МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
_____
(підпис)

червня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Телеграм-бот для персоналізованих тренувань: створення системи надання відео-вправ та індивідуальних тренувальних програм»
здобувача групи ІН-06-2 Миненка Артема Володимировича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Артем МИНЕНКО
_____
(підпис)

Керівник, доцент, кандидат наук Зоя МАСЛОВА
_____
(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри

_____ **Ігор ШЕЛЕХОВ**
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
на здобуття освітнього ступеня бакалавра
зі спеціальності 122 – Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-06-2 Миненка А.В.

1. Тема роботи: «Телеграм-бот для персоналізованих тренувань: створення системи надання відео-вправ та індивідуальних тренувальних програм»
затверджую наказом по СумДУ від _____
2. Термін здачі здобувачем кваліфікаційної роботи_____
3. Вхідні дані до кваліфікаційної роботи
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для розробки інформаційного та програмного забезпечення системи. 3) Розробка інформаційного та програмного забезпечення. 4) Аналіз результатів._____
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до _____ (підпис)

Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	Аналіз проблеми предметної області, постановка й формування завдань дослідження		
2	Огляд технологій для розробки інформаційної системи		
3	Розробка інформаційної системи		
4	Аналіз отриманих результатів		
5	Оформлення пояснювальної записки до кваліфікаційної роботи		

Здобувач вищої освіти _____ (підпис)

Керівник _____ (підпис)

АНОТАЦІЯ

Записка: 59 стор., 21 рис., 3 додаток, 14 джерел.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи належить до актуальних напрямків розвитку інформаційних технологій, оскільки в розробка мобільних додатків залишається актуальною з численних вони відіграють ключову роль у сучасному технологічному ландшафті та суспільстві загалом.

Об'єкт дослідження – інформаційна система для маніпуляції та відображення даних всередині мобільного застосунку.

Мета роботи – розробити інформаційну модель в вигляді мобільного додатку для надання послуг зі сфери фітнесу.

Методи дослідження – огляд та аналіз аналогічних рішень, вивчення потреб ринку, виявлення поточних тенденцій.

Результати – було досягнуто всіх першочергових цілей, розроблено базу даних, API сервер та прогресивний веб-додаток (PWA) з використанням найпопулярніших та найефективніших технологій. Розгорнуто проєкт на Netlify та Digital Ocean.

ІНФОРМАЦІЙНА СИСТЕМА, МОБІЛЬНИЙ ЗАСТОСУНОК, TELEGRAM API, JAVASCRIPT, REACT, NODE.JS.

ЗМІСТ

ВСТУП

В епоху стрімкого технологічного розвитку та зростаючої популярності мобільних пристроїв, створення мобільного додатку для свого продукту стає ключовим завданням для багатьох організацій та розробників. Мобільні додатки стали невід'ємною частиною сучасного життя, сприяючи спрощенню різноманітних завдань і покращенню взаємодії зі світом інформації та послуг.

Актуальність роботи: Розробка мобільних додатків залишається актуальною з численних причин, оскільки вони відіграють ключову роль у сучасному технологічному ландшафті та суспільстві загалом. Кількість користувачів смартфонів та планшетів постійно зростає. Мобільні пристрої стали необхідною частиною повсякденного життя, що створює величезний потенціал для досягнення аудиторії через мобільні додатки. Водночас, з доступом до швидкого Інтернету на мобільних пристроях стало легше використовувати онлайн-сервіси та взаємодіяти з додатками з будь-якого місця та в будь-який час. Мобільні застосунки надають зручний спосіб доступу до інформації, розваг, послуг та товарів. Вони розробляються з урахуванням користувацької зручності та максимальної доступності. Для підприємств розробка додатків може забезпечити конкурентні переваги, поліпшити обслуговування клієнтів, збільшити продажі та підвищити лояльність користувачів.

Загалом, розробка мобільних застосунків є ключовою для задоволення потреб користувачів, розвитку бізнесу та використання сучасних технологій для досягнення різноманітних цілей. Таким чином, ця галузь залишається надзвичайно актуальною та перспективною для розробників та підприємств.

Метою роботи є розробити інформаційну модель в вигляді мобільного додатку для надання послуг зі сфери фітнесу.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Огляд предметної області

Останні події, пов'язані з глобальним карантином та воєнним станом, вплинули на всі сфери життя, включаючи бізнес-середовище. Одним із важливих викликів для бізнесу стало переведення діяльності в режим онлайн. Однак, цей перехід також виявився вдалим рішенням для багатьох компаній, які змогли адаптуватися до нових умов. Це відкрило нові можливості та переваги, але й викликало важкості у впровадженні та конкуренції. Важливою є гнучкість і швидкість адаптації, які дозволяють бізнесу виживати та розвиватися в змінному середовищі.

1.2 Огляд існуючих рішень

У сучасному інтернеті спостерігається величезна кількість спортивних додатків, які відповідають різним цілям та потребам користувачів. Важливо відзначити, що прагнення до покращення фізичних можливостей завжди лишається актуальним та зростає з часом. Ця тенденція зумовлена кількома факторами. По-перше, зростаюча свідомість про важливість здорового способу життя і регулярної фізичної активності зробила фітнес і спорт популярними серед широкого кола людей. По-друге, доступність сучасних технологій, особливо мобільних додатків, робить можливим отримання індивідуальних тренувальних програм та порад за кілька клацань.

Аналізуючи спортивні фітнес додатки, які спрямовані на роботу в спорт клубі було виділено декілька варіантів:

- Nike Training Club
- Fitness App – Muscle Gym Workout
- BRONX FitnessHub BRO-BOT (Telegram bot)

Nike Training Club було обрано як ідеальний додаток. Насправді на площадках є застосунки конкурентів (PUMATRAC, Gymshark Training and Fitness, Adidas Run club, і купа інших), які мають дуже схожий функціонал.

Причиною, за якої саме Nike виявився в підбірці – він абсолютно безкоштовний. Тобто весь його функціонал доступний будь-якому користувачу. Також застосунок має систему нагород та інтуїтивно зрозумілу та зручну статистику.

Принцип програми полягає у наступному:

1. Користувач заповнює свої біометричні параметри, вказує свої спортивні цілі та побажання.
2. Отримує список тренувань, які відповідають обраним цілям.
3. Виконує тренування, отримує за це заохочувальні та мотивуючі нагороди.
4. Після виконаного тренування воно переходить у вкладку “Активності” та залишається в історії для статистики, або, в випадку коли тренування не було завершене – для продовження цього самого тренування.

Кожне тренування містить відео з детальним відображенням техніки вправи, описом вправи та таймером

Загалом, додаток Nike Training Club інтуїтивно зрозумілий, зручний, не навантажений зайвим функціоналом та має безліч гарних відгуків. Доступний для скачування як на iOS, так і на Android.

Fitness App – Muscle Gym Workout

Даний застосунок було обрано через високу оцінку в App Store.

Серед функціоналу він має:

- обширний список тренувань (рисунок 1.1);

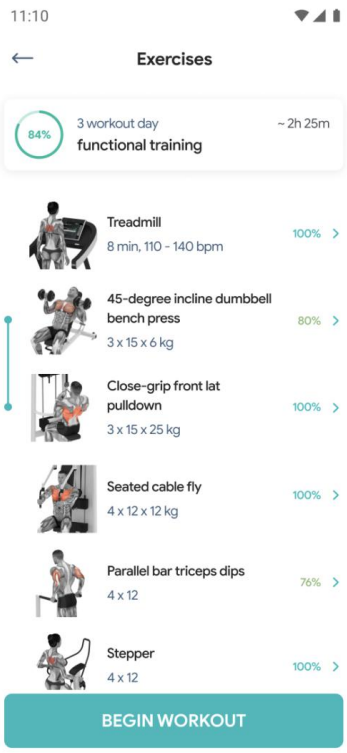


Рисунок 1.1 – Приклад тренувань Fitness App

- розумні плани занять;
- зайняття для чоловіків та жінок;
- корисна аналітика (рисунок 1.2);

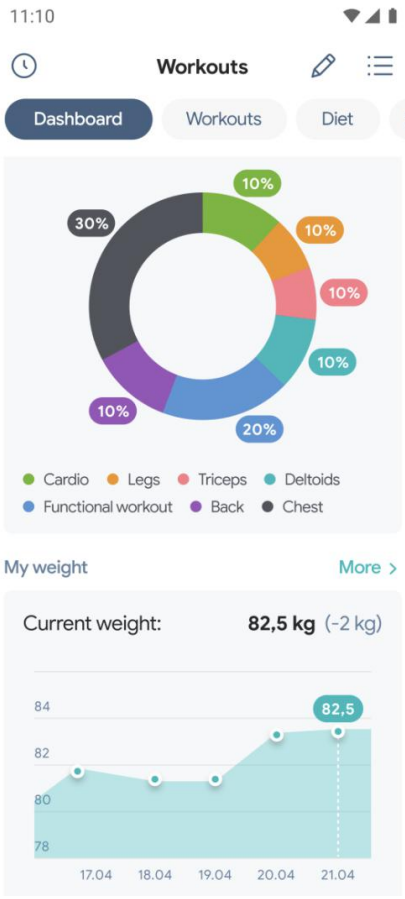


Рисунок 1.2 – Інформаційна панель користувача Fitness App

- спортивна стрічка подій;
- план харчування (рисунок 1.3);

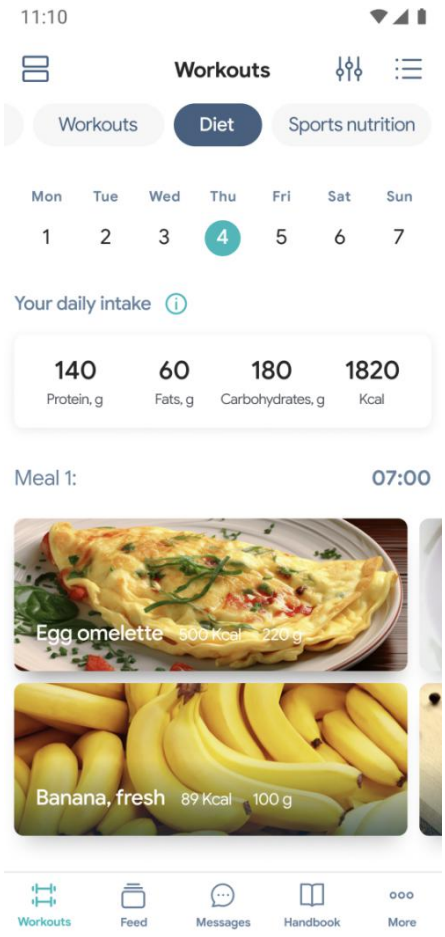


Рисунок 1.3 – Розділ харчування Fitness App

- чат з однодумцями;
- сумарна статистика по користувачу.

