, які допомагають у розвитку швидкого мислення. А також аналіз того, як ефективно використати зібрану інформацію для реалізації корисного продукту.

Методи дослідження, включають поєднання огляду літератури, аналізу когнітивної психології, та особистих вподобань користувачів – усе це допомагає правильно сформулювати перелік завдань, а також оцінити ефективність та

зручність використання додатку для майбутнього удосконалення. А дослідження процесу розробки програмного забезпечення, сприятиме створенню функціоналу та користувацького інтерфейсу.

Новизна створення даного веб-застосунку полягає у його адаптації до змін. Оскільки створення тестів відбувається за допомогою використання спеціальних форм – це забезпечує гнучкість і можливість швидкої модифікації завдань.

Сфера застосування цього додатку є різноманітною. Його можна використовувати в освітніх установах для підвищення академічних досягнень та рівня загального розвитку. Також цей застосунок може мати практичне застосування в різних професійних сферах. Наприклад, для фахівців, таких як рятувальники, медики або керівники підприємств, що працюють в умовах високого тиску.

Досліджуючи потенційний розвиток цього додатку, можна зробити кілька прогнозних припущень щодо його майбутнього зростання та ефективності. Одним із можливих напрямків подальшого розвитку є інтеграція алгоритмів штучного інтелекту. Це дозволить аналізувати результати користувача і надавати персоналізовані тренувальні вправи, адаптовані до конкретних потреб.

Ще один напрямок розвитку - розширення контенту та вправ застосунку. Додавання різноманітних когнітивних завдань, забезпечить користувачам комплексний досвід навчання. Постійно оновлюючи і розширюючи бібліотеку вправ, додаток може тримати користувачів залученими і мотивованими, запобігаючи монотонності і пропонуючи постійний розвиток навичок.

Крім того, інтеграція зворотного зв'язку в режимі реального часу та аналітика ефективності можуть значно покращити навчальний процес. Надаючи користувачам негайний зворотний зв'язок щодо їхніх відповідей, включаючи пояснення та поради щодо вдосконалення, додаток може сприяти відчуттю прогресу та заохочувати до постійного зростання.

Враховуючи широке використання мобільних пристроїв, розробка мобільної версії додатку є логічним кроком. Створивши який, користувачі зможуть отримати доступ до тренувальних вправ на своїх смартфонах або планшетах, що зробить його зручнішим і доступнішим, оскільки з’явиться можливість використовувати його будь-де та будь-коли.

Зрештою, майбутня розробка може прогнозувати партнерство з освітніми закладами або корпоративними навчальними програмами. Співпраця з цими організаціями може надати цінну інформацію про конкретні потреби та вимоги учнів у різних контекстах, сприяючи адаптації програми, та розширюючи її охоплення та вплив.

РОЗДІЛ 1 АНАЛІЗ СТАНУ ПРОБЛЕМНОЇ ОБЛАСТІ

У сучасній цифровій епосі у яку поринуло усе людство, веб-застосунки відіграли колосальну роль. Вони зробили революцію у можливості отримання інформації, спілкування з іншими та ведення бізнесу. Ці потужні інструменти стали невід'ємною частиною нашого повсякденного життя, пропонуючи зручність, ефективність і безмежні можливості. Веб-застосунки заполонили такі сфери діяльності:

Інформаційна та комунікаційна:

Веб-додатки – проміжна ланка до величезного обсягу інформації. Лише за кілька кліків люди можуть отримати доступ до величезних баз даних, освітніх ресурсів, новинних статей та дослідницьких матеріалів. Інтернет став платформою для глобального спілкування, об'єднуючи людей з різних куточків світу через соціальні мережі та електронну пошту. Веб-додатки подолали проблеми у відстані та часі, сприяючи налагодженню зв'язків і розширенню горизонтів.

Електронна комерція та онлайн-покупки:

Розвиток електронної комерції змінив вектор роздрібної торгівлі. Веб-додатки надають людям можливість переглядати і купувати товари та послуги, не виходячи з дому. Онлайн-ринки пропонують широкий вибір, конкурентні ціни та відгуки клієнтів, що дає змогу споживачам робити обґрунтовані покупки. Веб-додатки зробили шопінг доступним, ефективним і безпечним, а адресна доставка товарів стала чимось буденним.

Управління фінансами:

Веб-додатки спростили управління фінансами, дозволивши людям контролювати їх. Послуги онлайн-банкінгу забезпечують безпечний доступ до власних рахунків, що дозволяє здійснювати різного виду транзакції. Додатки для особистих фінансів допомагають людям у складанні бюджету, відстеженні витрат

та управлінні інвестиціями. Веб-додатки зробили управління фінансами більш прозорим, зручним і персоналізованим.

Продуктивність та співпраця:

Веб-додатки підвищили продуктивність і співпрацю як в особистому, так і в професійному середовищі. Системи управління проектами дозволяють командам оптимізувати завдання, відстежувати прогрес і ефективно спілкуватися. Платформи для обміну документами та хмарні сервіси зберігання даних забезпечують безперешкодну співпрацю та легкий доступ до файлів з будь-якого пристрою. Веб-застосунки зруйнували бар'єри для співпраці, даючи змогу людям працювати разом віддалено та ефективно.

Здоров'я:

Веб-додатки мають значний вплив на здоров'я людини, оскільки з’явилась можливість отримувати інформацію про стан його стан, планувати зустрічі та спілкуватися з медичними працівниками за допомогою телемедичних послуг. Веб-застосунки полегшують дистанційний моніторинг, профілактику та забезпечують доступ до онлайн ресурсів для тренувань, які пропонують персоналізовані програми вправ і пропагують здоровий спосіб життя.

Освіта та електронне навчання:

Веб-додатки зруйнували традиційні освітні системи, запропонувавши інноваційні підходи до навчання. Платформи електронного навчання надають людям доступ до онлайн-курсів, віртуальних класів та освітніх ресурсів. Студенти можуть брати участь в інтерактивних уроках, співпрацювати з однолітками та отримувати персоналізований зворотній зв'язок. Веб-додатки демократизували освіту, зробивши навчання доступним для людей по всьому світу, незалежно від їхнього географічного розташування чи соціально-економічного походження.

Розваги та споживання медіа:

Потокові платформи пропонують величезну бібліотеку фільмів, телепередач і музики, доступну в будь-який час і в будь-якому місці. Цифровий контент, такий

як електронні книги, журнали та блоги, надає безмежні можливості для дослідження та самовираження. Онлайн-ігри стали популярною формою розваг, об'єднуючи гравців по всьому світу і створюючи віртуальні спільноти. Веб-додатки перетворили дозвілля на захоплюючий та інтерактивний досвід.

Відзначаючи усю користь веб-застосунків, їх різноманіття та сфери використання, хочеться відмітити зростаючу кількість додатків, які направлені на тренування когнітивних здібностей та покращення навичок прийняття швидких рішень. За ці функції, як відомо, відповідає мозок. Він відіграє важливу роль у таких процесах як: прийняття рішень, запам’ятовування, аналіз та обробка інформації. Тому розвиток та тренування швидкого мислення неабияк впливатиме на навчання, роботу та загальний розвиток особистості.

1.1 Користь швидкого мислення

Швидке мислення – це безцінна навичка, яка має величезне значення в різних аспектах життя, та буде корисною для людей будь якої вікової категорії. Ось кілька ключових причин, чому швидке мислення корисне:

Управління часом

Швидке мислення дозволяє швидко обробляти інформацію та оперативно приймати рішення, що сприяє ефективному управлінню часом. Воно допомагає розставляти пріоритети, справлятися з кількома обов'язками одночасно та дотримуватися реченців.

Адаптивність

У світі, що стрімко змінюється, швидке мислення має вирішальне значення для адаптації до нових ситуацій, викликів і несподіваних подій. Воно дозволяє людям мислити “на ходу”, коригувати стратегії та знаходити інноваційні рішення для подолання перешкод.

Вирішення проблем

Швидко мислячі люди не просто думають швидше - вони швидко визначають найважливіші деталі, аналізують складні ситуації, виявляють закономірності та генерують креативні рішення. Швидке мислення покращує навички вирішення проблем, дозволяючи аналізувати складні ситуації, виявляти закономірності та генерувати креативні рішення. Це допомагає знаходити ефективні та дієві способи подолання викликів і досягнення бажаних результатів.

Прийняття рішень

Дослідження показали , що швидке мислення покращує здатність прийняття рішень, дозволяючи людям збирати необхідну інформацію, оцінювати альтернативи та вчасно робити обґрунтований вибір.

Комунікація та співпраця

Багато людей, які висловлюють свої думки чітко і лаконічно, вважаються розумними, тому що у них завжди є відповідь або ідея, якою вони можуть поділитися Це сприяє ефективній комунікації та роботі в команді.

Управління стресом

Швидке мислення допомагає людям краще справлятися зі стресом і напругою. Воно дозволяє їм зберігати спокій, раціонально мислити і приймати правильні рішення навіть у складних ситуаціях, зменшуючи негативний вплив стресу на продуктивність і самопочуття.

1.2 Методи тренування швидкого мислення

Існує широкий спектр вправ та завдань для розвитку швидкості мислення. Найефективнішими з яких є:

Швидкісне читання – це техніка, яка направлена на читання та сприйняття інформації в прискореному темпі. Покращуючи його, розвивається навичка

ефективно опрацьовувати дані, що дозволяє аналізувати та приймати рішення значно швидше.

Тренування багатозадачності забезпечує набуття таких навичок як виконання кількох завдань одночасно, визначення пріоритетів, управління часом та розподіл уваги. Один із поширених методів полягає у тому, щоб починати з вирішення простих завдань і поступово збільшувати їх складність і кількість. Це допоможе людям бути більш продуктивними та зосередженими на реалізації поставлених задач.

Фізичні вправи такі як заняття аеробікою, біг або їзда на велосипеді, збільшують приплив крові до мозку, покращуючи когнітивні показники та розумову спритність.

Медитація – метод, який допоможе покращити фокус, концентрацію та когнітивну гнучкість. Ці практики допомагають людям стати більш стресостійкими та адаптивними до мінливості обставин.

Ментальна арифметика – здатність виконувати обчислення подумки без використання зовнішніх засобів, таких як калькулятори, відіграє важливу роль у тренуванні швидкого мислення. Вона вливає на розвиток розумової спритності, когнітивної гнучкості та покращує фокус та концентрацію уваги на поставлених завданнях.

Тренування пам’яті – ефективний метод, який покращує об'єм пам'яті та здатність до пригадування. Він містить такі техніки та методи як:

- *Активна стратегія навчання*. Активна взаємодія з матеріалом, який ви хочете запам'ятати, може покращити вашу пам'ять. А такі методи, як узагальнення інформації своїми словами, викладання матеріалу комусь іншому або його обговорення , допомагають посилити формування та відновлення пам'яті.
- *Мнемотехніка.* Це система, яка володіє допоміжними засоби для запам'ятовування, які використовують асоціації, візуалізації або шаблони, щоб

допомогти запам'ятати інформацію. Сюди входить метод локусів (асоціювання предметів з конкретними місцями у знайомому місці), абревіатури (створення слова або фрази з перших літер списку) і чанкінг (групування інформації в менші, більш зручні для сприйняття шматки).

- *Регулярна практика і перевірка*. Послідовна практика має важливе значення для підтримки та зміцнення навичок пам'яті, а регулярне повторення і закріплення інформації допомагає консолідувати спогади, та зробити їх більш доступними.

- *Ігри та вправи*. Це інструменти, які допомагають ефективно тренувати пам’ять, використовуючи для цього: ігри на встановлення відповідності та концентрацію, кросворди, судоку та вправи на запам’ятовування об’єктів.

Проаналізувавши усі вище згадані пункти, можна сказати, що незважаючи на таке різноманіття методів тренування швидкого мислення, люди зазвичай обирають саме використання веб-застосунків з різними варіаціями вправ та ігор. Мабуть найпопулярнішими з яких є завдання у вигляді математичних обчислень, та задачі на логіку.

1.3 Веб-застосунки для розвитку швидкості мислення. Їх переваги та недоліки.

Lumosity

Переваги: Lumosity вміщає широкий вибір ігор, які розроблені таким чином, щоб бути приємними та складними, надаючи користувачам стимулюючий досвід під час тренування їхнього мозку. Також формує персоналізовані тренувальні програми та реалізовує відстеження прогресу.

Недоліки: Застосунок пропонує обмежену кількість ігор безкоштовно, повний доступ до функцій додатку та ширшого асортименту ігор вимагає підписки. Також деякі користувачі повідомляють про занепокоєння щодо перенесення навичок,

набутих в іграх, на завдання у реальному житті, що є досить небезпечним, тому що в людей розмивається межа між реальним та віртуальним світами.

Elevate

Переваги: Додаток фокусується на покращенні таких навичок, як пам'ять, розуміння прочитаного та математику. Він пропонує щоденні індивідуальні тренування, відстеження результатів та адаптивні рівні складності.