При реєстрації потрібно вказати свої біометричні дані, після чого застосунок надає доступ до безлічі функцій. До них входять:

- тренування, які підходять до вказаних даних;
- статистика по спортивним показникам;
- стрічка подій;
- стрічка дієт під різні фітнес цілі;
- і інші.

З тренуванням також надаються 3D фото, на яких показано приблизну техніку виконання вправи та задіяну групу м'язів. Нижче опис виконання та таймер.

Аналізуючи даний застосунок можна виділити декілька суттєвих мінусів:

- Надмірна навантаженість програми: Застосунок включає велику кількість функцій та послуг, що може ускладнювати користування особливо для новачків. Поміж тренуваннями вибухає багато інших пропозицій, що може відволікати користувача та знижувати зосередженість під час тренування.
- Наявність соцмережі в програмі: Існування невеликої соціальної мережі в межах застосунку може негативно впливати на користувацький досвід, особливо якщо вона не є основною функцією програми та може відволікати від основних завдань.
- Недостатня якість ілюстративних інструкцій: Картинки та відео в програмі не завжди чітко передають техніку виконання вправ, що може призвести до неправильного виконання та, внаслідок цього, травмування користувача або втрату мотивації.
- Перевищена кількість статистики: Велика кількість цифрових показників та статистики може перевантажувати інтерфейс, відволікаючи користувача та знижуючи зручність використання програми.
- Заплутаність інтерфейсу: Інтерфейс програми може бути перенасиченим активними елементами, що робить його складним для розуміння та навігації для користувачів.

Загалом додаток є не зручним в використанні та має багато платних планів. Безплатний контент доступний тільки для новачків в спорті.

BRONX FitnessHub BRO-BOT це програма, яка має схожий функціонал, але побудована на базі Telegram Bot API.

Цей підхід сподобався тим, що користувачу не потрібно проходити довгий етап реєстрації, не потрібно завантажувати додаток, а користуватись послугами можна взагалі з будь-якого гаджету, який має Telegram.

Бот надає наступний послуги:

- програми тренувань;
- розклад та вартість занять в спорт клубі;
- локації тренажерних залів;
- секція відгуків

Програми тренувань включають тільки самі вправи, без текстових чи ілюстраційних пояснень. Відображається відразу весь період тренування за раз. Цей підхід може дуже негативно вплинути на користувача, який є новачком, так як тренування передбачені тільки на людей, які вже мають деякі знання в галузі спорту.

Також великим недоліком є повторне надсилання клавіатури меню, та іншої інформації. Перенасиченість або надмірний спамінг повідомленнями можуть негативно позначитися на базі клієнтів(рисунок 1.4).

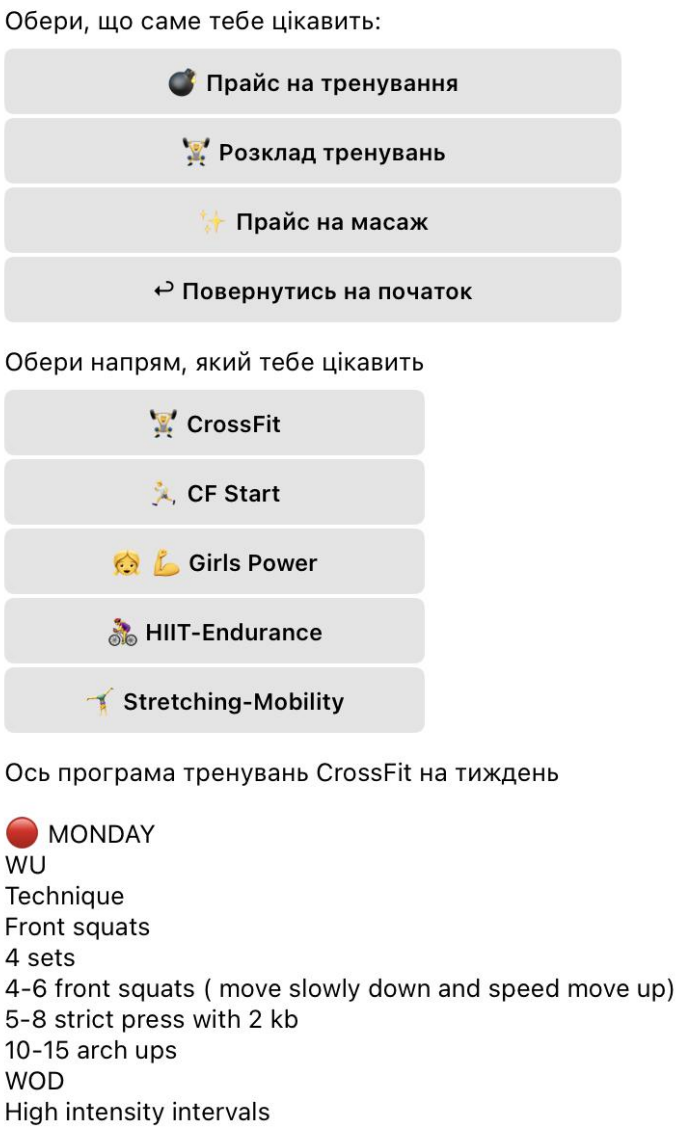


Рисунок 1.4 – Приклад “захламління” чату користувача повідомленнями

1.3 Постановка задачі

Спираючись на інформацію, отриману після проведення аналізу схожих програм, можна визначити деякі ключові аспекти, які слід врахувати при розробці застосунку:

1. Розробити зручний, інтуїтивно зрозумілий інтерфейс, який не буде перенасиченим інформацією, але одночасно надаватиме всі необхідні функції користувачам.

2. Розробити механізм персоналізації контенту для користувачів, що дозволить пропонувати індивідуалізовані тренувальні програми та рекомендації.

3. Забезпечити безпеку та конфіденційність: Важливою задачею є забезпечення високого рівня безпеки та захисту персональних даних користувачів.

4. Забезпечення крос-платформленості. Можливість використовувати програму на різних гаджетах.

З-поміж основних функцій додаток повинен мати:

1. Можливість створювати вправи
2. Можливість створювати тренування за створеними вправами
3. Можливість залишати відгуки
4. Систему ролей, яка буде розподіляти можливості використання додатку для різних користувачів.
5. Можливість сортувати, або фільтрувати контент за декількома типами.

2. ВИБІР МЕТОДІВ ВИРІШЕННЯ ЗАДАЧІ

Для вирішення поставлених задач необхідно дослідити та обрати необхідні для розробки технології.

2.1 Засоби розробки інтерфейсу користувача

Є декілька варіантів розробки мобільного застосунку, які можна розглянути в залежності від потреб і можливостей:

Нативний застосунок:

- iOS (Swift/Objective-C): Розробка спеціально під iOS для iPhone та iPad.
- Android (Java/Kotlin): Розробка для платформи Android для різних пристроїв та версій ОС.

Гібридний застосунок:

- React Native: Використання JavaScript та React для створення застосунку, який може працювати як на iOS, так і на Android.
- Flutter: Використання Dart та Flutter SDK для створення кросплатформеного застосунку, який наближено до нативного за виглядом та продуктивністю.

веб-застосунок: Прогресивні веб-застосунки (PWA): Використання веб-технологій (HTML, CSS, JavaScript) з можливістю додавати певні функції, що раніше були доступні лише для нативних застосунків, такі як робота оффлайн, повідомлення та інші.

Розробка веб-застосунку була обрана в якості оптимального варіанту для проєкту з кількох ключових причин, які визначають його швидкість і гнучкість серед інших методів розробки мобільних додатків.

По-перше, веб-застосунки використовують відкриті веб-стандарти, такі як HTML, CSS та JavaScript, що робить їх доступними на будь-якому пристрої з веб-браузером, незалежно від платформи (комп'ютери, планшети, смартфони).

По-друге, розробка веб-застосунків зазвичай є швидшою та ефективнішою за рахунок використання сучасних фреймворків та бібліотек для веб-розробки, таких як React.js, Angular, або Vue.js. Ці інструменти дозволяють створювати високоякісний та динамічний веб-інтерфейс з меншими зусиллями порівняно з розробкою нативних або гібридних застосунків. Також, веб-додатки дозволяють більш швидке впровадження змін та оновлень, оскільки зміни у веб-коді можна вносити безпосередньо на сервері без необхідності оновлення самого додатку на кожному пристрої користувача. Це робить розробку та підтримку додатку більш гнучкою та ефективною у майбутньому.

В якості інструменту для побудови інтерфейсу було обрано Javascript, та бібліотеку React, так як вона має велику спільноту розробників, численні бібліотеки та інструменти, які спрощують розробку та розширюють можливості застосунку. Також React використовує JSX–розширення синтаксису JavaScript, яке дозволяє писати HTML-подібний код всередині JavaScript. Це робить шаблонізацію більш інтуїтивно зрозумілою та зручною.

Для роботи з пагінацією сторінок було обрано бібліотеку React Router.

React Router — це бібліотека для керування маршрутизацією у програмах, створених на базі React. Вона дозволяє створювати та керувати шляхами (routes) в односторінкових програмах (SPA), що дає можливість переходу між різними компонентами або сторінками без перезавантаження сторінки.

Для прискорення розробки та шаблонізації стилів буде використовуватись Tailwind CSS та Ant Design.

Tailwind CSS – це бібліотека, яка дозволяє стилізувати елементи за допомогою css класів (рисунок 2.1). Tailwind CSS стає все більш популярним серед розробників завдяки своїй гнучкості та ефективності.

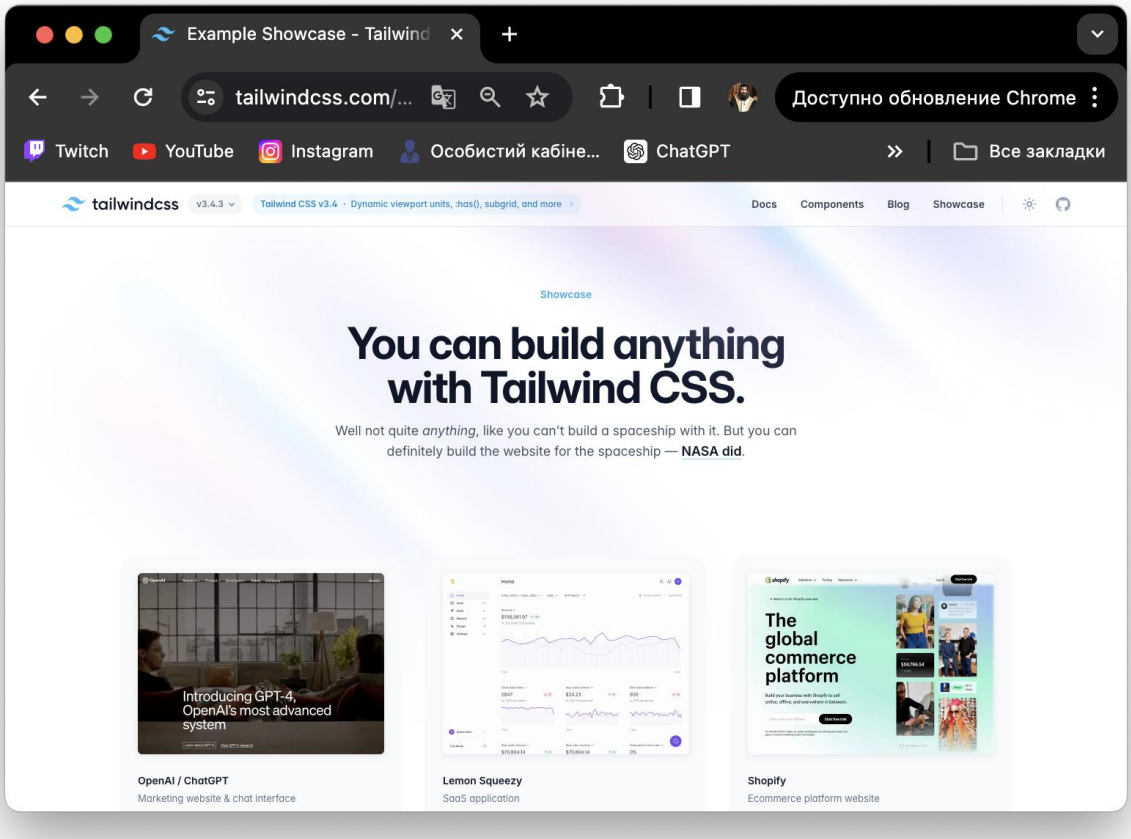


Рисунок 2.1 – Сайт Tailwind CSS

Ant Design – бібліотека готових, стилізованих React - компонентів, яка допомагає прискорити розробку, має високу якість компонентів, консистентий дизайн, велику гнучкість та широку підтримку і документацію (рисунок 2.2).

Серед бібліотек компонентів React також можна виділити Material UI, Bootstrap та Chakra UI. Це все є аналоги Ant Design. Вони всі мають свої переваги і недоліки, але обрано було саме Ant Design через більшу кількість власного досвіду.

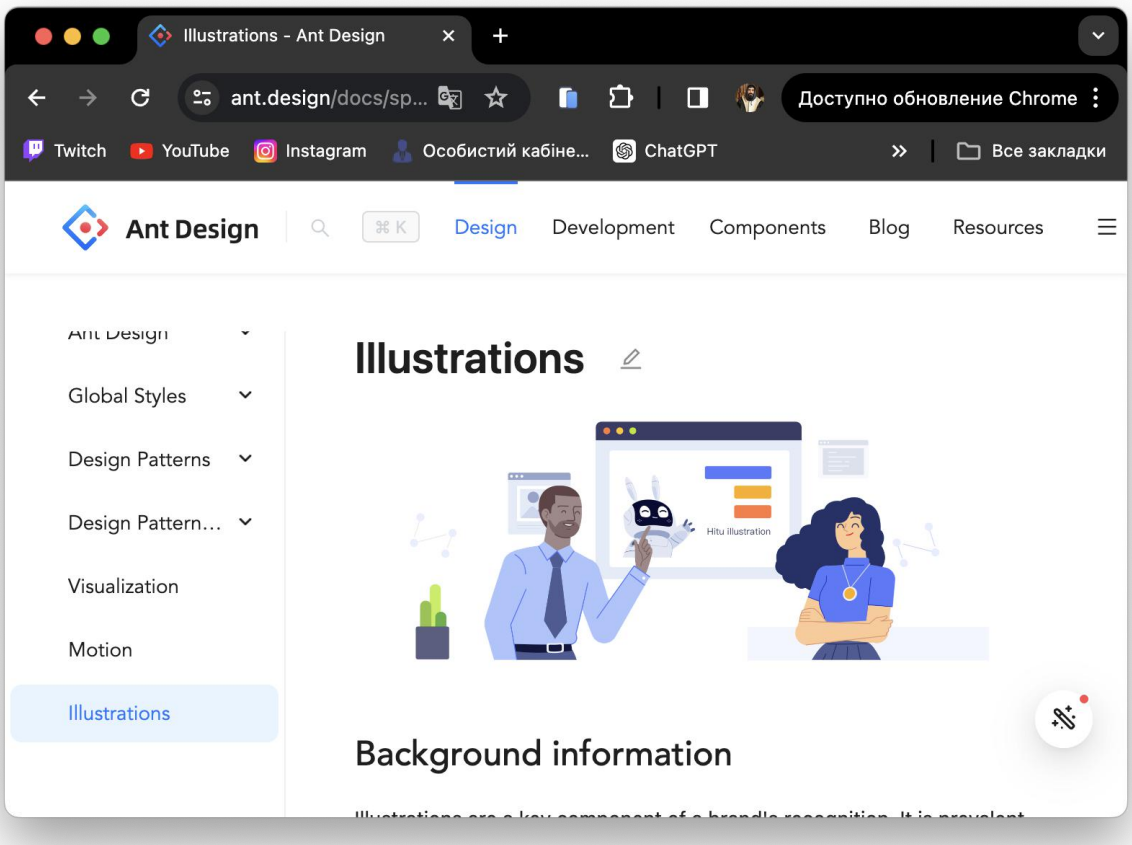


Рисунок 2.2 – Головна сторінка бібліотеки Ant Design

Для роботи з http запитами буде використано бібліотеку axios та React Query.

Axios – це бібліотека, написана на основі Fetch API, яка по суті є візуально спрощеною реалізацією бібліотеки Fetch.

React Query — це бібліотека для управління станом серверних даних у застосунках на базі React. Вона значно спрощує роботу з асинхронними запитами до серверу, кешуванням, синхронізацією та оновленням даних.

До появи React Query часто використовували global state managers, такі як MobX, Redux, Jotai, Justand та купа інших. Для кожного запиту на сервер раніше потрібно було самостійно писати логіку для зберігання даних, їх редагування, відстеження статусу запиту та обробки помилок. React Query автоматизує всі ці процеси, надаючи готові рішення "під капотом" (рисунок 2.3),

а також має додаткові можливості, які значно спрощують роботу розробника та економлять час.

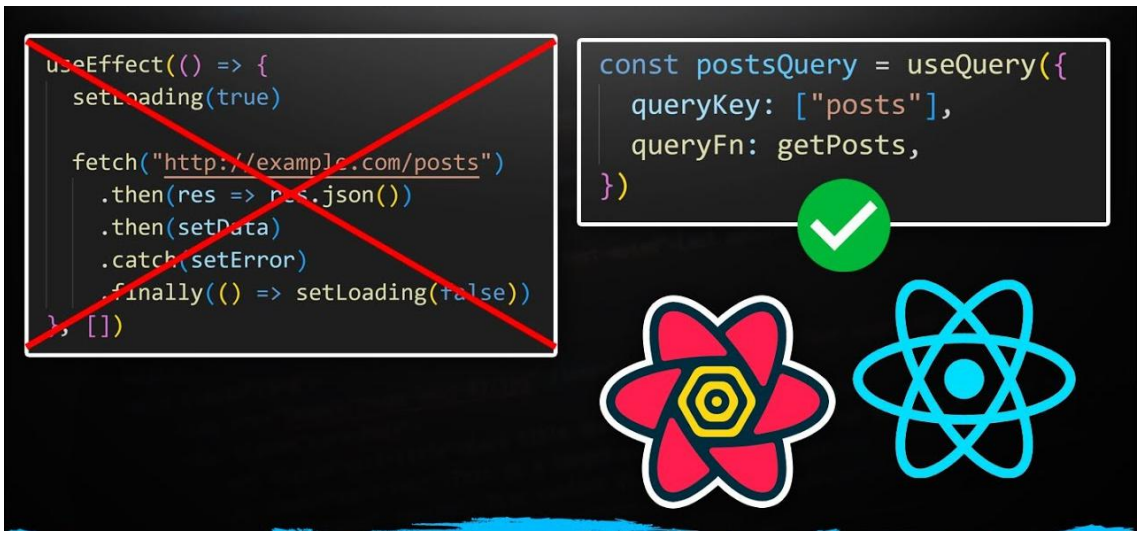


Рисунок 2.3 – Приклад використання React Query

Додатково також будуть використані наступні бібліотеки:

- React Icons – це бібліотека, яка надає інтеграцію популярних іконок з React. Вона дозволяє легко додавати іконки з різних наборів, таких як FontAwesome, Material Design, Ionicons та інших, без необхідності імпортувати кожну іконку окремо.
- React Video – це бібліотека для вбудовування відео в React-застосунки. Вона забезпечує зручний компонент для відтворення відео з налаштовуваними параметрами, такими як автозапуск, відображення кнопок керування, обробка подій тощо.
- DND-kit – це сучасна бібліотека для реалізації drag-and-drop функціональності в React. Вона забезпечує простий API для створення перетягуваних елементів та зони прийому (droppable areas), з фокусом на продуктивність і користувацький досвід.

- Framer-motion – це потужна бібліотека для анімацій у React. Вона надає простий і гнучкий API для створення складних анімацій та переходів між станами, з підтримкою жестів та фізичних симуляцій.

2.2 Засоби розробки серверної частини застосунку

В угоду швидкості написання коду, для серверної частини було вирішено використовувати Node.js та відповідно JavaScript.

Node.js це середовище виконання js коду. Якщо ще простішою мовою то це виконання js скриптів окремо від браузера.

Плюсами використання в проєкті node являється його асинхронна модель, знайомий синтаксис мови програмування, а також велику кількість зручних пакетів, бібліотек та фреймворків.

Сервер буде реалізований за допомогою фреймворку Express. Це найвідоміший фреймворк для побудови веб-застосунків. Він реалізований як вільне і відкрите програмне забезпечення під ліцензією MIT. За допомогою express щоденно створюються сотні проєктів.

Також для зберігання, додавання та роботою з даними знадобиться база даних.

MySQL та PostgreSQL являються найпоширенішими реляційними базами даних які використовуються у різних за складністю рішеннях. Кожна з них має свої переваги і недоліки. Ось порівняння PostgreSQL і MySQL:

Плюси PostgreSQL:

- Розширені можливості SQL: PostgreSQL підтримує більше розширених функцій SQL порівняно з MySQL, такі як віконні функції, рекурсивні запити та багато інших, що робить його більш потужним для складних операцій з даними.
- ACID-сумісність: PostgreSQL відомий своєю надійністю і високою рівновагою ACID (Atomicity, Consistency, Isolation, Durability), що робить його популярним в сферах, де потрібна надійність та консистентність даних.