Недоліки: Безкоштовна версія Elevate має обмежений доступ до ігор та функцій.

CogniFit

Переваги: Застосунок пропонує комплексну програму тренувань, яка оцінює і тренує кілька когнітивних сфер, включаючи пам'ять, увагу і виконавчі функції.

Недоліки: Користувачі скаржаться на складність побудови інтерфейсу, порівнюючи його з іншими програмами. А також доступ до всіх можливостей сайту можна отримати лише оформивши підписку.

BrainHQ

Переваги: BrainHQ пропонує різноманітні вправи, спираючись на широкі наукові дослідження нейробіологів. Також додаток надає функцію зворотного зв’язку, що може бути дуже корисно людям, які вперше користуються подібними застосунками.

Недоліки: Існує проблема правильного підбору складності запитань, що негативно впливає на ефективність тренувань та перешкоджає користувачам активно продовжувати навчання.

Підсумовуючи характеристики усіх популярних веб-застосунків, відстежується певна закономірність, яка полягає в обов’язковій підписці, а це

зазвичай відштовхує користувачів. Одні вважають недоцільним витрачати кошти на онлайн ігри, а інші посилаються на скрутне фінансове становище. Тому створення безплатного аналога неабияк допоможе людям виконати їхнє бажання стати кращими.

1.4 Сучасні технології та їх можливості

Програмне забезпечення – це фундамент, на якому будується будь-який успішний веб-додаток. Воно створює передумови для ефективної розробки, безперешкодної співпраці, масштабованості та довготривалої підтримки. Обираючи програмне забезпечення, яке відповідає вимогам проєкту та використовує сильні сторони розробника, ви створюєте усі умови для створення високоякісного веб-застосунку. Це критично важливе рішення, яке може суттєво вплинути на весь процес розробки та кінцевий успіх з точки зору продуктивності, функціональності та задоволеності користувачів.

Сучасне ПЗ для реалізації веб-додатків складається з таких ключових компонентів як:

Інтегровані середовища розробки

IDE, такі як Visual Studio Code, IntelliJ IDEA та Sublime Text, надають комплексні середовища кодування з такими функціями, як підсвічування коду, інтеграція контролю версій та підтримка плагінів. Вони спрощують процес розробки та підвищують продуктивність.

Мови програмування

Популярні мови програмування для веб-розробки включають JavaScript, Python, Ruby, PHP та Java. Ці мови мають надійні екосистеми, великі бібліотеки та фреймворки, які підтримують розробку веб-додатків.

Фреймворки інтерфейсу

React, Angular та Vue.js, спрощують розробку інтерфейсів, надаючи можливості багаторазового використання компонентів, управління станами, маршрутизації та прив'язки даних. Вони покращують процес розробки інтерфейсу користувача (UI) і дозволяють створювати інтерактивні та адаптивні веб-застосунки.

Back-end фреймворки

Express.js (Node.js), Django (Python), Ruby on Rails та Laravel (PHP), пропонують готові модулі, бібліотеки та утиліти для бекенд-розробки. Вони надають інструменти для обробки запитів, управління базами даних, впровадження заходів безпеки та створення RESTful API.

Бази даних

Реляційні бази даних, такі як MySQL, PostgreSQL та Oracle, а також NoSQL бази даних, такі як MongoDB та Redis, зазвичай використовуються для веб-додатків. Вони ефективно зберігають і отримують дані та пропонують такі функції, як моделювання даних, запити та індексування.

Хмарні платформи

Amazon Web Services (AWS), Microsoft Azure та Google Cloud, надають масштабовану інфраструктуру для розміщення веб-додатків. Вони пропонують такі послуги, як віртуальні машини, контейнери, безсерверні обчислення та керовані бази даних, що спрощує розгортання та обслуговування.

Системи контролю версій (VCS)

Інструменти VCS, такі як Git, дозволяють розробникам відстежувати зміни, співпрацювати та ефективно керувати сховищами вихідного коду. Такі платформи, як GitHub та GitLab, надають послуги хостингу для віддалених сховищ коду та полегшують командну роботу.

Контейнеризація

Такі інструменти, як Docker та Kubernetes, дозволяють створювати та керувати контейнерними додатками. Вони забезпечують узгодженість та ізольованість середовищ, масштабованість і спрощене розгортання в різних інфраструктурах.

Інструменти тестування та забезпечення якості (QA)

Фреймворки, такі як Jest, Selenium та Cypress, полегшують автоматизоване тестування веб-додатків. Інструменти QA допомагають забезпечити функціональність, продуктивність і безпеку додатків, виявляючи і виправляючи помилки та вразливості.

Інструменти безперервної інтеграції та розгортання (CI/CD)

Інструменти CI/CD, такі як Jenkins, GitLab CI/CD та Travis CI, автоматизують процес збірки, тестування та розгортання. Вони дозволяють безперешкодно інтегрувати, безперервно тестувати та розгортати веб-додатки, забезпечуючи швидші цикли випуску та кращу якість коду.

Опираючись на усі вище згадані інструменти, для реалізації веб-застосунку тренування швидкого мислення було обрано:

- Середовище Visual Studio Code. Основними перевагами якого є простота використання, його можливості та функціональність, підтримка великої кількості мов програмування та високий рівень продуктивності.
- Мова програмування JavaScript. Вона володіє широкою підтримкою браузерів, забезпечує інтерактивність на стороні клієнта та має розгалужену екосистему.
- Фреймворки React.js та Express.js. Перший з яких забезпечує односпрямований потік даних, можливість мобільної розробки, оптимізацію продуктивності та полегшує тестування. А другий в свою чергу надає надійну маршрутизація, підтримку проміжного програмного забезпечення, а переваги

повного стеку Java Script роблять його популярним вибором для створення масштабованих та продуктивних веб-застосунків з використанням Node.js.

- База даних MongoDB. Вона надає потужні можливості для масштабування, узгодженості, відмовостійкості та гнучкості, що сприяють швидкій розробці та збереженню даних.

РОЗДІЛ 2 ОГЛЯД ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ

2.1 Середовище розробки – Visual Studio Code

Visual Studio Code (відомий як VS Code) - це безкоштовний текстовий редактор з відкритим вихідним кодом від Microsoft, логотип якого можна побачити на рисунку 2.1.1. Він доступний для таких операційних систем , як Windows, Linux та macOS[1].

Рис.2.1.1 – логотип Visual Studio Code[2]

Visual Studio Code - це не простий редактор коду; це потужне середовище, яке ставить написання коду в центр уваги. Основною метою Visual Studio Code є не створення двійкових файлів (таких як .exe та .dll), а полегшення написання коду для веб-, мобільних та хмарних платформ для будь-яких розробників, що працюють на різних операційних системах. VS Code надає середовище кодування, яке відрізняється від інших інструментів розробника, наприклад таких як Microsoft Visual Studio. По суті це середовище на основі папок, яке дозволяє легко працювати з файлами коду та пропонує уніфікований спосіб роботи з різними мовами.

Visual Studio Code має багато унікальних функцій. Вони перераховані нижче:

Підтримка кількох мов програмування. VS Code має вбудовану підтримку багатьох мов, наприклад таких як: C++, Java, JavaScript, Python та інші [3].

Можливості редагування. Забезпечує ряд потужних функцій редагування, які покращують процес кодування. Ось деякі з них, які можна використовувати у VS Code:

- *Підсвічування синтаксису*: VS Code використовує підсвічування синтаксису для виділення різних елементів коду, це допомагає ідентифікувати ключові слова, змінні, рядки, коментарі та інші компоненти коду[3].

- *IntelliSense:* Це інтелектуальна функція завершення коду. Вона надає контекстно-залежні підказки під час введення, допомагаючи розробникам писати код швидше і з меншою кількістю помилок[3].

- *Форматування коду:* Це вбудовані стандарти, які автоматично форматують код відповідно до попередньо визначених правил або визначених користувачем конфігурацій[3].

- *Рефакторинг коду:* VS Code надає вбудовані можливості рефакторингу для покращення структури коду та його підтримки[3].

- *Фрагменти:* Це попередньо визначені шаблони коду, які можна використовувати для написання коду. VS Code підтримує як вбудовані, так і визначені користувачем фрагменти, що дозволяє розробникам автоматизувати повторювані шаблони коду або складні структури коду[3].

Інтеграція з Git. VS Code легко інтегрується з Git-ом, популярною системою контролю версій. Він надає команди Git-а та візуальні індикатори для відстеження змін файлів, етапів модифікацій, фіксацій коду та управління гілками.

Робота з файлами та папками. Visual Studio Code базується на файлах і папках, а це означає, що ви можете відкрити один або декілька файлів коду окремо. Або відкрити папку, яка містить файли вихідного коду, і працювати з ними у структурований та організований спосіб.

2.2 Мова програмування JavaScript та її особливості

JavaScript – це динамічна та універсальна мова програмування, яка відіграє ключову роль у веб-розробці. Вона стала наріжним каменем сучасних веб-додатків, забезпечуючи інтерактивний та захоплюючий користувацький досвід. Вона дозволяє реалізовувати динамічні функції на веб-сторінках, які неможливо зробити лише за допомогою HTML і CSS.

Багато браузерів використовують JavaScript як мову сценаріїв для виконання динамічних речей в Інтернеті. Кожного разу, коли ви бачите спадне меню «Показати при натисканні», додатковий вміст, доданий на сторінку, і динамічно змінювані кольори елементів на сторінці, зокрема деякі функції, ви бачите наслідки JavaScript.

Згідно зі звітом Github Octoverse за 2022 рік, JavaScript є найбільш використовуваною мовою програмування(рис.2.2.1).

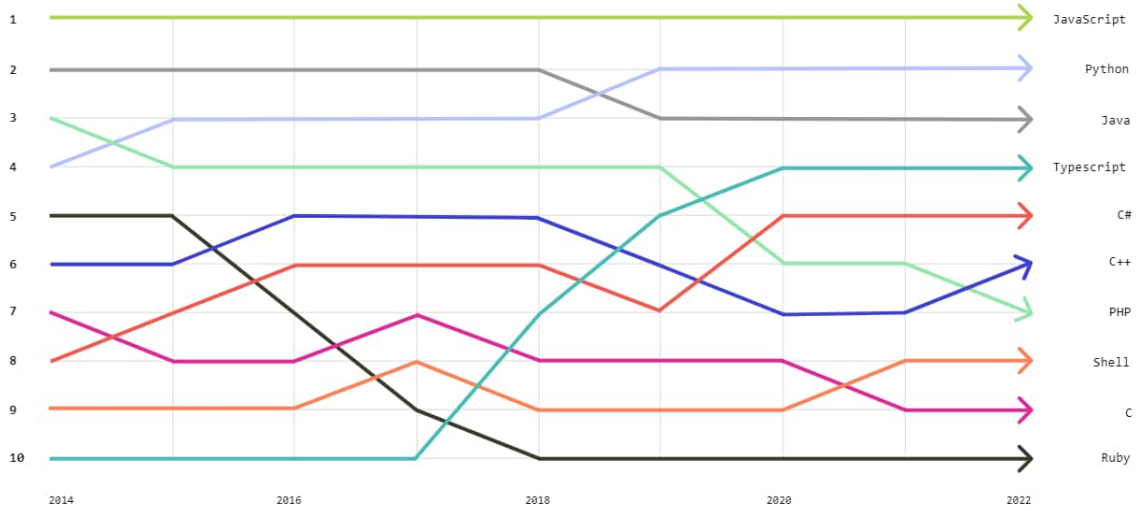

Рис. 2.2.1 – рейтинг мов програмування[4]

2.2.1 Переваги та недоліки JavaScript

JavaScript, як і будь-яка інша мова програмування, має свої переваги та недоліки. Говорячи про **переваги** – можна виділити деякі з них:

Швидкість

Оскільки JavaScript - це "інтерпретована" мова, тому вона скорочує час, необхідний для компіляції. Крім того, вона є сценарієм на стороні клієнта, який прискорює виконання програми, оскільки економить час, необхідний для підключення до сервера [5].

Завантаження сервера

JavaScript працює на стороні клієнта, тому серверу не доводиться мати справу з навантаженням, пов'язаним з виконанням JavaScript. Внаслідок цього, сервер працюватиме швидше і зможе сконцентруватися на інших завданнях, наприклад таких як управління даними[5].

Простота використання

JS - одна з найпростіших мов для вивчення, особливо для веб-програмування. Вона була розроблена таким чином, щоб бути простою для розуміння та використання веб-розробниками. Оскільки над складними мовами працює менше інженерів, то це вимагає більшого бюджету, тому використовуючи JavaScript веб-компанії заощаджують значні кошти на розробці [5].