- Геопросторові розширення: PostgreSQL має вбудовану підтримку геопросторових даних та географічних запитів, що робить його ідеальним для геолокаційних додатків та систем інформації про геопростір. Спільнота та підтримка: PostgreSQL має активну спільноту користувачів і розробників, а також доступ до професійної підтримки, що допомагає вирішувати проблеми та отримувати оновлення.
- Розширені типи даних: PostgreSQL дозволяє користувачам визначати свої власні типи даних, оператори та функції, що дозволяє створювати більш складні структури даних.

Мінуси PostgreSQL:

- Більший обсяг ресурсів: PostgreSQL може бути більш вимогливим до ресурсів порівняно з MySQL, що може призвести до вищих витрат на обладнання та пам'ять.
- Менше поширеність: Незважаючи на свою популярність, PostgreSQL не так поширений, як MySQL, і це може змусити мати справу з меншою кількістю розробників та адміністраторів, які знають цю СУБД.

Плюси MySQL:

- Простота використання: MySQL відомий своєю простотою та швидкістю встановлення та налаштування, що робить його популярним в середовищах з обмеженими ресурсами.
- Широке поширення: MySQL є однією з найпоширеніших СУБД у світі, і вона має велику спільноту користувачів та підтримку.

Мінуси MySQL:

- Обмежені можливості SQL: MySQL може бути менш потужним у плані розширених SQL-функцій порівняно з PostgreSQL.
- Обмежена підтримка геоданих: Підтримка геоданих у MySQL менш розширена та обмежена порівняно з PostgreSQL.
- Проблеми з ACID-сумісністю: В окремих випадках MySQL може мати проблеми з виконанням консистентних транзакцій (залежно від конфігурації та версії).

В якості БД буде використано саме PostgreSQL, так як він надає більше можливостей, а також підтримується на більшості платформах і легко встановлюється/конфігурується.

Для роботи з базою даних буде використана бібліотека Sequelize.

Sequelize – це популярна ORM (Object-Relational Mapping) для Node.js, яка дозволяє працювати з реляційними базами даних (такими як PostgreSQL, MySQL, MariaDB, SQLite, та MSSQL) за допомогою JavaScript. Sequelize спрощує взаємодію з базами даних, забезпечуючи зручний інтерфейс для визначення моделей, виконання запитів та керування транзакціями.

Переваги використання Sequelize:

- Зручність: Sequelize спрощує роботу з реляційними базами даних, дозволяючи використовувати JavaScript для визначення моделей та виконання запитів.
- Агностичність до бази даних: Sequelize підтримує різні реляційні бази даних, що дозволяє легко переключатися між ними без значних змін у коді.
- Підтримка складних запитів: Бібліотека дозволяє виконувати складні запити, включаючи об'єднання таблиць, підзапити та агреговані функції.
- Масштабованість: Sequelize підходить для проєктів будь-якого масштабу, від простих застосунків до великих комерційних проєктів.

2.3 Робота з Telegram Bot API, Mini app API

Для роботи з telegram api потрібно мати токен – унікальний ключ, який буде доступний тільки одній людині. По цьому токену телеграм надає безліч функції всередині свого сервісу.

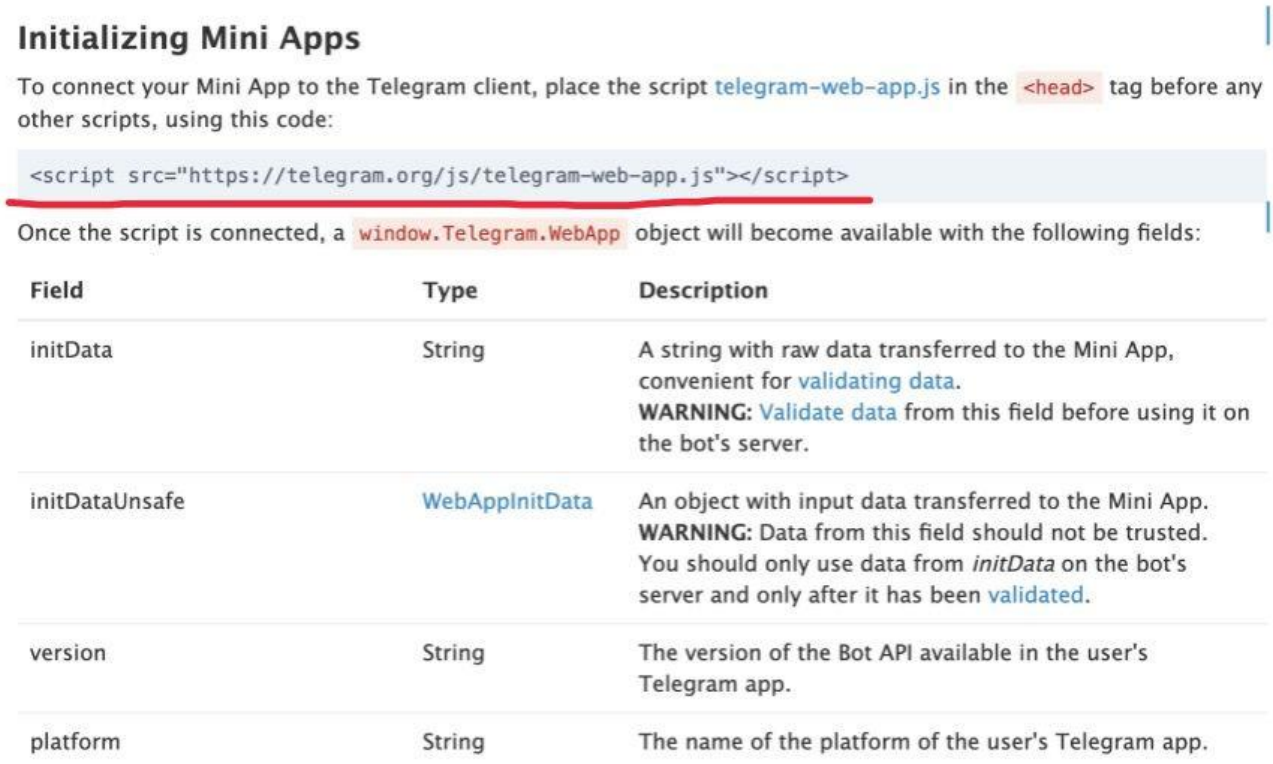
Для отримання токену потрібно звернутись до головного боту – BotFather (рисунок 2.4).

Рисунок 2.4 – Бот, який надає послуги по створенню редагуванню ботів

Для змоги використовувати сторонні веб сервіси в документації до Telegram API є розділ Web-Apps.

Telegram Mini App дозволяє запускати js скрипти прямо з чату. Після конфігурації клавіатури та налаштування потрібної адреси telegram буде запускати невеличкий локальний браузер, всередині якого і буде працювати веб-застосунок.

Для того щоб отримувати додаткову інформацію про користувача, поточну тему і т.д., на головну сторінку в тег *<head>* потрібно додати скрипт, який надає сам telegram. Після чого в глобальному об’єкті window з’явиться нове поле – Telegram (рисунок 2.5).



Initializing Mini Apps

To connect your Mini App to the Telegram client, place the script telegram-web-app.js in the `<head>` tag before any other scripts, using this code:

```
<script src="https://telegram.org/js/telegram-web-app.js"></script>
```

Once the script is connected, a `window.Telegram.WebApp` object will become available with the following fields:

Field	Type	Description
initData	String	A string with raw data transferred to the Mini App, convenient for validating data. **WARNING:** Validate data from this field before using it on the bot's server.
initDataUnsafe	WebAppInitData	An object with input data transferred to the Mini App. **WARNING:** Data from this field should not be trusted. You should only use data from *initData* on the bot's server and only after it has been validated.
version	String	The version of the Bot API available in the user's Telegram app.
platform	String	The name of the platform of the user's Telegram app.

Рисунок 2.5 – Документація Telegram Api до mini apps

З цього об’єкту потрібні будуть наступні поля:

- onClose – метод, при виклику якого вікно з веб-застосунком буде закриватись.
- initDataUnsafe – об’єкт, який містить загальну інформацію про чат та співрозмовника.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ

Перед початком розробки гарноюипрактикою вважається створити git-репозиторій, для зберігання файлів, зручною міграцією застосунку та контролю версій.

Git — це система контролю версій, яка дозволяє розробникам відстежувати зміни в коді, співпрацювати з іншими розробниками та зберігати історію проєкту. Створення репозиторію Git є першочерговим кроком у більшості проєктів розробки програмного забезпечення.

Для ініціалізацію git-репозиторію потрібно перейти в локальний каталог, де буде знаходитись майбутній проєкт за допомогою консолі, та, якщо git вже встановлено – ініціалізувати репозиторій за допомогою команди git init. Для збереження змін в каталозі та відправки оновленої версії на сервер використовують наступні команди:

1. **git add .** – додає всі нові файли з каталогу до системи git.
2. **git commit -m “<повідомлення з описом про поточні зміни>”** – фіксує додані файли.
3. **git push** – надсилає зафіксовані дані на сервер.

3.1 Розробка моделі бази даних

Застосунок буде мати наступні сутності:

- Сутність USER – таблиця, в якій будуть зберігатись додаткові дані про користувачів (так як застосунок буде побудовано спільно з Telegram API, необхідність в збереженні всієї інформації про користувача відсутня, так як вона вже зберігається у Telegram). USER буде мати такі поля:

- USERNAME – Для можливості користувачів в подальшому змінювати своє ім'я.

- ChatID – Для синхронізації користувачів з базами telegram і локальною БД
- GENDER (optional) – Для зберігання інформації щодо статі користувача, та надання більш напрямлених тренувань.
- ROLE – Для ідентифікації ролі користувача, та його можливостей в рамках застосунку.

- Сутність TRAINING – головна таблиця, яка буде мати інформацію про тренування. Вона буде мати такі поля:
  - LEVEL – Рівень навичок користувача, для визначення та надання більш направлених видів тренувань.
  - GENDER – Для якої статі пасує тренування.
  - TITLE – Назва тренування.
  - CONTENT – Опис до тренування.
  - IMAGE – Прев'ю зображення тренування.
  - EXEC_TIME – Приблизний час для проходження всіх вправ з тренування.

- Сутність REVIEWS – таблиця відгуків. В ній буде зберігатись інформація про відгуки після проходження тренувань. Вона має відношення до таблиці USER та TRAINING. Один користувач буде мати лише одну можливість залишити відгук на одне конкретне тренування. Поля таблиці:
  - USER_ID – Ідентифікатор користувача.
  - TRAINING_ID – Ідентифікатор тренування.
  - REVIEW – Текст відгуку.
  - RATING – Оцінка тренування від 0 до 5.

- Сутність EXERCISE – таблиця, яка буде містити інформацію про одну вправу. Сутність має відношення до таблиці TRAINING як багато-до-багатьох, тобто одне тренування може мати багато вправ, одна вправа може бути у багатьох тренуваннях. Вона буде мати наступні поля:

  - TITLE – Назва вправи.
  - CONTENT – Опис до виконання вправи.
  - VIDEO – Відео з наглядним прикладом виконання вправи.

- Сутність TYPES – Зберігає типи тренувань. наприклад: “розминка”, “заминка”, “кардіо тренування” і т.д.

  - NAME – Назва типу.

- Сутність CATEGORIES – Зберігає категорії тренувань, наприклад: “Для схуднення”, “Для набору м’язів”.

  - NAME – Назва категорії.

Нижче наведено ERD діаграму створеної бази даних (рисунок 3.1)

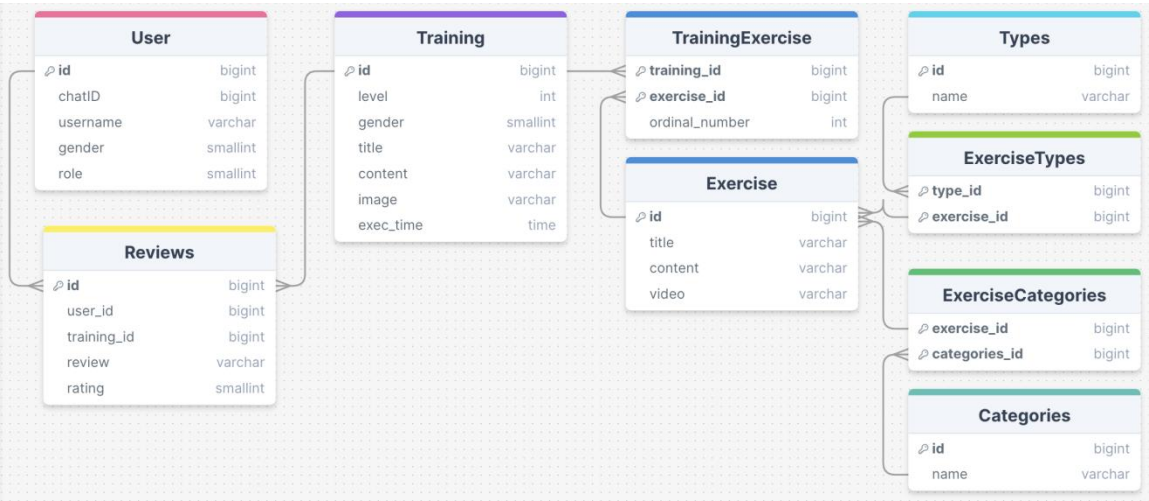


Рисунок 3.1 – ERD-діаграма розробленої моделі бази даних

3.2 Розробка серверної частини

Робота по розробці серверної серверної частини починається з ініціалізації проєкту та встановлення залежностей.

Для ініціалізації проєкту потрібно перейти в його каталог. Для серверу каталог буде мати назву server. В папці server потрібно прописати команду npm init, яка створить декілька файлів: package.json, package-lock.json.

package.json - це основний файл конфігурації для проєкту Node.js. Він містить метадані про проєкт, а також список залежностей, необхідних для його роботи.

Основні поля файлу:

- name: Ім'я проєкту.
- version: Версія проєкту.
- description: Короткий опис проєкту.
- main: Точка входу у проєкт (зазвичай головний файл, який буде запущено).
- scripts: Сценарії, які можна запускати за допомогою команди npm run.
- repository: Інформація про репозиторій вихідного коду проєкту.
- keywords: Список ключових слів, пов'язаних із проєктом.
- author: Автор проєкту.
- license: Тип ліцензії проєкту.
- dependencies: Список залежностей, необхідних для роботи проєкту.
- devDependencies: Залежності, що використовуються тільки в процесі розробки.

Файл package-lock.json

package-lock.json - це файл, який автоматично створюється та оновлюється npm під час встановлення залежностей. Він фіксує точні версії всіх встановлених пакетів, включаючи їхні залежні пакети. Цей файл

забезпечує детерміновані установки, тобто гарантує, що залежності будуть встановлені однаково на всіх машинах.

Після встановлення всіх залежностей (npm install <назва бібліотеки>) потрібно створити головний файл, який буде запускати сервер.

В головному файлі повинна бути ініціалізація та запуск серверу бази даних, серверу бота, та серверу для обробки API запитів.

Після створення подібного файлу в package.json потрібно вказати команду запуску серверу. В даному випадку команда npm run start буде виконувати команду nodemon app.js, що значить, що сервер (app.js) буде запущено за допомогою nodemon.

Nodemon – інструмент, яких допомагає в розробці Node.js проєкту, автоматично оновлюючи node-застосунок коли в каталозі з’являються якісь зміни до файлів.

Далі потрібно налаштувати архітектуру проєкту. Відомо що в ньому повинні бути моделі бази даних, маршрути, по яким будуть оброблятись запити, контролери, які будуть обробляти запити, каталог для роздачі статичних файлів та каталог utils для скриптів, які можуть бути повторно використані та папка middlewares, так як відомо, що в проєкту будуть ролі, тому як мінімум для них потрібно буде створити middleware для перевірки ролі.

Отже, архітектура серверу буде виглядати як на рисунку 3.2.

Також, в каталозі db є файл index.js, який ініціалізує та експортує об’єкт sequelize.

Каталог error має декілька файлів для обробки помилок з http запитів.

Додатково в корені створено декілька файлів:

- consts.js – файл, в якому знаходяться часто використовуванні константи. В цьому випадку в файлі зберігаються enum для рейтингу тренувань.
- .env – файл, в якому зберігаються часто використовуванні константи, але які мають бути приховані від загального доступу. В такий файл часто записують паролі чи токени від API сервісів.

- models.js – файл, в якому реалізовані всі моделі застосунку, та їх зв’язки (Додаток А).
- index.js – файл, який створює об’єкт телеграм боту
- app.js – головний файл, який запускає весь сервер

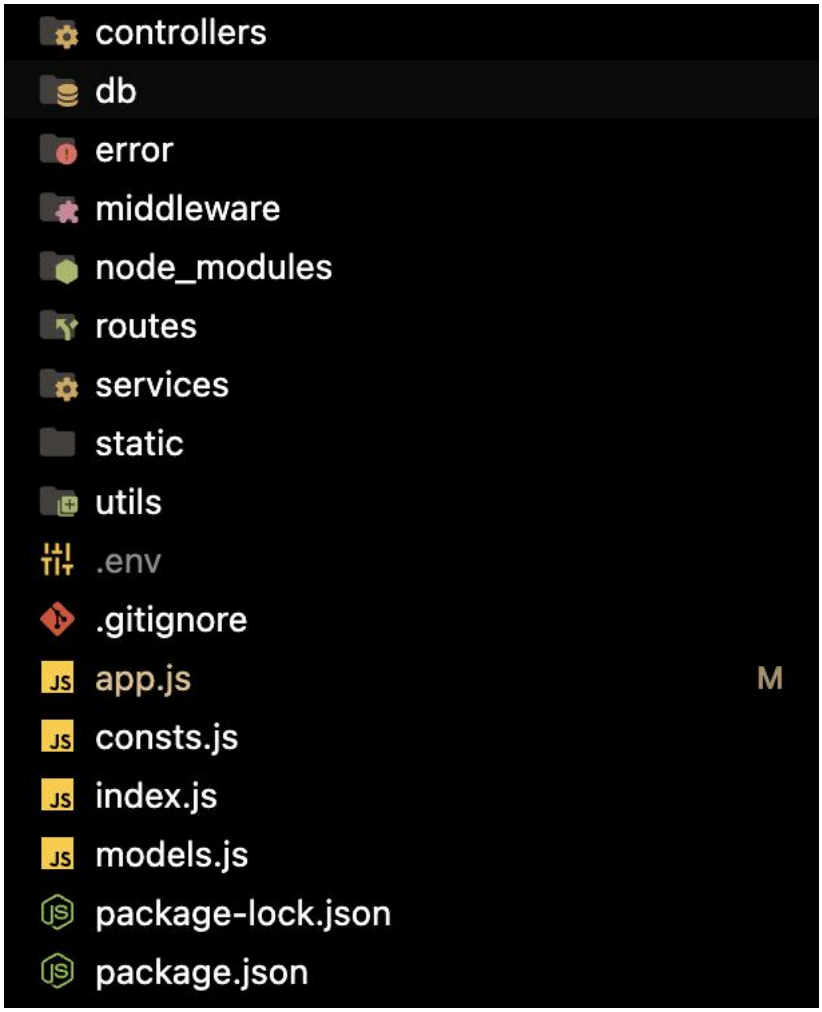


Рисунок 3.2 – Структура/архітектура каталогів проєкту

Для кожної з сутностей було створено її router та controller.

Router це так званий маршрутизатор, який буде виконувати функцію, залежно від вказаного http шляху запиту (рисунок 3.3).

Приклад одного з роутерів наведено на рисунку нижче.

```
const Router = require("express");
const categoryController = require("../controllers/CategoryController");
const checkRoleMiddleware = require("../middleware/checkRoleMiddleware");
const router = new Router();

router.get("/", categoryController.getAll);
router.get("/:id", categoryController.getOne);
router.post("/", checkRoleMiddleware("ADMIN"), categoryController.create);
router.put("/:id", checkRoleMiddleware("ADMIN"), categoryController.edit);
router.delete("/:id", checkRoleMiddleware("ADMIN"), categoryController.remove);

module.exports = router;
```

Рисунок 3.3 – Файл маршрутизації для категорії

В деяких маршрутах можна помітити middleware для перевірки на роль користувача. Перевірка відбувається тільки в разі, коли категорії змінюються, створюються, або видаляються.

Для створення ендпоінту – шляху, за яким буде відбуватись обробка запиту, потрібно створити об'єкт маршрутизатору, вказати тип запиту через так званий “dot notation” та в аргументи першим вказати кінцевий шлях, а другим – функцію колбек, яка буде відпрацьовувати після переходу по цьому маршруту.

Для більшої читабельності прийнято писати функції-обробники в окремих файлах, які називаються “контролерами” (Додаток Б).

На рисунку 3.4 зображено приклад контролеру, який буде відпрацьовувати при отриманні запитів. Кожен контролер отримує декілька аргументів:

- request – об'єкт запиту, який містить всю інформацію про HTTP-запит, включаючи параметри, заголовки, тіло запиту та іншу інформацію.
- response – об'єкт відповіді, який використовується для формування та відправлення відповіді назад клієнту. За допомогою нього можна встановити статус відповіді, заголовки та тіло відповіді.