Універсальність

JavaScript добре працює з іншими мовами і може використовуватися у величезній різноманітності програм. На відміну від сценаріїв PHP або SSI, JavaScript можна застосовувати для будь-якої веб-сторінки незалежно від розширення файлу, та в сценаріях, написаних іншими мовами, такими як Perl і PHP. Також цю мову програмування можна використовувати як для front-end, так і для back-end розробки. Back-end використовує NodeJS, тоді як багато інших бібліотек допомагають у розробці інтерфейсу, як-от AngularJS , ReactJS тощо [5].

Розширена функціональність

JavaScript – це мова програмування, яка розвивалася протягом багатьох років, надаючи розробникам різні способи розширення її функціональності. Завдяки бібліотекам, фреймворкам, API та індивідуальній розробці вона відкриває численні можливості для створення потужних і багатофункціональних веб-додатків. Використовуючи ці ресурси, розробники можуть підвищити свою продуктивність, та створити надійні та масштабовані додатки [5].

Популярність

Оскільки всі сучасні браузери підтримують JavaScript, тому її використання можна побачити майже скрізь. Усі відомі компанії використовують JavaScript як інструмент, включаючи Google, Amazon, PayPal тощо [5].

Недоліки JavaScript:

- *Безпека:* Оскільки код доступний для перегляду користувачеві, інші можуть використовувати його в зловмисних цілях, що ставить під загрозу безпеку даних, які передаються через сайт [5].

- *Підтримка браузерів:* JavaScript іноді інтерпретується різними браузерами по-різному. У той час як сценарії на стороні сервера завжди видають однакові результати, а сценарії на стороні клієнта можуть бути трохи непередбачуваними [5].

2.3 Програмна платформа Node.js

Node.js – це потужне і популярне серверне середовище, яке дозволяє розробникам запускати JavaScript-код поза веб-браузером. Воно підтримує написання сценаріїв на стороні сервера і надає ряд функцій, які допомагають створювати масштабовані веб-застосунки.

Node.js побудовано на JavaScript-движку V8, який був розроблений компанією Google для браузера Chrome. Ця платформа використовує модель вводу/виводу, керовану подіями, що робить її високоефективною і здатною

обробляти паралельні запити, не блокуючи виконання інших операцій. Ця дозволяє Node.js обробляти велику кількість з'єднань одночасно, що є ідеальним рішенням для додатків з високим трафіком і вимогами до комунікації в реальному часі.

2.3.1 Переваги Node.js

Однією з головних переваг Node.js є можливість використання JavaScript як на стороні клієнта, так і на стороні сервера, що забезпечує безперебійну комунікацію та спільне використання коду front-end-ом та back-end-ом додатку. Це позбавляє розробників необхідності використовувати декілька мов програмування і створює більш згуртований і впорядкований процес розробки.

Node.js має багату екосистему модулів та пакетів, доступних через менеджер npm. Ця велика колекція бібліотек з відкритим вихідним кодом дозволяє розробникам легко інтегрувати різні функціональні можливості у свої додатки, заощаджуючи час та зусилля. Node.js надає широкий спектр інструментів та ресурсів для підвищення продуктивності, наприклад таких як веб-фреймворки та конектори баз даних.

Масштабованість - ще одна ключова перевага. Архітектура Node.js дозволяє обробляти велику кількість паралельних запитів з мінімальним споживанням ресурсів.

Крім того, середовище Node.js пропагує модульний і блочний підхід до розробки. Завдяки підтримці CommonJS розробники можуть легко розбивати свій код на багаторазові та підтримувані модулі, покращуючи організацію коду та полегшуючи співпрацю між членами команди.

2.3.2 Недоліки Node.js

Незважаючи на численні переваги, Node.js також має деякі обмеження. Оскільки це однопотокове середовище, важкі завдання, що завантажують процесор, можуть блокувати цикл обробки подій і знижувати продуктивність. Однак це можна

усунути, розвантаживши ресурсозатратні завдання в окремі потоки або використавши кластерний модуль для розподілу навантаження між декількома екземплярами програми.

Іншим потенційним недоліком є стиль програмування на основі зворотних викликів, який може призвести до складних структур коду. Керування зворотними викликами та обробка помилок може стати важким завданням, особливо у великих і складних додатках, що призводить до створення коду, який важче підтримувати та налагоджувати.

2.4 Express.js

Express.js – безкоштовна платформа для Node.js [6], яка надає широкі можливості для створення веб- та мобільних додатків. Завдяки мінімалістичному підходу та потужній системі проміжного програмного забезпечення Express.js отримав широке розповсюдження серед розробників.

Express.js - це фреймворк Node.js, а це означає, що більша частина коду вже написана для роботи програмістів[6] . Використовуючи цей інструмент , ви можете створювати односторінкові, багатосторінкові або гібридні веб-додатки.

Також Express.js розширює функціональність Node.js. Він спростив програмування на Node.js і надав багато додаткових можливостей. Проте, ExpressJS і NodeJS багато в чому відрізняються один від одного. NodeJS - це середовище виконання JavaScript, яке надає платформу для виконання коду JavaScript на стороні сервера, а ExpressJS – це веб-фреймворк, побудований на основі NodeJS, який забезпечує ряд корисних функцій, таких як проміжне програмне забезпечення, маршрутизація та шаблони, які полегшують розробку та підтримку веб-додатків [7].

2.4.1 Особливості Express.js

Проміжне програмне забезпечення

Функції проміжного програмного забезпечення відіграють важливу роль в Express.js. Вони дозволяють розробникам додавати модульну функціональність до циклу запиту-відповіді свого додатку. Також ці функції можна використовувати для таких завдань, як автентифікація, ведення журналів, перевірка даних, обробка помилок тощо.

Маршрутизація

Express.js надає гнучку та інтуїтивно зрозумілу систему маршрутизації. Розробники можуть визначати маршрути для різних URL-адрес і HTTP-методів, що полегшує обробку різних типів запитів.

Шаблонування

Express.js надає вбудовану підтримку декількох популярних шаблонізаторів , таких як EJS та Jade, які полегшують створення динамічних та інтерактивних сторінок [7].

Інтеграція з базами даних

ExpressJS можна легко інтегрувати з популярними базами даних, такими як MongoDB та MySQL, що спрощує зберігання та отримання даних у вашому веб-додатку [7].

Обслуговування статичних файлів

Express.js дозволяє обслуговувати файли, такі як: HTML, CSS, зображення та файли JavaScript. Ця функція спрощує процес розміщення статичних ресурсів і робить зручним включення зовнішніх файлів у веб-застосунках.

Розробка RESTful API

Express.js добре підходить для створення RESTful API. Він надає інструменти для створення маршрутів, які відповідають принципам Representational State Transfer (REST- архітектурний стиль, визначений для створення та організації

розподілених систем [8]). Розробники можуть легко обробляти CRUD-операції та реалізовувати кінцеві точки API для клієнт-серверної взаємодії.

Налагодження

Налагодження має вирішальне значення для успішної розробки веб-додатків. ExpressJS спрощує цей процес, надаючи механізм, який надає можливість точно визначити частину веб-програми, яка має помилки.

2.5 Бібліотека React.js

React.js — це бібліотека JavaScript з відкритим кодом, яка використовується для швидкого та ефективного створення інтерактивних користувацьких інтерфейсів та онлайн-додатків з набагато меншою кількістю коду, ніж звичайний JS [9]. Вперше з'явилася в травні 2013 року і зараз є однією з найпоширеніших фронтенд-бібліотек для веб-розробки.

React дозволяє розробникам створювати великі веб-застосунки, які можуть змінювати дані, не перезавантажуючи сторінку, та генерувати багаторазові компоненти інтерфейсу, які можна порівняти з незалежними блоками Lego. Основна мета React — бути швидким, масштабованим і простим. Також його можна використовувати з комбінацією інших бібліотек або фреймворків JavaScript, таких як Angular JS у MVC.

Переваги React.js:

Легке створення динамічних додатків: React полегшує створення динамічних веб-додатків, оскільки вимагає менше коду і пропонує більше функціональності. Він використовує JSX (розширення JavaScript), яке є особливим синтаксисом, що дозволяє використовувати HTML-цитати та синтаксис HTML-тегів для відображення певних підкомпонентів.

Покращена продуктивність: React використовує Virtual DOM, завдяки чому веб-застосунки створюються швидше. DOM – це крос-платформний API для програмування, який працює з HTML, XML або XHTML. Virtual DOM порівнює попередні стани компонентів і оновлює тільки ті елементи, які були змінені, замість того, щоб оновлювати усі компоненти знову[10].

Багаторазові компоненти: Веб-застсоунок на ReactJS складається з декількох компонентів, кожен з яких має власну логіку та елементи керування. Ці компоненти відповідають за створення невеликого, багаторазового фрагменту HTML-коду, який можна використовувати повторно, де вам потрібно. Багаторазовий код допомагає полегшити розробку та підтримку ваших додатків. А компоненти можуть бути вкладені в інші, що дозволяє створювати складні додатки з простих блоків [10].

Односпрямований потік даних: React використовує односпрямований потік даних, а це означає, що при розробці додатків розробники часто вкладають дочірні компоненти в батьківські[10].

Швидке навчання: React легко вивчити, оскільки він здебільшого поєднує в собі базові концепції HTML та JavaScript з деякими корисними доповненнями[9].

Підтримка зручних інструментів: React JS також набув популярності завдяки наявності зручного набору інструментів, які роблять завдання розробників зрозумілішими та простішими. React Developer Tools були розроблені як розширення для Chrome та Firefox і дозволяють переглядати ієрархію React-компонентів у віртуальному DOM. Вони також дозволяють вибирати певні компоненти, переглядати та редагувати їхні поточні пропси та стан [10].

Розробки мобільних додатків: React Native – фреймворк, похідний від самого React, який є надзвичайно популярним і використовується для створення мобільних додатків [10].

Недоліки:

Високий темп розвитку має як переваги, так і недоліки. Недоліком є те, що оскільки середовище постійно змінюється, розробники відчувають деякі незручності, оскільки потрібно регулярно вивчати нові способи роботи.

Погана документація – це ще один мінус, який характерний для технологій, що постійно оновлюються. Тому з появою нових версій та інструментів , розробники пишуть інструкції самостійно.

ReactJS забезпечує лише рівні інтерфейсу користувача програми і нічого більше. Тому необхідно використовувати додаткові технології, щоб отримати повний набір інструментів для розробки.

2.5.1 React Native

React Native – фреймворк з відкритим вихідним кодом, розроблений компанією Facebook, який здійснив революцію у створенні мобільних додатків. З React Native розробники можуть створювати високопродуктивні мобільні додатки для платформ iOS та Android.

Однією з ключових переваг React Native є його здатність досягати крос-платформної сумісності. Це усуває потребу в окремих командах розробників і значно скорочує час та витрати на розробку. Повторне використання коду не лише пришвидшує процес розробки, але й забезпечує узгодженість на різних платформах.

React Native також має велику та активну спільноту розробників. Цей фреймворк має велику екосистему сторонніх бібліотек і пакетів, які можна легко інтегрувати в проекти. Ці бібліотеки надають рішення для загальних функціональних можливостей, таких як навігація, управління даними та компонентами інтерфейсу користувача, що дозволяє розробникам прискорити процес розробки та створювати багатофункціональні додатки.

Хоча React Native пропонує численні переваги, важливо враховувати його обмеження. Через свою крос-платформенну природу React Native може мати

обмежений доступ до певних специфічних API та функцій. У таких випадках розробникам може знадобитися написання власного коду або використання сторонніх бібліотек для доступу до цих функцій. Крім того, складні та графічно насичені додатки можуть стикатися з обмеженнями продуктивності.

2.5.2 JSX

JSX (JavaScript XML) - це розширення мови JavaScript, яке дозволяє розробникам писати HTML-подібний синтаксис безпосередньо в коді JavaScript. JSX змінив спосіб створення користувацьких інтерфейсів, зробивши із складних та інтерактивних веб-додатків простіші та інтуїтивно зрозуміліші.

Однією з ключових переваг JSX є його читабельність. Вбудовуючи HTML-подібний синтаксис у JavaScript, JSX забезпечує зрозумілий спосіб опису структури та зовнішнього вигляду користувацьких інтерфейсів. Розробники можуть легко візуалізувати макет та ієрархію компонентів, що полегшує розуміння та підтримку кодової бази. Крім того, JSX дозволяє використовувати атрибути HTML, що робить зручним встановлення властивостей і обробку подій безпосередньо в JSX-коді.

JSX також підвищує ефективність процесу розробки, оскільки дозволяє розробникам писати багаторазові компоненти, які можна компонувати та використовувати для створення складних інтерфейсів користувача. Крім того, JSX легко інтегрується з JavaScript, дозволяючи розробникам використовувати всю потужність цієї мови.

Отже, JSX став потужним інструментом для створення користувацьких інтерфейсів на JavaScript. Його декларативний та HTML-подібний синтаксис спрощує процес створення складних інтерфейсів, покращує читабельність коду та сприяє повторному використанню коду. У поєднанні з такими бібліотеками, як React, JSX стає ще більш потужним, дозволяючи створювати динамічні та інтерактивні веб-додатки.