- next (optional) – функція, яку викликають для передачі управління наступному проміжному обробнику в ланцюзі. Використовується для обробки помилок або продовження виконання інших проміжних обробників.

```
async function create(req, res) {
  try {
    const { name } = req.body;
    const category = await Categories.create({ name });
    res.status(201).json(category);
  } catch (error) {
    console.error("Error creating category:", error);
    res.status(500).json({ error: "Could not create category" });
  }
}

async function edit(req, res) {
  const categoryId = req.params.id;
  const { name } = req.body;

  try {
    const category = await Categories.findByPk(categoryId);
    if (!category) {
      return res.status(404).json({ error: "Category not found" });
    }

    await category.update({ name });
    res.status(200).json(category);
  } catch (error) {
    console.error("Error editing category:", error);
    res.status(500).json({ error: "Could not edit category" });
  }
}
```

Рисунок 3.4 – Приклад контролеру для категорій

3.3 Розробка клієнтської частини

Перші етапи розробки клієнтської частини майже нічим не відрізняються від етапів для серверу, які були вказані вище.

Єдина різниця є в тому, як буде ініціалізовано проєкт. Так як веб-застосунок буде побудований на React, треба створити відповідні залежності та файл ініціалізації. Для більш зручного та швидкого створення відповідних файлів, а також для прискорення та оптимізації процесу збірки проєкту в фінальний набір файлів (файл з розміткою, файл зі стилями та файл зі скриптами) було використано Vite.

Vite - це сучасний інструмент для розробки фронтенд-додатків, створений для забезпечення високої продуктивності та ефективності. Назва "Vite" походить від французького слова "швидко", що відображає головну мету інструменту — забезпечити швидке та оптимізоване розроблення веб-додатків.

Основні особливості Vite:

- Швидкий старт проєкту: Vite використовує ES-модулі, що підтримуються сучасними браузерами, для імпорту та експорту модулів. Це дозволяє уникнути необхідності в попередньому пакетуванні модулів під час розробки, що значно прискорює час запуску сервера розробки.
- Миттєвий HMR (Hot Module Replacement): Vite забезпечує миттєве оновлення модулів під час зміни коду, без необхідності повного перезавантаження сторінки. Це дозволяє розробникам бачити зміни в реальному часі, що значно покращує продуктивність роботи.
- Підтримка TypeScript та інших передпроцесорів: Vite має вбудовану підтримку для TypeScript, JSX, CSS та інших передпроцесорів, що робить його гнучким та зручним інструментом для різних проєктів.
- Ефективне збірка для продакшену: Для продакшен-збірки Vite використовує Rollup, що дозволяє створювати високоефективні та оптимізовані збірки з мінімальним обсягом коду.

- Плагіни: Vite підтримує систему плагінів, що дозволяє легко розширювати функціональність інструменту за допомогою спільноти або власних рішень.

Створити React проєкт за допомогою Vite можна наступною командою (рисунок 3.5):

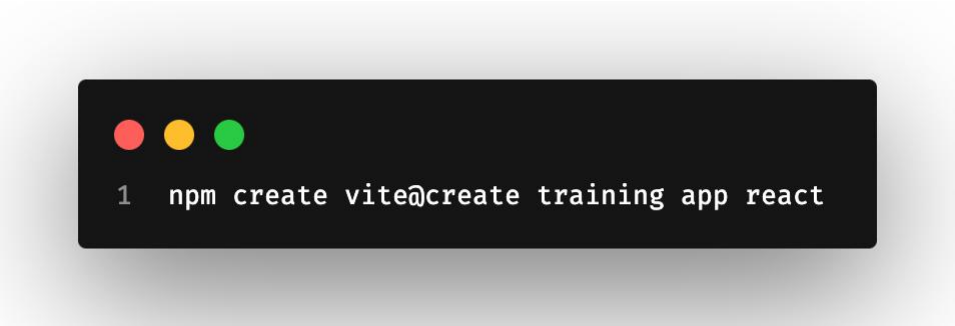


Рисунок 3.5 – Ініціалізація React проєкту за допомогою Vite

Після ініціалізації React проєкту потрібно створити структуру проєкту. В даному прикладі показано стандарту структуру, яка буде використана в ході написання застосунку (рисунок 3.6).

Каталог dist зберігає скомпільовані файли проєкту, готові до завантаження на сервер чи хостинг.

Каталог public – для зберігання всіх статичних файлів.

Каталог src – головний каталог, в якому знаходяться всі файли проєкту.

Він має наступну структуру:

- assets – каталог для зберігання зображень
- components – каталог для зберігання компонентів, так як React використовує компонентний підхід в верстці, тобто багаторазове використання повторюваних UI блоків.
- hooks – каталог, який містить власні/самописні хуки. Хуки – функцій, які дозволяють повторно використовувати логіку стану та інші можливості React у функціональних компонентах. Використання власних хуків покращує організацію коду, підвищує його читабельність та спрощує підтримку.

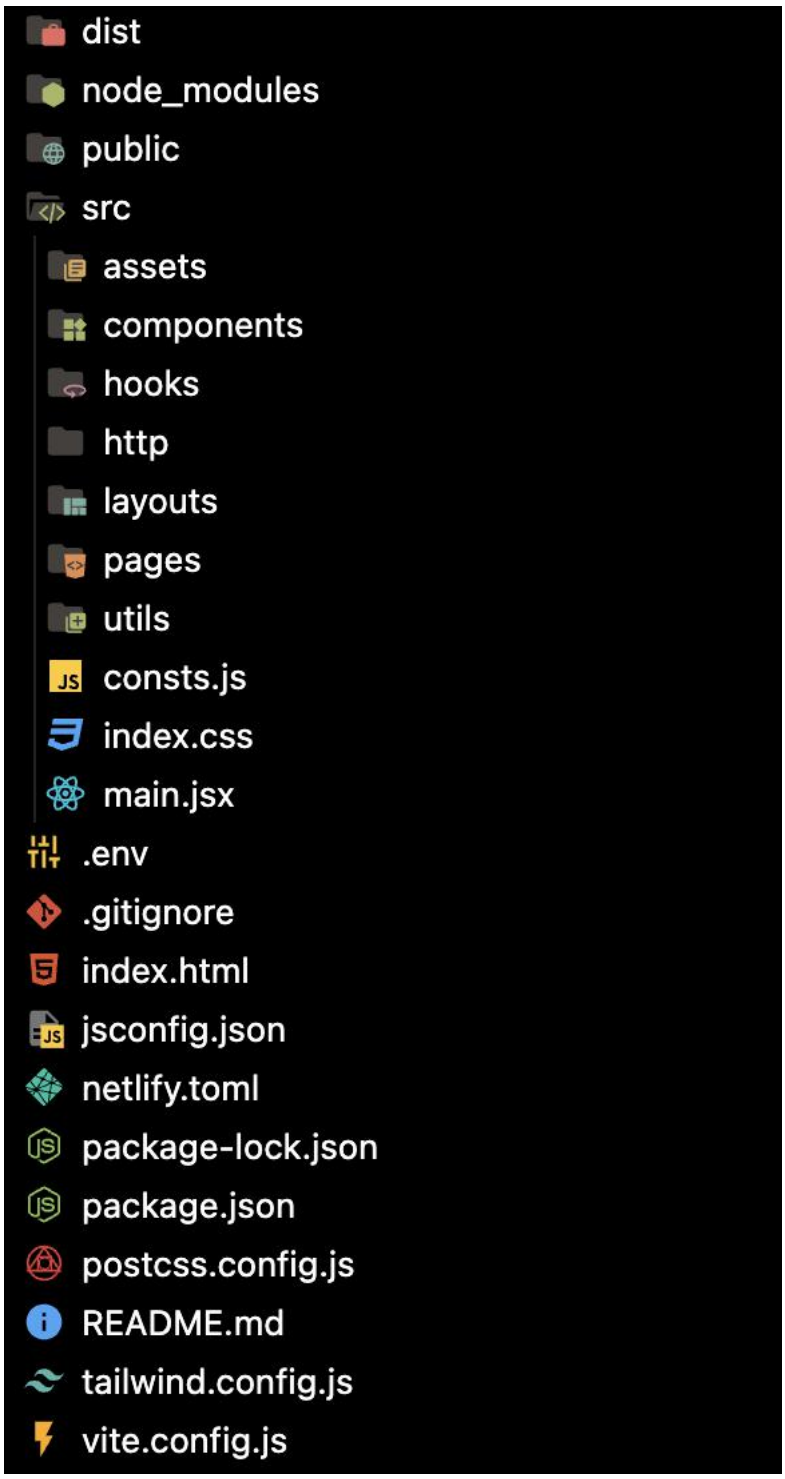


Рисунок 3.6 – Структура проєкту

- http – каталог, в якому знаходяться файли, які відповідають за всі api запити до серверу.
- layouts – каталог, який містить файли макетів сторінок.

- pages – каталог, в якому зберігаються файли-компоненти, які будуть відображені після переходу на індивідуальне посилання.
- utils для допоміжних файлів з кодом, який повторюється

Також варто згадати про деякі важливі файли проєкту, такі як:

- vite.config.js – файл конфігурації vite. В ньому прописані всі налаштування для збірника – скрипту, який з усіх вхідних файлів створить один js файл, який буде використовуватись у продакшені. Приклад такого файлу зображено на рисунку 3.7.

```
export default defineConfig({
  plugins: [react()], //splitVendorChunkPlugin(), visualizer()
  esbuild: {
    loader: "jsx",
  },
  optimizeDeps: {
    esbuildOptions: {
      loader: {
        ".js": "jsx",
      },
    },
  },
  resolve: {
    alias: {
      "@components": path.resolve(__dirname, "src/components"),
      "@pages": path.resolve(__dirname, "src/pages"),
      "@utils": path.resolve(__dirname, "src/utils"),
      "@hooks": path.resolve(__dirname, "src/hooks"),
      "@http": path.resolve(__dirname, "src/http"),
      "@consts": path.resolve(__dirname, "src/consts.js"),
    },
  },
```

Рисунок 3.7 – Файл конфігурації Vite

- consts.js – файл з константами. В ньому зберігаються деякі enum проєкту, маршрути навігації та унікальні кольори.

Для написання React компонентів було використано підхід ФП (функціональне програмування).

Раніше React компоненти декларувалися в ООР (об'єкто-орієнтоване програмування) стилі, але згодом з'явилась підтримка ФП підходу, яка стала головним стилем написання компонентів. Перехід був необхідний з декількох причин:

1. Покращена Оптимізація: Функціональні компоненти краще оптимізуються сучасними інструментами розробки та трансляторами, такими як Babel. Крім того, функціональні компоненти легше аналізувати для виявлення та оптимізації чистих функцій.

2. Краща Композиція: Функціональні компоненти легко комбінуються та повторно використовуються за допомогою хуків та інших інструментів. Це сприяє створенню більш модульного та масштабованого коду.

3. Консистентність та Прогнозованість: компоненти забезпечують більш консистентну та прогнозовану поведінку. Вони схильні до меншої кількості побічних ефектів, що покращує стабільність програми та полегшує налагодження.