2.6 База даних MongoDB

MongoDB - це програма для керування базами даних NoSQL з відкритим вихідним кодом. Вона відрізняється від традиційних реляційних баз даних тим, що використовує документно-орієнтовану модель, яка дозволяє створювати більш динамічні та схематичні структури даних.

MongoDB використовується для зберігання великих обсягів даних, дозволяючи підприємствам і організаціям керувати даними. А масштабованість і продуктивність роблять її придатною для широкого спектру застосувань, включаючи платформи електронної комерції, системи управління контентом та IoT додатки .

Використовувати MongoDB для можна наступних цілей:

- **Зберігання.** MongoDB може зберігати великі обсяги структурованих і неструктурованих даних і масштабується по вертикалі та горизонталі.
- **Інтеграція даних** для додатків, у тому числі для гібридних і мультихмарних.
- **Балансування навантаження.** MongoDB можна використовувати для роботи на декількох серверах.

MongoDB має свої плюси і мінуси, на які потрібно звернути увагу. Компанії повинні розуміти переваги, які пропонує MongoDB, і недоліки, якими вона володіє.

2.6.1 Переваги MongoDB

Гнучкість

MongoDB використовує гнучку модель даних на основі документів, що дозволяє легко зберігати неструктуровані та напівструктуровані дані та керувати ними. Ви можете зберігати дані в JSON-подібних документах, які можуть мати

різну структуру та поля. Така гнучкість забезпечує швидшу розробку та адаптацію до вимог. Крім цього, вона пропонує сумісність з популярними хмарними платформами, що полегшує розгортання та управління MongoDB в хмарних середовищах.

Масштабованість та продуктивність

MongoDB розроблена для роботи з великими об'ємами та високими швидкостями. Вона побудована таким чином, щоб бути ефективною та масштабованою. Працюючи на звичайному апаратному забезпеченні, підприємства можуть створювати потужні рішення для роботи з великими даними у власних центрах обробки даних і легко масштабувати їх у міру зростання та розвитку застосунків [11].

Висока доступність

MongoDB має вбудовані механізми реплікації та автоматичного обходу збоїв. Реплікація даних між декількома вузлами забезпечує високу доступність і відмовостійкість. Якщо один вузол виходить з ладу, то його репліка автоматично бере на себе цю роботу, забезпечуючи безперервний доступ до даних.

Розширена мова запитів

MongoDB підтримує широкий спектр операцій над запитами, включаючи фільтрацію, сортування, агрегування та геопросторові запити. Це дозволяє розробникам ефективно отримувати та аналізувати дані.

Активна спільнота та екосистема

База даних має активну спільноту розробників, що означає наявність великої кількості документації, навчальних посібників та ресурсів. Екосистема навколо MongoDB включає широкий спектр бібліотек, фреймворків та інструментів, які розширюють її функціональність і полегшують роботу з нею.

Продуктивність розробників

Гнучка схема та інтуїтивно зрозуміла мова запитів MongoDB сприяють підвищенню продуктивності розробників. JSON-подібна структура документів

також добре узгоджується з сучасними мовами програмування та фреймворками, спрощуючи процес розробки.

2.6.2 Недоліки MongoDB

Транзакції

У попередніх версіях MongoDB не підтримувалися транзакції між кількома документами або колекціями. Хоча в останніх версіях з'явилися багатодокументні транзакції, проте вони можуть бути не такими надійними, як у реляційних баз даних.

Пам'ять

MongoDB вимагає великого обсягу пам’яті через відсутність функцій об’єднання, що призводить до дублювання даних, які займають багато місця.

Обмежена кількість операцій приєднання

Документно-орієнтована природа MongoDB не підтримує складні операції об'єднання, як у традиційних реляційних базах даних. Хоча MongoDB надає деякі механізми для виконання агрегації та денормалізації даних, але вона може бути не такою ефективною та гнучкою.

Навчання

Мова запитів і модель даних MongoDB можуть вимагати від розробників вивчення нових концепцій і підходів. А адаптація до документно-орієнтованої природи MongoDB і розуміння того, як ефективно використовувати її можливості, може зайняти деякий час.

РОЗДІЛ 3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ

Архітектура даного веб-застосунку базується на стекові MERN. MERN (MongoDB, Express, React, NodeJS) – популярний стек для створення веб-додатків з великою кількістю інтерактивностей вбудованих у частину з front-end. Архітектура MERN дозволяє легко побудувати трирівневу архітектуру (front-end, back-end, DB) повністю за допомогою JavaScript і JSON [12].

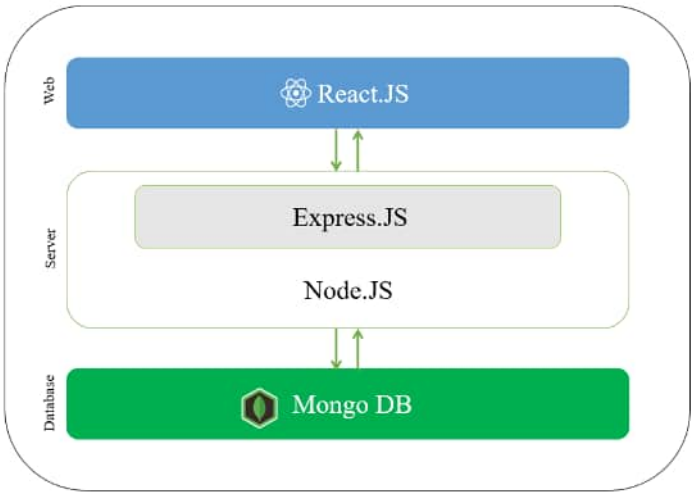

Рис.3.1 - Загальна схема взаємодії стеку MERN

3.1 Верхній рівень стеку MERN – React.JS

React – декларативна, ефективна і гнучка JavaScript-бібліотека, призначена для створення динамічних програм на стороні клієнта в HTML [13]. React дозволяє створювати складні інтерфейси за допомогою простих компонентів, підключати їх до даних на вашому сервері та відображати їх як HTML.

На основі React.js був побудований користувацький веб-інтерфейс. Перш за все необхідно правильно налаштувати React-застосунок. Для цього у терміналі проєкту потрібно створити новий React-застосунок за допомогою команди: *npx create-react-app* client, де client це назва частини проекту, яка відповідає за front-end.

Після створення потрібно перейти до папки client, використавши команду *cd client*, та запустити проєкт написавши команду *npm start*.

Успішно виконавши попередні кроки, переходимо до встановлення необхідних NPM-модулів. Спершу інсталюємо *react-router-dom* – основний пакет, що відповідає за реалізацію динамічної маршрутизації у веб-додатках [14], *antd* – бібліотека інтерфейсу користувача для створення веб-додатків, та *axios* – клієнт HTTP на основі Promise для node.js та браузера, використавши команду *npm install react-router-dom antd axios*. А також встановлюємо Redux Toolkit – це бібліотека, яка надає набір інструментів та утиліт для спрощення типових завдань Redux(бібліотека управління передбачуваним станом для JavaScript-додатків) та зменшення шаблонного коду, командою *npm i @reduxjs/toolkit redux react-redux* [15].

3.1.1 Реєстрація та авторизація користувачів

За екран **Реєстрації** відповідає файл index.js, шлях до якого виглядає наступним чином: src → pages →common →Register. У цьому файлів формується весь інтерфейс Реєстрації, який складається з полів для вводу імені, електронної пошти та паролю, кнопки Register, яка скеровує на екран Входу, а також з посилання на вхід в обліковий запис. Фрагмент коду програми, що реалізує вище сказані компоненти, наведений на рисунку 3.1.1.1.

```
<div className='flex justify-center items-center h-screen w-screen bg-primary'>
  <div className='card w-400 p-3 bg-white'>
    <div className="flex flex-col">
      <div className='flex'>
        <h1 className='text-2xl'>QUICKI - REGISTER
          <i class="ri-user-add-line"></i>
        </h1>
      </div>
      <div className='divider'></div>
      <Form layout='vertical' className='mt-2' onFinish={onFinish}>
        <Form.Item name='name' label='Name'>
          <input type='text' />
        </Form.Item>
        <Form.Item name='email' label='Email'>
          <input type='text' />
        </Form.Item>
        <Form.Item name='password' label='Password'>
          <input type='password' />
        </Form.Item>
        <div className='flex flex-col gap-2'>
          <button type='submit' className='primary-contained-btn mt-2 w-100'>Register</button>
          <Link to="/login" className='underline'>Already have an account?Login here.</Link>
        </div>
      </Form>
    </div>
  </div>

</div>
```

Рис.3.1.1.1 – формування інтерфейсу для екрану Реєстрації

Файл index.js , що відповідає за екран **Входу** знаходиться за наступним шляхом: src → pages →common →Login, та складається з тих самих компонентів, що і екран реєстрації, окрім поля для вводу імені–воно відсутнє. Також відмінність є у кнопці Log in , яка скеровує на головну сторінку сайту, та посиланні на реєстрацію у разі відсутності власного облікового запису. Фрагмент коду програми, що реалізує перелічені вище компоненти, наведений на рисунку 3.1.1.2.

```
<div className="flex justify-center items-center h-screen w-screen bg-primary">
  <div className="card w-400 p-3 bg-white">
    <div className="flex flex-col">
      <div className="flex">
        <h1 className="text-2xl">QUICKI - LOGIN <i class="ri-login-circle-line"></i></h1>

      </div>
      <div className="divider"></div>
      <Form layout="vertical" className="mt-2" onFinish={onFinish}>
        <Form.Item name="email" label="Email">
          <input type="text" />
        </Form.Item>
        <Form.Item name="password" label="Password">
          <input type="password" />
        </Form.Item>
        <div className='flex flex-col gap-2'>
          <button type='submit' className='primary-contained-btn mt-2 w-100' >Log in</button>
          <Link to="/register" className='underline'>Don't have an account?Register here.</Link>
        </div>
      </Form>
    </div>
  </div>

</div>
```

Рис.3.1.1.2 – формування інтерфейсу для екрану Входу

3.1.2 Створення сторінок сайту

Для легкого використання сайту, є такі необхідні інструменти як: навігаційне меню, що знаходиться по ліву сторону екрану та містить такі сторінки як: Home, Tests, Reports та Logout(вихід з облікового запису), та верхня панель на якій вказано статус користувача, та можливість зменшити навігаційне меню, натиснувши хрестик , який знаходиться у лівій частині.

Головна сторінка – **Home**, на якій зображено усі створені тести для тренування швидкого мислення. На кожній з форм тестування вказано: категорію, загальну кількість балів, яку можна отримати, мінімальний поріг для проходження та тривалість. Натиснувши кнопку “Start Test” ви зможете розпочати тестування. Також на цьому екрані можна спостерігати персональне звернення для кожного користувача, приклад якого зображено на рисунку 3.1.2.1

Hi Ira, Welcome to Quicki!

Рис.3.1.2.1 – персональне привітання

Наступна сторінка **Tests** доступна лише для облікового запису адміністратора. На ній зберігаються усі створені тести, а реалізація їх редагування та видалення зображена на рисунку 3.1.2.2.

```
title: "Action",
dataIndex: "action",
render: (text, record) => (
    <div className="flex gap-2">
        <i
            className="ri-pencil-line"
            onClick={() => {
                setSelectedQuestion(record);
                setShowAddEditQuestionModal(true);
            }}
        ></i>
        <i
            className="ri-delete-bin-line"
            onClick={() => {
                deleteQuestion(record._id);
            }}
        ></i>
    </div>
)
```

Рис.3.1.2.2 – редагування та видалення тестів

Перейшовши на панель редагування, буде наданий доступ до створення запитань. Для цього необхідно перейти на вкладку "Questions" та натиснути на кнопку "Add Question". Для формування складових запитання потрібно скористатися спеціальною формою, візуалізацію якої можна побачити на рисунку 3.1.2.3. Вона містить поля для введення запиту та позначення правильної відповіді, а також чотири комірки для можливих варіантів.

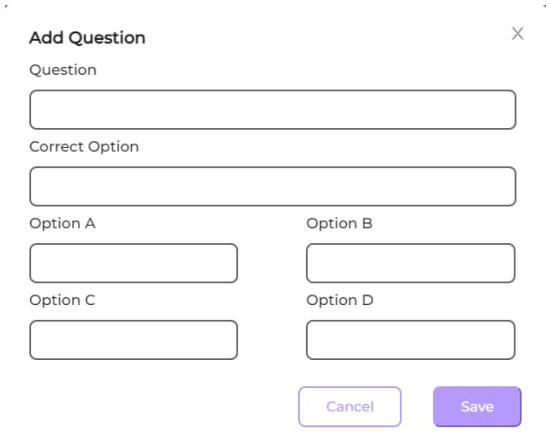

Рис.3.1.2.3 – форма створення запитань

Створені питання також можна редагувати та видаляти обравши одну з функцій розділу “Action”, наведених на рисунку 3.1.2.4.