4. Підтримка Спільноти та Майбутні Оновлення: React активно розвивається у напрямку функціонального програмування. Спільнота активно підтримує та використовує функціональні компоненти, що сприяє швидкому впровадженню нових можливостей та покращень.

Приклад з написання подібного компоненту наведено в додатку В.

На прикладі видно чітку і просту логіку побудови UI компонента. В самому початку відбувається імпорт всіх необхідних залежностей (рисунок 3.8).

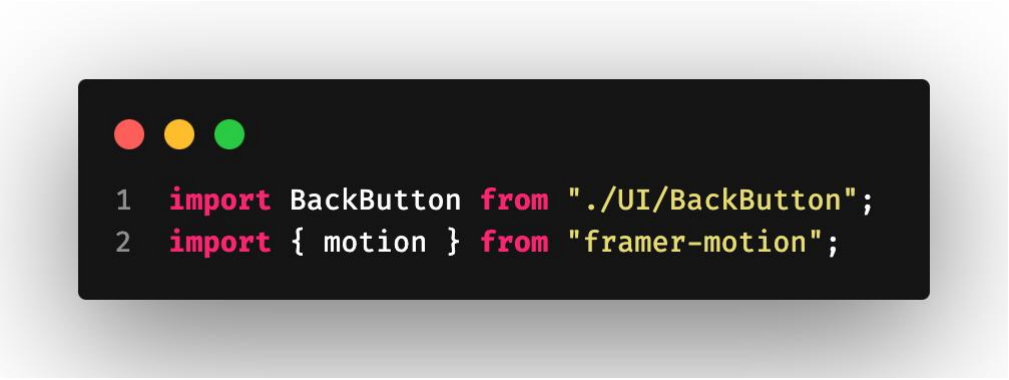


Рисунок 3.8 – Імпорт необхідних залежностей компоненту “PageHeader”

Далі відбувається ініціалізація всіх статичних даних компоненту.

Для відображення динамічних даних React застосунку необхідно “перемальовувати” поточний елемент всередині DOM. Для пришвидшення таких операцій він має Virtual DOM, всередині якого відбуваються стеження за динамічними елементами, маніпуляція та рендер вже в сам DOM документу сторінки. Тобто при коректній кожній зміні будь-якого значення всередині компоненту React виконує його повний ре-рендер. Для того, щоб не перевираховувати значення, які не змінюються, прийнято записувати їх перед декларацією компоненту, або всередині компоненту за допомогою відповідних хуків – useMemo, useCallback та memo.

В даному випадку винесений об’єкт “sizes” (рисунок 3.9), який зберігає необхідну розмітку для заголовку, відповідно до вказаного розміру.

Далі відбувається ініціалізація самого компоненту – PageHeader (рисунок 3.10). Кожен компонент приймає один об’єкт, який містить всі аргументи передані до функції. Його прийнято називати props (properties). Пропси – це аргументи, які передаються в кожен компонент як атрибути HTML тегу. На зображені всі атрибути відбираються від загального об’єкту props за допомогою можливості/властивості мови JavaScript – деструктуризація, яка дає змогу “вибрати” всі елементи з об’єкту.

В тілі функції PageHeader вказується вся логіка – renderTitle та headerAnim. renderTitle необхідна для виведення коректної розмітки, яка

залежить від вказаного розміру. headerAnim – об’єкт, який збегірає параметри для анімації компоненту.

Для відображення HTML розмітки, React компонент повинен “повертати” її всередині функції.

За допомогою JSX є можливість записувати HTML теги всередині JavaScript, що робить написання компонентів більш зручнішим та візуально зрозумілим.

Рисунок 3.9 – Об’єкт “sizes”, задекларований до компоненту, щоб запобігти повторного перерахунку після ре-рендеру сторінки

Рисунок 3.10 – Ініціалізація компоненту PageHeader

4. РОЗГОРТАННЯ ЗАСТОСУНКУ

Для того, щоб користувачі мали доступ до розробленого застосунку, необхідно розмістити його в мережі Інтернет для загального доступу. Існує кілька варіантів розміщення застосунку:

1. Розміщення додатку локально з використанням "сірої" IP-адреси.

У цьому випадку потрібно буде замовити "сіру" IP-адресу у провайдера та налаштувати застосунок для роботи з цією адресою. Однак, оскільки веб-клієнт буде запущений всередині Telegram, необхідно отримати SSL-сертифікат для доступу до файлів за HTTPS-адресою. Оскільки клієнтська частина буде мати SSL-сертифікат, також потрібно буде додати його до серверної частини застосунку, оскільки цього вимагає політика CORS.

Отримання SSL-сертифікату включає кілька нескладних кроків, але основною вимогою є наявність доменного імені. Тому, для локального розміщення застосунку, окрім статичної IP-адреси, потрібно буде придбати доменне ім'я. Якщо всі ці кроки виконано, застосунок успішно працюватиме онлайн. Залишається лише впевнитись, що локальний сервер працюватиме цілодобово, щоб забезпечити постійну доступність даних.

Якщо всі кроки буде виконано, то застосунок буде успішно працювати онлайн. Все що залишиться – впевнитись в тому, що локальний сервер буде працювати цілодобово, щоб забезпечити доступність даних.

Основним недоліком такого підходу є складність забезпечення цілодобового електропостачання в домашніх умовах, що може бути серйозною проблемою, особливо в сучасних реаліях

Звідси й випливає основний мінус, так як в сьогоднішніх умовах, та й в стандартних, домашніх умовах забезпечити цілодобове електроспоживання являється досить важкою задачею.

2. Розміщення на сторонніх сервісах

Розміщення на сторонніх сервісах має кілька суттєвих переваг, які роблять його вигідним вибором для багатьох розробників і компаній.

Основні причини вибору сторонніх сервісів для розміщення своїх проєктів:

1. Зручність та простота налаштування. Сторонні сервіси надають інтуїтивні інтерфейси та інструменти, які спрощують процес розгортання та управління додатками. Більшість платформ пропонують автоматичне розгортання, інтеграцію з системами контролю версій (GitHub, GitLab, Bitbucket) та попередньо налаштовані середовища, що дозволяє швидко розпочати роботу без необхідності глибоких технічних знань.

2. Економія часу та ресурсів. Замість того, щоб налаштовувати і обслуговувати власну інфраструктуру, розробники можуть зосередитись на написанні коду та вдосконаленні функціоналу свого застосунку. Сторонні сервіси беруть на себе більшість завдань, пов'язаних з обслуговуванням серверів, оновленням програмного забезпечення, резервним копіюванням та безпекою.

3. Масштабованість. Сторонні хостинг-платформи надають можливості для легкого масштабування ресурсів відповідно до потреб додатка.

4. Вбудована безпека. Більшість сторонніх сервісів забезпечують високий рівень безпеки, включаючи SSL-сертифікати, захист від DDoS-атак, регулярні оновлення безпеки та можливості для резервного копіювання даних. Це значно зменшує ризики, пов'язані з безпекою, порівняно з самостійним управлінням інфраструктурою.

5. Надійність та доступність. Сторонні сервіси зазвичай мають високу доступність і надійність завдяки розподіленій інфраструктурі та потужним дата-центрам по всьому світу. Це дозволяє забезпечити безперебійну роботу застосунку навіть у разі непередбачуваних обставин.

6. Вартість. Використання сторонніх сервісів часто є більш економічно вигідним, ніж розгортання та обслуговування власної інфраструктури. Оплата йде тільки за використані ресурси, що дозволяє уникнути зайвих витрат на обладнання та обслуговування.

4.1 Розгортання клієнтської частини

Для розміщення клієнта було вирішено скористатися хостингом Netlify. Netlify — це платформа для розміщення веб-додатків, яка надає прості та зручні інструменти для розробників. Вона підтримує автоматичне розгортання з Git-репозиторіїв, має вбудовану систему безпеки, включаючи безкоштовні SSL-сертифікати, та пропонує безліч додаткових функцій для масштабування та оптимізації додатку.

Кроки для налаштування хостингу на Netlify:

- Підключення репозиторію: Необхідно відкрити Netlify та обрати опцію для створення нового сайту.
- Підключення Git-репозиторію (GitHub, GitLab або Bitbucket).
- Налаштування параметрів розгортання: Необхідно вказати команду для побудови проєкту (наприклад, npm run build). Далі потрібно вказати директорію, де розміщені збірки додатку (наприклад, build або dist).
- Запуск розгортання: Після налаштування параметрів потрібно натиснути кнопку для запуску розгортання. Netlify автоматично збудує та розгорне додаток.
- Отримання домену: Після успішного розгортання додаток буде доступний за автоматично згенерованою URL-адресою. Налаштувати власний домен можливо через панель управління Netlify.
- Налаштування SSL-сертифіката: Netlify автоматично надає безкоштовний SSL-сертифікат для сайту.

Додатково до кроків, описаних вище, також необхідно створити конфігураційний файл netlify.toml (рисунок 4.1), який вкаже Netlify, що кожен запит до сторінки необхідно переадресовувати на адресу index.html, так як побудований додаток являється SPA застосунком і має лише одну динамічну сторінку.

Рисунок 4.1 – Файл конфігурації netlify.toml

Перевагами використання Netlify є простота у використанні, автоматичне розгортання, безкоштовні SSL-сертифікати та захист від DDoS атак.

4.2 Розгортання серверної частини

Сервер було розміщено на власному сервері сервісу Digital Ocean.

DigitalOcean — це хмарний інфраструктурний провайдер, який надає прості та доступні рішення для хостингу додатків та серверів. DigitalOcean пропонує різноманітні інструменти для розгортання, масштабування та управління вашими додатками. Вони включають віртуальні сервери (Droplets), керовані бази даних, Kubernetes, об'єктне сховище та багато іншого.

Етапи розміщення серверної чатини коду на сервері:

1. Підключення до серверу.

Для підключення до серверу за допомогою ssh необхідно в консоль ввести наступну команду: “ssh <ім’я користувача>@<адреса серверу>”

Далі сервер запитає пароль, та, після успішного введення – допустить до домашньої директорії.

2. Створення каталогу проєкту.

Для створення каталогу необхідно в обраному каталозі ввести команду “mkdir <назва каталогу>”.

3. Клонування git-репозиторію до створеного каталогу

Для клонування репозиторію всередину створеного каталогу необхідно прописати команду “git clone <адреса репозиторію> /<назва каталогу проєкту>”.

4. Запуск проєкту в фоновому режимі.

Для того, щоб створений застосунок працював навіть після виходу з серверу необхідно прописати перейти в корінь проєкту та прописати команду “nohup <команда запуску> &”. Знак “&” виведе в консоль номер запущеного процесу. Щоб в подальшому мати змогу швидко та легко вимкнути сервер необхідно прописати команду “kill <номер процесу>”.

Для автоматизації процесу оновлення серверу після внесення змін до репозиторію необхідно було зробити декілька кроків:

1. В корені проєкту створити папку .github, всередині якої розмістити ще один каталог – workflows.