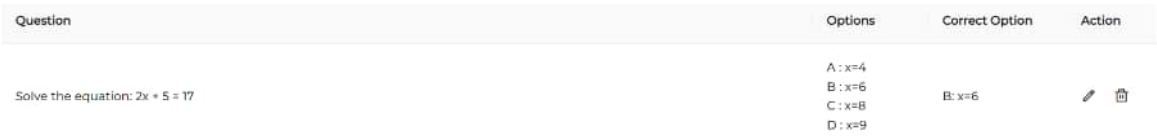

Рис.3.1.2.4 – редагування та видалення запитань

Сторінка **Reports** містить усю інформацію про завершені тести, проте є відмінність у даних між обліковими записами користувача та адміністратора. У першому доступні такі дані як назва, дата, та статус завершення тестування, а також загальна, мінімальна та поточна кількість балів. Фрагмент коду, що реалізує вище описані складові сторінки наведено на рисунку 3.1.2.5.

```
{
  title: "Tests Name",
  dataIndex: "examName",
  render: (text, record) => <>{record.exam.name}</>,
},
{
  title: "Date",
  dataIndex: "date",
  render: (text, record) => (
    <>{moment(record.createdAt).format("DD-MM-YYYY hh:mm:ss")}</>
  ),
},
{
  title: "Total Marks",
  dataIndex: "totalQuestions",
  render: (text, record) => <>{record.exam.totalMarks}</>,
},
{
  title: "Passing Marks",
  dataIndex: "correctAnswers",
  render: (text, record) => <>{record.exam.passingMarks}</>,
},
{
  title: "Obtained Marks",
  dataIndex: "correctAnswers",
  render: (text, record) => <>{record.result.correctAnswers.length}</>,
},
{
  title: "Verdict",
  dataIndex: "verdict",
  render: (text, record) => <>{record.result.verdict}</>,
},
```

Рис.3.1.2.5 – виведення інформації про пройдені тести

Для облікового запису адміністратора доступні ті ж самі дані, але додатково було створено розділ “User name” для відстеження тестів, які виконували користувачі. Також для зручності є дві форми сортування, а саме пошук за назвою тесту та за іменем користувача. Частина коду, яка відповідає за це зображено на рисунку 3.1.2.6 .

```
<div>
    <PageTitle title="Reports" />
    <div className="divider"></div>
    <div className='flex gap-2'>
        <input type='text' placeholder='Test Name'
            value={filters.examName}
            onChange={(e) => setFilters({ ...filters, examName: e.target.value })}
        />
        <input type='text' placeholder='User'
            value={filters.userName}
            onChange={(e) => setFilters({ ...filters, userName: e.target.value })}
        />
        <button className='primary-outlined-btn'
            onClick={() => {
                setFilters({
                    examName : "",
                    userName : "",
                })
                getData({
                    examName : "",
                    userName : "",
                });
            }}
        >Clear
        </button>
        <button className='primary-contained-btn'
            onClick={() => getData(filters)}
        >Search
        </button>
    </div>
    <Table columns={columns} dataSource={reportsData} className='mt-2' />
</div>
```

Рис.3.1.2.6 – сортування звітності

3.2 Серверна частина – Node.js та Express.js

Node.js – це середовище виконання JavaScript з відкритим вихідним кодом. Він дозволяє розробникам запускати JavaScript-код, поза браузером, а також створювати масштабовані та ефективні веб-додатки на стороні сервера. В свою чергу Express.js – це фреймворк, який спрощує завдання написання серверного коду. Він дозволяє визначати маршрути, та обробляти HTTP-запити, які відповідають певному шаблону.

Виконавши частину роботи з React.js, яка описана в розділі 3.1, усе створене навантаження необхідно відправити на серверну частину(back-end). Основною задачею цієї частини– це транспортування отриманої інформації до бази даних.

Перш за все необхідно провести ініціалізацію проекту за допомогою команди *npm init*, відкривши новий термінал проєкту. Ця команда запитує основну інформацію про проєкт, та створює файл package.json. Далі потрібно інсталювати

NPM-модулі такі як: *express*(фреймворк , що відповідає за маршрутизацію), *mongoose*(бібліотека, що створює зв'язок між MongoDB та середовищем виконання JavaScript Node.js), *jsonwebtoken*(токен для шифрування сесій), *bcryptjs*(функція що виконує безпечне зберігання паролів) та *dotenv*(компонент для зчитування файлів типу .env на Express), використавши команду npm і *express mongoose jsonwebtoken bcryptjs dotenv*. Також було завантажено утиліту nodemon, яка стежить за файловою системою проєкту та у разі необхідності автоматично перезапускає усі процеси.

Важливим етапом є створення Express.js сервера. Для цього у файлі server.js потрібно підключити модуль Express, створити екземпляр додатку, визначити порт на якому сервер буде слухати запити та імпортувати модуль підключення бази даних. Фрагмент коду, що реалізує це зображено на рисунку 3.2.1, а його успішну роботу можна побачити на рисунку 3.2.2.

```
const express = require("express");
const app = express();
require("dotenv").config();
app.use(express.json());
const dbConfig = require("./config/dbConfig");


const port = process.env.PORT || 5000;
app.listen(port, () => {
  console.log(`Server listening on port ${port}`);
});
```

Рис.3.2.1 – створення сервера Express.js

```
PS C:\Users\Ira\Desktop\Quicki_demo\server> n
odemon server
[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Server listening on port 5000
```

Рис.3.2.2 – запуск серверного файлу

3.2.1 Створення API для сторінок веб-застосунку

Спочатку розглянемо створення API для реєстрації та авторизації. Перш за все необхідно створити модель користувача. Для цього необхідно імпортувати потрібні модулі та визначити схему. У останній описуємо такі властивості як : ім'я,

електронна пошта, пароль та вказуємо, що усі користувачі за замовчуванням не матимуть доступу до облікового запису адміністратора. Реалізацію цієї частини зображено на рисунку 3.2.1.1.

```
const mongoose = require("mongoose");
const userSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: true,
    },
    email: {
      type: String,
      required: true,
      unique: true,
    },
    password: {
      type: String,
      required: true,
    },
    isAdmin: {
      type: Boolean,
      default: false,
    },
  },
  {
    timestamps: true,
  }
);

const userModel = mongoose.model("users", userSchema);

module.exports = userModel;
```

Рис. 3.2.1.1 – модель користувача

Структура API для **реєстрації** , створення якої зображено на рисунку 3.2.1.2, складається з перевірки на наявність облікового запису даного користувача, скрипта для шифрування паролів та створення нового акаунта (при успішній реєстрації якого можна побачити повідомлення продемонстроване на рисунку 3.2.1.3)

```
router.post("/register", async (req, res) => {
  try {
    // check if user already exists
    const userExists = await User.findOne({ email: req.body.email });
    if (userExists) {···
    }

    // hash password
    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(req.body.password, salt);
    req.body.password = hashedPassword;

    // create new user
    const newUser = new User(req.body);
    await newUser.save();
    res.send({
      message: "User created successfully",
      success: true,
    });
  } catch (error) {···
  }
});
```

Рис.3.2.1.2 - структура API для реєстрації

Рис.3.2.1.3 – повідомлення про успішну реєстрацію

Щодо структури API сторінки **авторизації** , то вона включає в себе такі пункти, як перевірка на наявність даного користувача в базі даних та правильність введеного паролю. Якщо пароль невірний, обробник маршруту надсилає відповідь із зазначенням помилки(рис. 3.2.1.4 - 3.2.1.5), якщо ж навпаки, то він генерує JWT-токен за допомогою бібліотеки jsonwebtoken, підписує його секретним ключем і надсилає токен у відповідь.

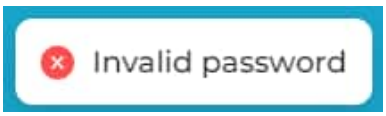

Рис.3.2.1.4 – повідомлення про неправильно вказаний пароль

Рис.3.2.1.5 – повідомлення про відсутність облікового запису

Фрагмент коду, що описує структуру вище згаданих пунктів зображено на рисунку 3.2.1.6.

```
router.post("/login", async (req, res) => {
  try {
    // check if user exists
    const user = await User.findOne({ email: req.body.email });
    if (!user) {
      return res
        .status(200)
        .send({ message: "User does not exist", success: false });
    }

    // check password
    const validPassword = await bcrypt.compare(···
    );
    if (!validPassword) {
      return res
        .status(200)
        .send({ message: "Invalid password", success: false });
    }

    const token = jwt.sign({ userId: user._id }, process.env.JWT_SECRET, {
      expiresIn: "1d",
    });

    res.send({
      message: "User logged in successfully",
      success: true,
      data: token,
    });
  } catch (error) {···
  }
});
```

Рис. 3.2.1.6 - структури API авторизації

Для сторінки **Tests**, яка відповідає за створення тестів, необхідна така ж модель, як і для реєстрації та авторизації, але з іншими властивостями. Сюди входять: назва тесту, тривалість, категорія, загальна та мінімальна оцінки та запитання. Реалізацію цієї моделі зображено на рисунку 3.2.1.7.

```
const mongoose = require('mongoose');

const examSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
    },
    duration: {
        type: Number,
        required: true,
    },
    category: {
        type: String,
        required: true,
    },
    totalMarks: {
        type: Number,
        required: true,
    },
    passingMarks: {
        type: Number,
        required: true,
    },
    questions: {
        type: [mongoose.Schema.Types.ObjectId],
        ref: "questions",
        required: true,
    },
}, {
    timestamps: true,
});

const Exam = mongoose.model("exams", examSchema);
module.exports = Exam;
```

3.2.1.7 – модель сторінки Tests

Описуючи структуру прикладного програмного інтерфейсу цієї сторінки можна відмітити такі функції як додавання тестів та запитань, а також їх редагування та видалення. Загальний вигляд яких представлено на рисунку 3.2.1.8.

```
// add exam
router.post("/add", authMiddleware, async (req, res) => {···
});

// get all exams
router.post("/get-all-exams", authMiddleware, async (req, res) => {···
});

// get exam by id
router.post("/get-exam-by-id", authMiddleware, async (req, res) => {···
})

// edit exam by id
router.post("/edit-exam-by-id", authMiddleware, async (req, res) => {···
});

// delete exam by id
router.post("/delete-exam-by-id", authMiddleware, async (req, res) => {···
});

// add question to exam
router.post("/add-question-to-exam", authMiddleware, async (req, res) => {···
});

// edit questions in exam
router.post("/eddit-question-in-exam", authMiddleware, async (req, res) => {···
});

// delete question in exam
router.post("/delete-question-in-exam", authMiddleware, async (req, res) => {···
});
```

Рис.3.2.1.8 – основні функції сторінки Tests

Сторінка **Reports** була реалізована для збереження усіх пройдених тестів. API якої включає в себе:

- Створення схеми та моделі Mongoose , яка знаходиться у файлі під назвою *reportModel.js* та складається з визначення Mongoose за допомогою mongoose.Schema(), яка визначає структуру та правила валідації для об'єкта «Reports» та включає в себе такі поля: user(збереження ідентифікатора користувача), exam(збереження ідентифікатора тесту), result(представляє результат звіту), timestamps(відповідає за відстеження міток часу їх створення і модифікації). Реалізацію цього зображено на рисунку 3.2.1.9.

```
const mongoose = require('mongoose');

const reportSchema = new mongoose.Schema({
    user: {
        type: mongoose.Schema.Types.ObjectId,
        ref: "users"
    },
    exam: {
        type: mongoose.Schema.Types.ObjectId,
        ref: "exams",
    },
    result: {
        type: Object,
        required: true,
    },
},{
    timestamps: true,
});

const Report = mongoose.model("reports", reportSchema);

module.exports = Report;
```

Рис.3.2.1.9 – схема та модель Mongoose

Код, що зображений вище, дозволяє створювати, змінювати і запитувати документи в колекції «reports» в MongoDB. Модель надає інтерфейс для взаємодії з базою даних і виконання таких операцій, як збереження нових звітів, оновлення існуючих та пошук на основі певних критеріїв.

- Створення файлу reportsRoute.js для взаємодії з базою даних. Для цього потрібно: підключити необхідні модулі, використати метод Post та функцію проміжного програмного забезпечення для автентифікації (authMiddleware).

Основними компонентами цього файлу є такі функції як додавання звітів та отримання звітності. Додавання звітів включає в себе використання моделей для обробки іспитів і звітів, та модель роботи з користувачами. А щодо отримання звітності то слід виділити такі два види як: отримання всієї звітності(реалізовується шляхом створення нового звіту виконуючи збереження інформації до бази даних) та звітність на основі заданих фільтрів(в даному випадку це examName та userName)

3.3 База даних MongoDB

Перш за все необхідно зареєструватись на MongoDB Atlas. Наступний крок - створення бази даних у кластері(рис.3.3.1).

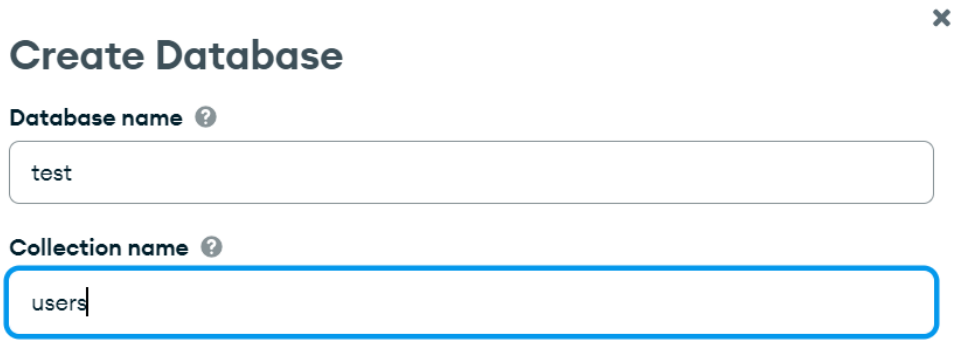

Рис.3.3.1 – створення бази даних

Використовуючи MongoDB Compas можна відстежувати усі зміни у базі даних. Для цього потрібно підключити кластер з створеною базою даних «test» скориставшись приєднанням , яке надається в MongoDB Atlas (рис.3.3.2)

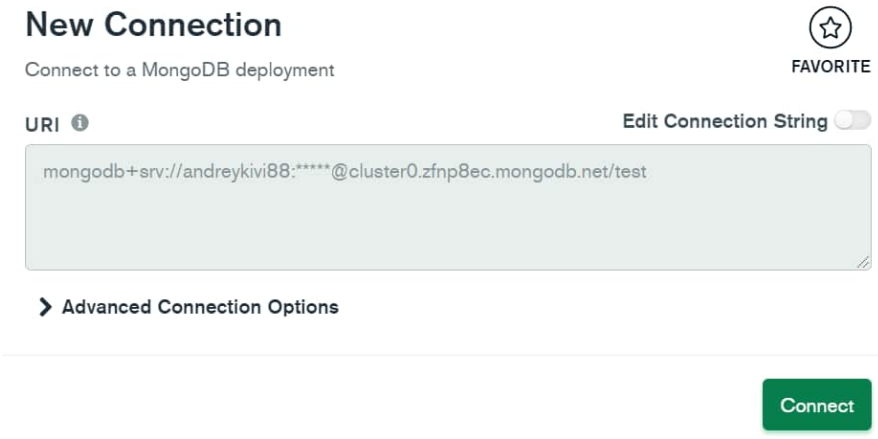

Рис.3.3.2 – приєднання бази даних до MongoDB Compas

Після успішного підключення, необхідно з’єднати базу даних з серверною частиною проєкту. За реалізацію приєднання відповідає файл db.config(рис.3.3.3).

```
const mongoose = require('mongoose');

mongoose.connect(process.env.MONGO_URL)

const connection = mongoose.connection;

connection.on('connected', () => {
    console.log('Mongo DB Connection Successful');
})

connection.on('error', (err) => {
    console.log('Mongo DB Connection Faild');
})

module.exports = connection;
```

Рис.3.3.3 – приєднання бази даних

Цей файл включає в себе:

- імпортування модуля mongoose;
- використання методу mongoose.connect, який викликається для встановлення з'єднання з базою даних та у якості аргументу використовує URL з'єднання , яке зберігається у файлі *.env*. Це дозволяє використовувати змінні оточення для зберігання конфіденційної інформації.
- використання подій 'connected' та 'error'. Вони обробляються функцією зворотного виклику connection.on та у разі успішного приєднання('connected') виводиться у консоль повідомлення про успішне з'єднання (рис.3.3.4), а у протилежному випадку('error') з’являється попередження про помилку.

```
Mongo DB Connection Successful
```

Рис.3.3.4 – повідомлення про успішне підключення бази даних

РОЗДІЛ 4 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ВЕБ-ЗАСТОСУНКУ QUICKI

4.1 Запуск веб-застосунку

Спершу необхідно перевірити роботу порта та підключитись до уже створеної бази даних. Для цього потрібно відкрити термінал проєкту, та переключитись на його серверну частину за допомогою команди *cd server*. Далі вводимо nodemon server щоб переконатись, що все правильно працює. Результат успішної роботи зображено на рисунку 4.1.1.

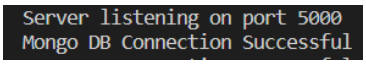

Рис.4.1.1 – робота порта та бази даних

Наступний крок це запуск самого веб-застосунку. Для цього необхідно відкрити це один термінал, та використавши команду cd client переключитися на клієнтську частину проєкту. Тепер написавши команду npm start виконуємо розгортання веб-сайту(рис.4.1.2).

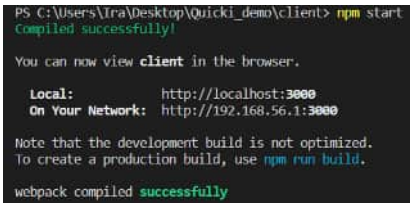

Рис.4.1.2 – успішний запуск застосунку

4.2 Взаємодія з веб-додатком

Після запуску, відкриється вкладка у браузері з формою для входу(рис.4.2.1).

Рис.4.2.1 – форма входу в застосунок

У разі відсутності облікового запису необхідно натиснути за посиланням «Don’t have an account?Register here.» для переходу на форму реєстрації(рис.4.2.1)

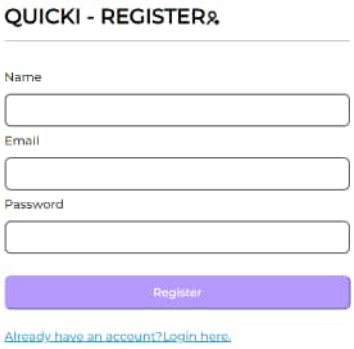

Рис.4.2.1 – форма для реєстрації у додаток

Увійшовши в обліковий запис адміністратора, відкривається екран з усіма тестами, який зображений на рисунку 4.2.2.

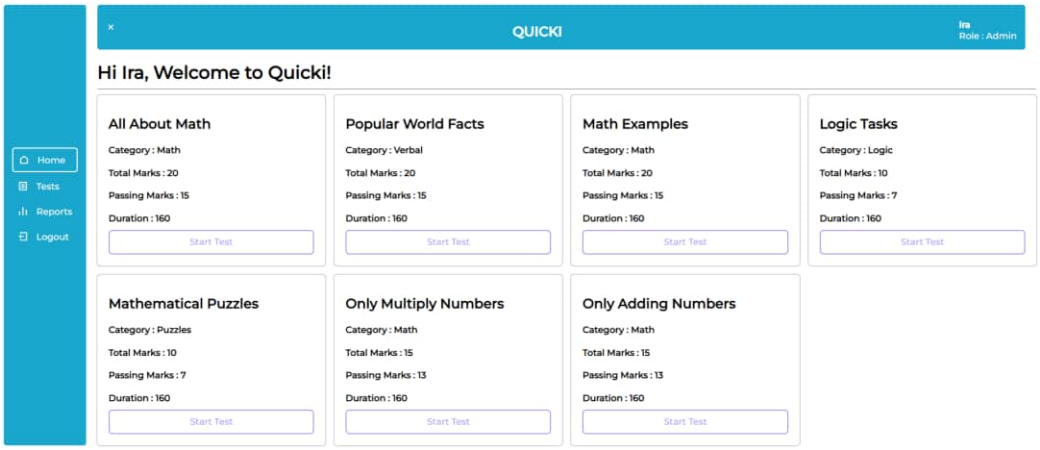

Рис.4.2.2 – перелік тестів

Натиснувши кнопку Start Test, спочатку з’явиться інструкція з правилами тестування(рис.4.2.3), після ознайомлення з якою можна розпочинати тестування. Фрагмент проходження тестування можна побачити на рисунку 4.2.4.

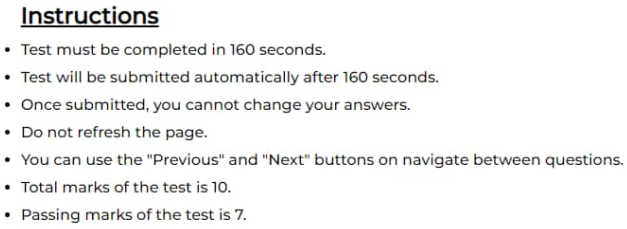

Рис.4.2.3 – індивідуальна інструкція до тестування

Рис.4.2.4 – приклад проходження тестування

Після закінчення тестування результат з’явиться на екрані, візуалізацію якого зображено на рисунку 4.2.5. Також є можливість повторного проходження та перегляду обраних відповідей(рис.4.2.6).

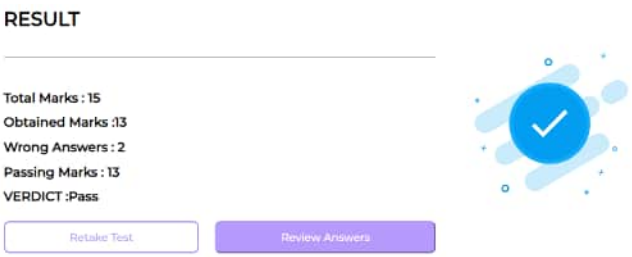

Рис.4.2.5 – зображення результату

Рис.4.2.6 – перегляд запитань

Звітність про проходження тестувань зберігаються у розділі Reports(рис.4.2.7).

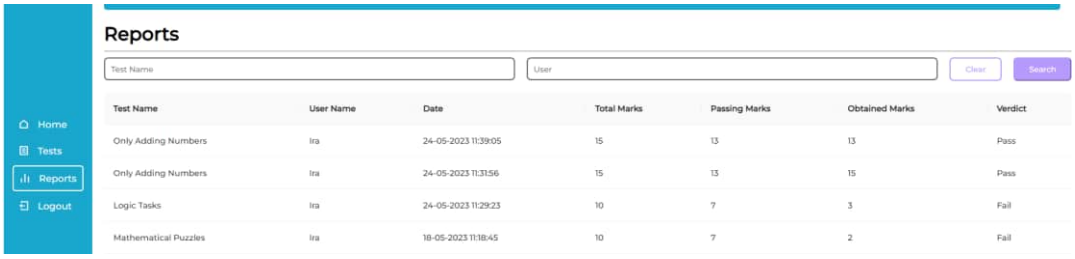

Рис.4.2.7 – інформація про тести

У розділі Tests, який доступний лише адміністративному обліковому запису, зберігаються створені тести. Натиснувши на кнопку Add Test, відкривається меню для створення тестів(рис.4.2.8), після створення якого з'являється можливість добавляти до них запитання.

Рис.4.2.8 – меню створення тестів

4.3 Відображення інформації у базі даних

Загальний вигляд бази даних проєкту включає в себе такі розділи як: exams, questions, reports, user, візуалізацію яких можна побачити на рисунку 4.3.1.

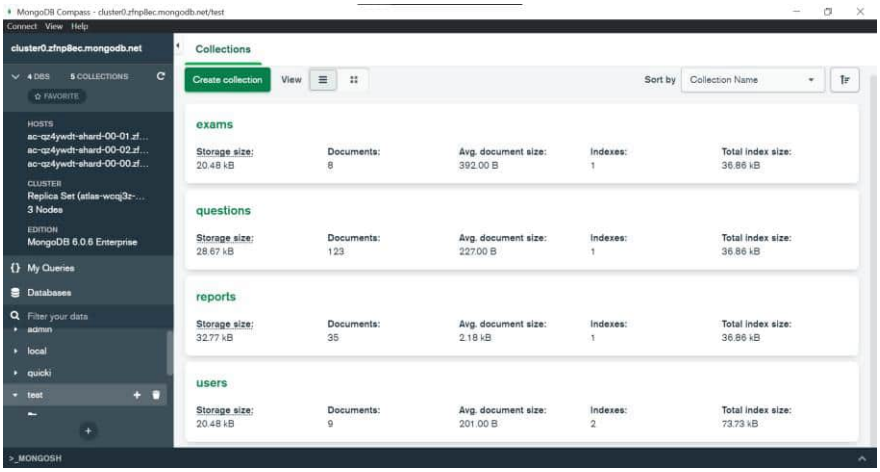

Рис.4.3.1 - Конфігурація MongoDB у програмі

На прикладі колекції users , показано зразок створення документів, який зображено на рисунку 4.3.2.