2. В створеному каталозі необхідно створити файл .yml з кодом, який вказує GitHub що потрібно робити після внесення змін в репозиторій. Приклад такого файлу наведено на рисунку 4.2.

Увагу слід приділити полю “script” – саме в ньому вказується, що потрібно виконати після оновлення репозиторію. Для того щоб підключитися до серверу до серверу, до GitHub secrets (аналог dotenv) було додано декілька полей з необхідною інформацією, такою як пароль, адреса, порт і тд.

```yaml
name: Update Server

on:
  push:
    branches:
      - main

jobs:
  update-server:
    runs-on: ubuntu-latest

    steps:
    - name: update fitness app
      uses: appleboy/ssh-action@v1.0.0
      with:
        host: ${{ secrets.HOST }}
        username: ${{ secrets.USERNAME }}
        password: ${{ secrets.PASSWORD }}
        port: ${{ secrets.PORT }}
        script: |
          cd ~/artem/TrainingAppServer
          git pull
          supervisorctl restart fitness_app
```

Рисунок 4.2 – Файл GitHub Actions

Висновки

Метою даної роботи було створення інформаційної системи для надання онлайн послуг з фітнесу.

В процесі виконання проєкту було ретельно досліджено предметну область, що дозволило сформулювати чітку постановку задачі, яка передбачала розробку інформаційної системи у вигляді Telegram-бота. Було розроблено базу даних, API сервер та прогресивний веб-додаток (PWA) з використанням найпопулярніших та найефективніших технологій. Додаток було розгорнуто на сервісах Netlify та Digital Ocean.

В результаті було реалізовано та успішно перевірено більшу частину функціоналу бота, що забезпечує надання онлайн послуг з фітнесу.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Тенденції розробки мобільних додатків [Електронний ресурс] // https://careers.easternpeak.com/blog/mobile-app-development-trends/

2. Modern веб Application Architecture Explained: Components, Best Practices and More [Електронний ресурс] // https://litslink.com/blog/веб-application-architecture#title3

3. What Is a Front-End Developer? [Електронний ресурс] // https://frontendmasters.com/guides/front-end-handbook/2018/what-is-a-FD.html

4. Javascript [Електронний ресурс] // https://developer.mozilla.org/ru/docs/веб/JavaScript

5. Official site Flutter [Електронний ресурс] // https://flutter.dev/

6. Official site React [Електронний ресурс] // https://uk.reactjs.org/

7. What is React Native? [Електронний ресурс] // https://www.netguru.com/glossary/react-native.

8. What is Backend Developer? [Електронний ресурс] // https://www.guru99.com/what-is-backend-developer.html

9. About Node.js [Електронний ресурс] // https://nodejs.org/en/about/

10. Документація Ant Design [Електронний ресурс] // https://ant.design/components/overview/

11. Документація Tailwind CSS [Електронний ресурс] // https://tailwindcss.com/docs/installation

12. Документація React Query [Електронний ресурс] // https://tanstack.com/query/latest/docs/framework/react/overview

13. Документація Telegram Mini Apps [Електронний ресурс] // https://core.telegram.org/bots/вебapps#initializing-mini-apps

14. Документація Framer Motion [Електронний ресурс] // https://www.framer.com/motion/

Додаток А. Файл моделей sequelize – models.js

```
const sequelize = require("./db/index")
const { DataTypes } = require("sequelize");

const User = sequelize.define("user", {
id: {
type: DataTypes.INTEGER,
primaryKey: true,
unique: true,
autoIncrement: true,
},
chatID: { type: DataTypes.INTEGER, unique: true },
username: { type: DataTypes.STRING, allowNull: true },
gender: { type: DataTypes.SMALLINT, allowNull: true },
role: { type: DataTypes.STRING, defaultValue: "USER" },
});

const Exercise = sequelize.define("exercise", {
id: {
type: DataTypes.INTEGER,
primaryKey: true,
unique: true,
autoIncrement: true,
},
title: { type: DataTypes.STRING, allowNull: false },
content: { type: DataTypes.TEXT, allowNull: true },
video: { type: DataTypes.STRING, allowNull: false },
});

const Categories = sequelize.define("categories", {
id: {
type: DataTypes.INTEGER,
primaryKey: true,
unique: true,
```

```
autoIncrement: true,
},
name: { type: DataTypes.STRING, allowNull: false, unique: true },
});

const Types = sequelize.define("types", {
id: {
type: DataTypes.INTEGER,
primaryKey: true,
unique: true,
autoIncrement: true,
},
name: { type: DataTypes.STRING, allowNull: false, unique: true },
});

const ExerciseCategories = sequelize.define(
"exercise_categories",
{},
{ timestamps: false }
);

const ExerciseTypes = sequelize.define(
"exercise_types",
{},
{ timestamps: false }
);

const Reviews = sequelize.define("reviews", {
id: {
type: DataTypes.INTEGER,
primaryKey: true,
unique: true,
autoIncrement: true,
},
rating: { type: DataTypes.FLOAT, allowNull: true },
```

```
review: { type: DataTypes.TEXT, allowNull: true },
});

const Training = sequelize.define("training", {
id: {
type: DataTypes.INTEGER,
primaryKey: true,
unique: true,
autoIncrement: true,
},
level: { type: DataTypes.INTEGER, allowNull: false },
gender: { type: DataTypes.SMALLINT, allowNull: false },
title: { type: DataTypes.STRING, allowNull: false },
content: { type: DataTypes.TEXT, allowNull: false },
image: { type: DataTypes.STRING, allowNull: false },
exec_time: { type: DataTypes.TIME, allowNull: false },
rating: { type: DataTypes.FLOAT, allowNull: true },
});

const TrainingExercise = sequelize.define(
"training_exercise",
{ ordinal_number: { type: DataTypes.INTEGER, allowNull: true } },
{ timestamps: false }
);

User.hasMany(Reviews);

Training.hasMany(Reviews);

Training.belongsToMany(Exercise, { through: TrainingExercise });
Exercise.belongsToMany(Training, { through: TrainingExercise });

Reviews.belongsTo(User, { foreignKey: "userId" });
Reviews.belongsTo(Training, { foreignKey: "trainingId" });
```

```
Exercise.belongsToMany(Categories, { through: ExerciseCategories });
Categories.belongsToMany(Exercise, { through: ExerciseCategories });
Exercise.belongsToMany(Types, { through: ExerciseTypes });
Types.belongsToMany(Exercise, { through: ExerciseTypes });

module.exports = {
User,
Exercise,
Types,
Categories,
ExerciseCategories,
ExerciseTypes,
Training,
TrainingExercise,
Reviews,
};
```

Додаток Б. Приклад контролеру на основі CategoryController.js.

```
const { Categories } = require("../models");
const { Op } = require("sequelize");

async function create(req, res) {
try {
const { name } = req.body;
const category = await Categories.create({ name });
res.status(201).json(category);
} catch (error) {
console.error("Error creating category:", error);
res.status(500).json({ error: "Could not create category" });
}
}

async function edit(req, res) {
const categoryId = req.params.id;
const { name } = req.body;

try {
const category = await Categories.findByPk(categoryId);
if (!category) {
return res.status(404).json({ error: "Category not found" });
}

await category.update({ name });
res.status(200).json(category);
} catch (error) {
console.error("Error editing category:", error);
res.status(500).json({ error: "Could not edit category" });
}
}

async function getAll(req, res) {
```

```
const {
page = 1,
limit = 10,
sortBy = "id",
sortOrder = "ASC",
filterByName = "",
} = req.query;
const offset = (page - 1) * limit;

try {
const whereClause = {
name: { [Op.like]: `%${filterByName}%` },
};

const orderClause = [[sortBy, sortOrder]];

const categories = await Categories.findAndCountAll({
where: whereClause,
order: orderClause,
limit: parseInt(limit, 10),
offset: parseInt(offset, 10),
});

const totalPages = Math.ceil(categories.count / limit);

return res.status(200).json({
totalPages,
currentPage: parseInt(page, 10),
totalCount: categories.count,
categories: categories.rows,
});
} catch (error) {
console.error("Error getting categories:", error);
res.status(500).json({ error: "Could not fetch categories" });
}
```

```
}

async function getOne(req, res) {
const categoryId = req.params.id;
try {
const category = await Categories.findByPk(categoryId);
if (!category) {
return res.status(404).json({ error: "Category not found" });
}
res.status(200).json(category);
} catch (error) {
console.error("Error getting category by id:", error);
res.status(500).json({ error: "Could not fetch category" });
}
}

async function remove(req, res) {
const categoryId = req.params.id;
try {
const category = await Categories.findByPk(categoryId);
if (!category) {
return res.status(404).json({ error: "Category not found" });
}

await category.destroy();
res.status(204).end();
} catch (error) {
console.error("Error removing category:", error);
res.status(500).json({ error: "Could not remove category" });
}
}

module.exports = {
create,
edit,
```

```
getAll,
getOne,
remove,
};
```

Додаток В. Приклад React компоненту на основі PageHeader

```
import BackButton from "./UI/BackButton";
import { motion } from "framer-motion";

const sizes = {
LARGE: (title) => (
<h1 className="text-gray-100 font-bold text-3xl drop-shadow-xl">{title}</h1>
),
MEDIUM: (title) => (
<h2 className="text-gray-100 font-bold text-2xl drop-shadow-xl">{title}</h2>
),
SMALL: (title) => (
<h3 className="text-gray-100 font-bold text-xl drop-shadow-xl">{title}</h3>
),
};

export const PageHeader = ({
size,
title,
extra = null,
noBackBtn = false,
customBack = null,
}) => {
const renderTitle = () => {
const sizeKey = size?.toUpperCase();

if (sizeKey && sizes[sizeKey]) return sizes[sizeKey](title);
return sizes.MEDIUM(title);
};

const headerAnim = {
visible: {
y: 0,
opacity: 1,
```

```
transition: {
stiffness: 100,
damping: 15,
},
},
hidden: {
y: -50,
opacity: 0,
},
};

return (
<motion.div className="relative h-20" variants={headerAnim} initial="hidden" animate="visible">
<div
className="grid fixed grid-cols-3 w-full h-20 items-center gap-4 shadow-sm"
style={{
background: "linear-gradient(to bottom, #ffc04b, #ffba43)",
zIndex: 999,
gridTemplateColumns: ".2fr 1fr .2fr",
}}
>
{!noBackBtn && (
<div className="flex items-center justify-center">
{customBack ? (
customBack
) : (
<BackButton
withIcon
withText={false}
withBG={false}
withPadding={false}
/>
)}
</div>
```

```
)}

<div className="text-center">{renderTitle()}</div>

{extra && (
<div className="flex items-center justify-center drop-shadow-md pr-4">
{extra}
</div>
)}
</div>
</motion.div>
);
};
```