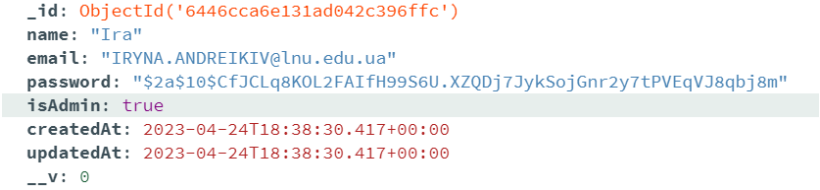

Рис.4.3.2 – документ адміністративного облікового запису

ВИСНОВКИ

Виконання цієї кваліфікаційної роботи було присвячене розробці веб-застосунку для тренування швидкого мислення за допомогою інтерактивного опитування. Основна мета полягала у досліджені когнітивних здібностей людей, на основі яких було створено корисну платформу, яка сприятиме розвитку навичок швидкого мислення та забезпечуватиме актуальними методиками для тренувань.

Реалізація передбачала використання стеку MERN, який забезпечив потужний набір технологій для побудови надійного та ефективного веб-застосунку. Використовуючи такі засоби як: MongoDB, Express.js, React.js та Node.js, вдалось сформували цілісну та масштабовану основу для розробки веб-додатку, забезпечивши безперебійну інтеграцію та оптимальну продуктивність.

У процесі розробки було реалізовано кілька ключових функцій для забезпечення ефективного та зручного користування додатком. По-перше, застосунок пропонує як адміністративний, так і користувацький облікові записи. Таке розмежування дозволяє адміністраторам керувати системою, створювати та редагувати тести, а користувачам – отримувати доступ до них, та переглядати результати їх проходження, за допомогою сторінки із звітністю. Цей механізм зворотного зв'язку дає змогу відстежувати прогрес і встановлювати особисті цілі для подальшого вдосконалення.

Загалом, ця бакалаврська робота продемонструвала доцільність і потенціал для створення веб-застосунків, які спрямовані на розвиток навичок швидкого мислення. А постійний розвиток технологій та методик відкриває багатообіцяючі можливості для їх подальшого вдосконалення та розширення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is Visual Studio Code?. Educative: Interactive Courses for Software Developers. URL:https://www.educative.io/answers/what-is-visual-studio-code (дата звернення: 18.05.2023).
2. File:Visual Studio Code 1.35 icon.svg - Wikimedia Commons. *Wikimedia Commons*.URL:https://commons.wikimedia.org/wiki/File:Visual_Studio_Code_1.35_icon.svg (дата звернення: 04.05.2023).
3. Syncfusion. .NET, Xamarin, JavaScript, Angular UI components | Syncfusion. URL: https://www.syncfusion.com/succinctly-free-ebooks/visual-studio-code-succinctly/code-editing-evolved-for-windows-linux-and-os-x (дата звернення: 05.05.2023).
4. The top programming languages. The State of the Octoverse. URL: https://octoverse.github.com/2022/top-programming-languages (дата звернення: 06.05.2023).
5. Advantages of JavaScript - Code Institute DE. Code Institute DE. URL: https://codeinstitute.net/global/blog/advantages-of-javascript/(дата звернення: 25.05.2023).
6. What is Express.js? | Why should use Express.js? | Features of Express.js. Besant Technologies. URL: https://www.besanttechnologies.com/what-is-expressjs (дата звернення: 05.05.2023).
7. What is Express JS? Its features & Uses | Intellipaat. Intellipaat Blog. URL: https://intellipaat.com/blog/what-is-express-js/?US (дата звернення: 06.05.2023).
8. Doglio F. REST API Development with Node.js: Manage and Understand the Full Capabilities of Successful REST Development. 2nd ed. New York : Apress, 2018. 402 p.

9. All About ReactJS | Flexiple. Hire Freelance Developers & Designers - Your Dream Team | Flexiple. URL: https://flexiple.com/react/deep-dive/ (дата звернення: 22.05.2023).
10. Deshpande C. The Best Guide to Know What Is React [Updated]. Simplilearn.com. URL: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs (дата звернення: 25.05.2023).
11. Cottrell N. MongoDB Topology Design: Scalability, Security, and Compliance on a Global Scale. New York : Apress, 2020. 351 p.
12. What Is The MERN Stack? Introduction & Examples. MongoDB. URL: https://www.mongodb.com/mern-stack (дата звернення: 04.05.2023).
13. Subramanian V. Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node. New York : Apress, 2019. 565 p.
14. React Router vs. React Router DOM | Syncfusion Blogs. Syncfusion. URL: https://www.syncfusion.com/blogs/post/react-router-vs-react-router-dom.aspx (дата звернення: 18.05.2023).
15. @reduxjs/toolkit. npm. URL: https://www.npmjs.com/package/@reduxjs/toolkit (дата звернення: 05.05.2023).

ДОДАТОК А. FrontCode

1. App.js

```
import { Button, Space } from 'antd';
import './stylesheets/theme.css';
import './stylesheets/alignments.css';
import './stylesheets/textelements.css';
import './stylesheets/custom-components.css';
import './stylesheets/form-elements.css';
import './stylesheets/layout.css';
import { Routes, Route, BrowserRouter } from 'react-router-dom';
import Login from './pages/common/Login';
import Register from './pages/common/Register';
import ProtectedRoute from './components/ProtectedRoute';
import Home from "./pages/common/Home";
import Exams from './pages/admin/Exams';
import AddEditExam from './pages/admin/Exams/AddEditExam';
import Loader from './components/Loader';
import { useSelector } from 'react-redux';
import WriteExam from './pages/user/WriteExam';
import UserReports from './pages/user/UserReports';
import AdminReports from './pages/admin/AdminReports';

function App() {
  const { loading } = useSelector(state => state.loader)
  return (
    <>
      {loading && <Loader />}
      <BrowserRouter>
        <Routes>
          {/* Common Routes */}
          <Route path="/login" element={<Login />} />
          <Route path="/register" element={<Register />} />

          {/* User Routes */}
          <Route
            path="/"
            element={
              <ProtectedRoute>
                <Home />
              </ProtectedRoute>
            }
          />
          <Route
            path="/user/write-exam/:id"
```

```
  element={
    <ProtectedRoute>
      <WriteExam />
    </ProtectedRoute>
  }
/>
<Route
  path="/user/reports"
  element={
    <ProtectedRoute>
      <UserReports />
    </ProtectedRoute>
  }
/>
{/* Admin Routes */}
<Route
  path="/admin/tests"
  element={
    <ProtectedRoute>
      <Exams />
    </ProtectedRoute>
  }
/>
<Route
  path="/admin/exams/add"
  element={
    <ProtectedRoute>
      <AddEditExam />
    </ProtectedRoute>
  }
/>

<Route
  path="/admin/exams/edit/:id"
  element={
    <ProtectedRoute>
      <AddEditExam />
    </ProtectedRoute>
  }
/>
<Route
  path="/admin/reports"
  element={
    <ProtectedRoute>
      <AdminReports />
    </ProtectedRoute>
  }
```

```
        />
      </Routes>
    </BrowserRouter>
  </>
 );
}

export default App;
```

2. exam.js

```
import axios from "axios";

const { default: axiosInstance } = require(".");

// add test

export const addExam = async (payload) => {
  try {
    const response = await axiosInstance.post("/api/exams/add", payload);
    return response.data;
  } catch (error) {
    return error.response.data;
  }
};

// get all tests
export const getAllExams = async () => {
  try {
    const response = await axiosInstance.post("/api/exams/get-all-exams");
    return response.data;
  } catch (error) {
    return error.response.data;
  }
};

// get test by id

export const getExamById = async (payload) => {
  try {
    const response = await axiosInstance.post("/api/exams/get-exam-by-id", payload);
    return response.data;
  } catch (error) {
    return error.response.data;
  }
};
```

```
//edit test by id

export const editExamById = async (payload) => {
    try {
        const response = await axiosInstance.post("/api/exams/edit-exam-by-id", payload)
        return response.data;
    } catch (error) {
        return error.response.data;
    }
}

//delete test by id

export const deleteExamById = async (payload) => {
    try {
        const response = await axiosInstance.post("/api/exams/delete-exam-by-id", payload);
        return response.data;
    } catch (error) {
        return error.response.data;
    }
}

// add question to test
export const addQuestionToExam = async (payload) => {
    try {
        const response = await axiosInstance.post("/api/exams/add-question-to-exam", payload);
        return response.data;
    } catch (error) {
        return error.response.data;
    }
}

export const editQuestionById = async (payload) => {
    try {
        const response = await axiosInstance.post("/api/exams/eddit-question-in-exam", payload);
        return response.data;
    } catch (error) {
        return error.response.data;
    }
}

export const deleteQuestionById = async(payload) => {
    try {
        const response = await axiosInstance.post("/api/exams/delete-question-in-exam", payload);
        return response.data;
    } catch (error) {
```

```
    return error.response.data; }};
```

3. reports.js

```
const { default: axiosInstance } = require(".");

//add report
export const addReport = async(payload) => {
   try {
      const response = await axiosInstance.post("/api/reports/add-report", payload);
      return response.data;
   } catch (error) {
      return error.response.data;
   }
}

// get all reports
export const getAllReports = async(filters) => {
   try {
      const response = await axiosInstance.post("/api/reports/get-all-reports", filters);
      return response.data;
   } catch (error) {
      return error.response.data;
   }
}

//get all reports by user
export const getAllReportsByUser = async() => {
   try {
      const response = await axiosInstance.post("/api/reports/get-all-reports-by-user");
      return response.data;
   } catch (error) {
      return error.response.data;
   }
}
```

4. user.js

```
const { default: axiosInstance } = require(".");

export const registerUser = async (payload) => {
   try {
      const response = await axiosInstance.post('/api/users/register', payload);
      return response.data;
   } catch (error) {
      return error.response.data;
```

```
    }
}

export const loginUser = async (payload) => {
    try {
        const response = await axiosInstance.post('/api/users/login', payload);
        return response.data;
    } catch (error) {
        return error.response.data;
    }
}

export const getUserInfo = async () => {
    try {
        const response = await axiosInstance.post('/api/users/get-user-info');
        return response.data;
    } catch (error) {
        return error.response.data;
    }
}
```

5. Exams → index.js

```
import React from 'react'
import { message, Table } from "antd";
import PageTitle from '../../../components/PageTitle';
import { useNavigate } from 'react-router-dom';
import { useEffect } from "react";
import { useDispatch } from "react-redux";
import { deleteExamById, getAllExams } from "../../../apicalls/exams";
import { HideLoading, ShowLoading } from "../../../redux/loaderSlice";

function Exams() {
  const navigate = useNavigate();
  const [exams, setExams] = React.useState([]);
  const dispatch = useDispatch();

  const getExamsData = async () => {
    try {
      dispatch(ShowLoading());
      const response = await getAllExams();
      dispatch(HideLoading());
      if (response.success) {
        setExams(response.data);
      } else {
        message.error(response.message);
```

```
    }
  } catch (error) {
    dispatch(HideLoading());
    message.error(error.message);
  }
};

const deleteExam = async (examId) => {
  try {
    dispatch(ShowLoading());
    const response = await deleteExamById({
      examId,
    });
    dispatch(HideLoading());
    if (response.success) {
      message.success(response.message);
      getExamsData();
    } else {
      message.error(response.message);
    }
  } catch (error) {
    dispatch(HideLoading());
    message.error(error.message);
  }
};
const columns = [
  {
    title: "Training Test",
    dataIndex: "name",
  },
  {
    title: "Duration",
    dataIndex: "duration",
  },
  {
    title: "Category",
    dataIndex: "category",
  },
  {
    title: "Total Marks",
    dataIndex: "totalMarks",
  },
  {
    title: "Passing Marks",
    dataIndex: "passingMarks",
  },
  {
```

```
    title: "Action",
    dataIndex: "action",
    render: (text, record) => (
      <div className="flex gap-2">
        <i
          className="ri-pencil-line"
          onClick={() => navigate(`/admin/exams/edit/${record._id}`)}
        ></i>
        <i
          className="ri-delete-bin-line"
          onClick={() => deleteExam(record._id)}
        ></i>
      </div>
    ),
  },

];

useEffect(() => {
  getExamsData();
}, []);
return (
  <div>
    <div className="flex justify-between mt-2 items-end">
      <PageTitle title="Training Tests" />

      <button
        className="primary-outlined-btn flex items-center"
        onClick={() => navigate("/admin/exams/add")}
      >
        <i className="ri-add-line"></i>
        Add Test
      </button>
    </div>
    <div className="divider"></div>

    <Table columns={columns} dataSource={exams} />
  </div>
);
}

export default Exams;
```

6. Home → index.js

```
import { Col, message, Row } from "antd";
import React, { useEffect } from "react";
```

```
import { useDispatch, useSelector } from "react-redux";
import { getAllExams } from "../../../apicalls/exams";
import { HideLoading, ShowLoading } from "../../../redux/loaderSlice";
import PageTitle from "../../../components/PageTitle";
import { useNavigate } from "react-router-dom";
function Home() {
  const [exams, setExams] = React.useState([]);
  const navigate = useNavigate();
  const dispatch = useDispatch();
  const { user } = useSelector((state) => state.users);
  const getExams = async () => {
    try {
      dispatch(ShowLoading());
      const response = await getAllExams();
      if (response.success) {
        setExams(response.data);
      } else {
        message.error(response.message);
      }
      dispatch(HideLoading());
    } catch (error) {
      dispatch(HideLoading());
      message.error(error.message);
    }
  };

  useEffect(() => {
    getExams();
  }, []);

  return (
    user && (
      <div>
        <PageTitle title={`Hi ${user.name}, Welcome to Quicki!`} />
        <div className="divider"></div>
        <Row gutter={[16, 16]}>
          {exams.map((exam) => (
            <Col span={6}>
              <div className="card-lg flex flex-col gap-1 p-2">
                <h1 className="text-2xl">{exam?.name}</h1>

                <h1 className="text-md">Category : {exam.category}</h1>

                <h1 className="text-md">Total Marks : {exam.totalMarks}</h1>
                <h1 className="text-md">Passing Marks : {exam.passingMarks}</h1>
                <h1 className="text-md">Duration : {exam.duration}</h1>
                <button
```

```
                  className="primary-outlined-btn text-md"
                  onClick={() => navigate(`/user/write-exam/${exam._id}`)}
                >
                  Start Test
                </button>
              </div>
            </Col>
          ))}
        </Row>
      </div>
    )
  );
}

export default Home;
```

7. Login → index.js

```
import React from 'react'
import { Form, message } from 'antd'
import { Link } from 'react-router-dom'
import { loginUser } from '../../../apicalls/users';
import { useDispatch } from 'react-redux';
import { HideLoading, ShowLoading } from '../../../redux/loaderSlice';

function Login() {
  const dispatch = useDispatch();
  const onFinish = async (values) => {
    try {
      dispatch(ShowLoading());
      const response = await loginUser(values);
      dispatch(HideLoading());
      if (response.success) {
        message.success(response.message);
        localStorage.setItem("token", response.data);
        window.location.href = "/";
      } else {
        message.error(response.message);
      }
    } catch (error) {
      dispatch(HideLoading());
      message.error(error.message);
    }
  }
  return (
    <div className="flex justify-center items-center h-screen w-screen bg-primary">
      <div className="card w-400 p-3 bg-white">
```

```
        <div className="flex flex-col">
          <div className="flex">
            <h1 className="text-2xl">QUICKI - LOGIN <i class="ri-login-circle-line"></i></h1>

          </div>
          <div className="divider"></div>
          <Form layout="vertical" className="mt-2" onFinish={onFinish}>
            <Form.Item name="email" label="Email">
              <input type="text" />
            </Form.Item>
            <Form.Item name="password" label="Password">
              <input type="password" />
            </Form.Item>
            <div className='flex flex-col gap-2'>
              <button type='submit' className='primary-contained-btn mt-2 w-100' >Log in</button>
              <Link to="/register" className='underline'>Don't have an account?Register here.</Link>
            </div>
          </Form>
        </div>
      </div>

    </div>
  )
}

export default Login
```

8. Register → index.js

```
import React from 'react'
import { Form, message } from 'antd'
import { Link, useNavigate } from 'react-router-dom'
import { registerUser } from '../../../apicalls/users'
import { useDispatch } from 'react-redux'
import { HideLoading, ShowLoading } from '../../../redux/loaderSlice'

function Register() {
  const dispatch = useDispatch();
  const navigate = useNavigate();
  const onFinish = async (values) => {
    try {
      dispatch(ShowLoading());
      const response = await registerUser(values);
      dispatch(HideLoading());

      if (response.success) {
        message.success(response.message);
```

```
      navigate("/login");
     } else {
      message.error(response.message);
     }
    } catch (error) {
     dispatch(HideLoading());
     message.error(error.message);
    }};
  return (
    <div className='flex justify-center items-center h-screen w-screen bg-primary'>
     <div className='card w-400 p-3 bg-white'>
      <div className="flex flex-col">
       <div className='flex'>
        <h1 className='text-2xl'>QUICKI - REGISTER
         <i class="ri-user-add-line"></i>
        </h1>
       </div>
       <div className='divider'></div>
       <Form layout='vertical' className='mt-2' onFinish={onFinish}>
        <Form.Item name='name' label='Name'>
         <input type='text' />
        </Form.Item>
        <Form.Item name='email' label='Email'>
         <input type='text' />
        </Form.Item>
        <Form.Item name='password' label='Password'>
         <input type='password' />
        </Form.Item>
        <div className='flex flex-col gap-2'>
         <button type='submit' className='primary-contained-btn mt-2 w-100'>Register</button>
         <Link to="/login" className='underline'>Already have an account?Login here.</Link>
        </div>
       </Form>
      </div>
     </div>

    </div>)}

export default Register;
```

ДОДАТОК Б. BackCode

1. server.js

```
const express = require("express");
const app = express();
require("dotenv").config();
app.use(express.json());
const dbConfig = require("./config/dbConfig");

const usersRoute = require("./routes/usersRoute");
const examsRoute = require("./routes/examsRoute");
const reportsRoute = require("./routes/reportsRoute");

app.use("/api/users", usersRoute);
app.use("/api/exams", examsRoute);
app.use("/api/reports", reportsRoute);

const port = process.env.PORT || 5000;
app.listen(port, () => {
  console.log(`Server listening on port ${port}`);
});
```

2. dbConfig.js

```
const mongoose = require('mongoose');

mongoose.connect(process.env.MONGO_URL)

const connection = mongoose.connection;

connection.on('connected', () => {

   console.log('Mongo DB Connection Successful');

})

connection.on('error', (err) => {

   console.log('Mongo DB Connection Faild');

})

module.exports = connection;
```

3. examsRoute.js

```
const router = require("express").Router();
const Exam = require("../models/examModel");
const authMiddleware = require("../middlewares/authMiddleware");
```

```
const Question = require("../models/questionModel");

// add test

router.post("/add", authMiddleware, async (req, res) => {
    try {
        // check if exam already exists
        const examExists = await Exam.findOne({ name: req.body.name });
        if (examExists) {
            return res
                .status(200)
                .send({ message: "Exam already exists", success: false });
        }
        req.body.questions = [];
        const newExam = new Exam(req.body);
        await newExam.save();
        res.send({
            message: "Exam added successfully",
            success: true,
        });
    } catch (error) {
        res.status(500).send({
            message: error.message,
            data: error,
            success: false,
        });
    }
});

// get all tests
router.post("/get-all-exams", authMiddleware, async (req, res) => {
    try {
        const exams = await Exam.find({});
        res.send({
            message: "Exams fetched successfully",
            data: exams,
            success: true,
        });
    } catch (error) {
        res.status(500).send({
            message: error.message,
            data: error,
            success: false,
        });
    }
});
```

```
// get test by id
router.post("/get-exam-by-id", authMiddleware, async (req, res) => {
    try {
        const exam = await Exam.findById(req.body.examId).populate("questions");
        res.send({
            message: "Exam fetched successfully",
            data: exam,
            success: true,
        });
    } catch (error) {
        res.status(500).send({
            message: error.message,
            data: error,
            success: false,
        });

    }
})

// edit test by id
router.post("/edit-exam-by-id", authMiddleware, async (req, res) => {
    try {
        await Exam.findByIdAndUpdate(req.body.examId, req.body);
        res.send({
            message: "Exam edited successfully",
            success: true,
        });
    } catch (error) {
        res.status(500).send({
            message: error.message,
            data: error,
            success: false,
        });

    }
});

// delete test by id
router.post("/delete-exam-by-id", authMiddleware, async (req, res) => {
    try {
        await Exam.findByIdAndDelete(req.body.examId);
        res.send({
            message: "Exam deleted successfully",
            success: true,
        });
    } catch (error) {
        res.status(500).send({
```

```
            message: error.message,
            data: error,
            success: false,
        });
    }
});

// add question to test
router.post("/add-question-to-exam", authMiddleware, async (req, res) => {
    try {
        // add question to Questions collection
        const newQuestion = new Question(req.body);
        const question = await newQuestion.save();

        // add question to test
        const exam = await Exam.findById(req.body.exam);
        exam.questions.push(question._id);
        await exam.save();
        res.send({
            message: "Question added successfully",
            success: true,
        });
    } catch (error) {
        res.status(500).send({
            message: error.message,
            data: error,
            success: false,
        });
    }

});

// edit questions in test
router.post("/eddit-question-in-exam", authMiddleware, async (req, res) => {
    try {
        // edit question in Questions collection
        await Question.findByIdAndUpdate(req.body.questionId, req.body);
        res.send({
            message: "Question edited successfully",
            success: true,
        });
    } catch (error) {
        res.status(500).send({
            message: error.message,
            data: error,
            success: false,
        });
```

```
    }

});

// delete question in test
router.post("/delete-question-in-exam", authMiddleware, async (req, res) => {
    try {
        // delete question in Questions collection
        await Question.findByIdAndDelete(req.body.questionId);

        // delete question in test
        const exam = await Exam.findById(req.body.examId);
        exam.questions = exam.questions.filter(
            (question) => question._id != req.body.questionId
        );
        await exam.save();
        res.send({
            message: "Question deleted successfully",
            success: true,
        });
    } catch (error) {

    }});

module.exports = router;
```

4. reportsRoute.js

```
const authMiddleware = require("../middlewares/authMiddleware");
const Exam = require("../models/examModel");
const User = require("../models/userModel");
const Report = require("../models/reportModel");
const router = require("express").Router();

//add report

router.post("/add-report", authMiddleware, async (req, res) => {
    try {
        const newReport = new Report(req.body);
        await newReport.save();
        res.send({
            message: "Attempt added successfully",
            success: true,
        });
    } catch (error) {
        res.status(500).send({
            message: error.message,
```

```
            data: error,
            success: false,
        });
    }
});

//get all reports

router.post("/get-all-reports", authMiddleware, async (req, res) => {
    try {
        const { examName, userName } = req.body;
        const exams = await Exam.find({
            name: {
                $regex: examName,
            }
        });

        const matchedExamIds = exams.map((exam) => exam._id);

        const users = await User.find({
            name: {
                $regex: userName,
            },
        });

        const matchedUserIds = users.map((user) => user._id);

        const reports = await Report.find({
            exam: {
                $in: matchedExamIds,
            },
            user: {
                $in: matchedUserIds,
            },
        }).populate("exam").populate("user").sort({ createdAt: -1 });
        res.send({
            message: "Attempt fetched successfully",
            data: reports,
            success: true,
        });
    } catch (error) {
        res.status(500).send({
            message: error.message,
            data: error,
            success: false,
        });
    }
```

```
});

//get all reports by user
router.post("/get-all-reports-by-user", authMiddleware, async (req, res) => {
   try {
      const reports = await Report.find({ user: req.body.userId
}).populate("exam").populate("user").sort({ createdAt: -1 });
      res.send({
         message: "Attempt fetched successfully",
         data: reports,
         success: true,
      });
   } catch (error) {
      res.status(500).send({
         message: error.message,
         data: error,
         success: false,
      });
   }
});

module.exports = router;
```

5. usersRoute.js

```
const router = require('express').Router();
const User = require('../models/userModel');
const bcrypt = require('bcryptjs');
const jwt = require("jsonwebtoken");
const authMiddleware = require('../middlewares/authMiddleware');

// user registration

router.post("/register", async (req, res) => {
  try {
    // check if user already exists
    const userExists = await User.findOne({ email: req.body.email });
    if (userExists) {
      return res
        .status(200)
        .send({ message: "User already exists", success: false });
    }

    // hash password
    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(req.body.password, salt);
```

```
    req.body.password = hashedPassword;

    // create new user
    const newUser = new User(req.body);
    await newUser.save();
    res.send({
      message: "User created successfully",
      success: true,
    });
  } catch (error) {
    res.status(500).send({
      message: error.message,
      data: error,
      success: false,
    });
  }
});

// user login

router.post("/login", async (req, res) => {
  try {
    // check if user exists
    const user = await User.findOne({ email: req.body.email });
    if (!user) {
      return res
        .status(200)
        .send({ message: "User does not exist", success: false });
    }

    // check password
    const validPassword = await bcrypt.compare(
      req.body.password,
      user.password
    );
    if (!validPassword) {
      return res
        .status(200)
        .send({ message: "Invalid password", success: false });
    }

    const token = jwt.sign({ userId: user._id }, process.env.JWT_SECRET, {
      expiresIn: "1d",
    });

    res.send({
      message: "User logged in successfully",
```

```
      success: true,
      data: token,
    });
  } catch (error) {
    res.status(500).send({
      message: error.message,
      data: error,
      success: false,
    });
  }
});

// get user info
router.post("/get-user-info", authMiddleware, async (req, res) => {
  try {
    const user = await User.findById(req.body.userId);
    res.send({
      message: "User info fetched successfully",
      success: true,
      data: user,
    });
  } catch (error) {
    res.status(500).send({
      message: error.message,
      data: error,
      success: false,
    });
  }
});

module.exports = router;
```