МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Економічний факультет
Кафедра економіко-математичного моделювання та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА/ПРОЄКТ
на здобуття освітнього ступеня бакалавра

на тему: **«РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ ВЕБ-ЗАСТОСУНКУ ДЛЯ ДОПОМОГИ У ФОРМУВАННІ ТА ПОЗБУТТІ ЗВИЧОК З ЕЛЕМЕНТАМИ ГЕЙМІФІКАЦІЇ»**

Виконала: студентка 4 курсу, групи КН-41 першого (бакалаврського) рівня вищої освіти спеціальності 122 Комп'ютерні науки освітньо-професійної програми «Комп'ютерні науки»
Корольчук Анна Миколаївна

Керівник: викладач кафедри ЕММІТ,
Красюк Богдан Віталійович

Рецензент: front-end developer "DOODLE", LLC
Місай Володимир Віталійович

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ
Завідувач кафедри економіко-математичного моделювання та інформаційних технологій _____ (проф., д.е.н. Кривицька О.Р.)
Протокол № 11 від « 18 » травня 2023 р.

Острог, 2023

Міністерство освіти і науки України
Національний університет «Острозька академія»

Факультет: *економічний*
Кафедра: *економіко-математичного моделювання та інформаційних технологій*
Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри економіко-математичного моделювання
та інформаційних технологій

_____ Ольга КРИВИЦЬКА
«____» _____20__ р.

З А В Д А Н Н Я
на кваліфікаційну роботу/проєкт студента

Корольчук Анни Миколаївни

*1. Тема роботи: "*Розробка клієнтської частини веб-застосунку для допомоги у формуванні та позбутті звичок з елементами гейміфікації".

керівник проєкту Красюк Богдан Віталійович, викладач кафедри економіко-математичного моделювання та інформаційних технологій.

Затверджено наказом ректора НаУОА від 31 жовтня 2022 року №77.

2. Термін здачі студентом закінченої роботи/проєкту: 31 травня 2023 року.

3. Вихідні дані до роботи/проєкту: дана робота полягає у розробці клієнтської частини веб-застосунку для допомоги у формуванні та позбутті звичок з елементами гейміфікації за допомогою ReactJS, Swagger, Figma, Material UI, Github, Visual Studio Code, Rider.

4. Перелік завдань, які належить виконати: реалізувати функції створення, перегляду, редагування та видалення звичок, проходження завдань на щодень з елементами гейміфікації та функцією відслідковування прогресу.

5. Перелік графічного матеріалу: рисунки, таблиці.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Красюк Б.В.	01.12.2022р.	01.12.2022р.
2	Красюк Б.В.	01.12.2022р.	01.12.2022р.
3	Красюк Б.В.	01.12.2022р.	01.12.2022р.

7. Дата видачі завдання: 01.12.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1.	Затвердження теми роботи/проєкту	до 31.10.2022 р.	
2.	Постановка технічного завдання	до 01.12.2022 р.	
3.	Розробка архітектури проєкту, погодження функціоналу	до 15.01.2023 р.	
4.	Розробка макетів дизайну проєкту	до 01.02.2023 р.	
5.	Розробка функціоналу для створення, редагування, перегляду та видалення звичок	до 15.03.2023 р.	
6.	Розробка функціоналу для виконання щоденних завдань	до 15.04.2023 р.	
7.	Розробка функціоналу для відслідковування прогресу та гейміфікації	до 01.05.2023 р.	
8.	Тестування системи	до 20.05.2023 р.	
9.	Попередній захист кваліфікаційної роботи/проєкту	до 31.05.2023 р.	
10.	Здача кваліфікаційної роботи/проєкту на кафедру	31.05.2023 р.	

Студент: _____ Анна КОРОЛЬЧУК

Керівник кваліфікаційної роботи: _____ Богдан КРАСЮК

АНОТАЦІЯ
кваліфікаційної роботи/проєкту
на здобуття освітнього ступеня бакалавра

Тема: *"Розробка клієнтської частини веб-застосунку для допомоги у формуванні та позбутті звичок з елементами гейміфікації"*

Автор: *Корольчук Анна Миколаївна*

Науковий керівник: *Красюк Богдан Віталійович, викладач кафедри економіко-математичного моделювання та інформаційних технологій*

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи: *61 с., 25 рис., 2 табл., 13 джерел.*

Ключові слова: *трекер, звичка, веб-застосунок, клієнтська частина, гейміфікація.*

Короткий зміст праці:

Метою кваліфікаційної роботи/проєкту була розробка алгоритму для трекеру формування та позбуття звичок "Habit Rabbit" і реалізація його у вигляді програмного забезпечення. Цей проєкт реалізує собою можливість створення, перегляду, редагування та видалення звичок, проходження завдань на щодень з елементами гейміфікації та функцією відслідковування прогресу. Користувачами додатку є особи, які мають на меті зробити своє життя кращим, шляхом формування чи позбуття звичок. Для створення клієнтської частини проекту було використано середовище розробки Visual Studio Code. Платформою для проекту стала бібліотека React.JS. Під час розробки для взаємодії із серверною частиною та API було використано середовище Rider та набір інструметів Swagger. Для розробки макетів та імплементації дизайну використано графічний редактор Figma та бібліотеку Material UI. Система контролю версій - Github.

The purpose of the qualification work/project was to develop an algorithm for the Habit Rabbit habit tracker and implement it in the form of software. This project provides the ability to create, view, edit, and delete habits, complete daily tasks with gamification elements, and track progress. Users of the app are people who want to make their lives better by forming or breaking habits. The Visual Studio Code development environment was used to create the client side of the project. React.JS library became the platform for the project. During development, the Rider environment and the Swagger toolkit were used to interact with the server side and API. The Figma graphic editor and the Material UI library were used to develop layouts and implement the design. The version control system is Github.

_____

ЗМІСТ

ВСТУП

В умовах сьогодення людство стикнулося не лише з періодом автоматизації, а й із мейнстрімом “зроби себе сам”. App Store чи Play Market переповнені від різноманіття застосунків, які мотивують користувача до удосконалення.

Актуальність теми кваліфікаційного проєкту є вагомою, адже ми живемо у період, коли кожна людина прагне саморозвитку та вдосконалення своїх навичок. Ідея створення системи для менеджменту звичок відповідає потребам сучасного суспільства, адже ця система допомагає формувати корисні звички та позбутися шкідливих. Програмний продукт розроблений з метою спрощення цього процесу шляхом контролю та відстеження прогресу за допомогою гейміфікації. За допомогою цього програмного забезпечення користувачі можуть встановлювати конкретні цілі, відстежувати свій прогрес та отримувати нагороду за свої досягнення.

Така система може стати незамінним помічником в досягненні власних цілей та покращенні життя кожної людини. Реалізація системи управління звичками може мати значний вплив на продуктивність та благополуччя людей, бізнесу та організацій.

Мета кваліфікаційної роботи/проєкту полягає у розробці алгоритму для трекеру набування та позбуття звичок “Habit Rabbit” і реалізація його у вигляді програмного забезпечення.

Під час виконання роботи було поставлено такі **завдання**:

1. Опис загальних положень;
2. Огляд інформаційного та математичного забезпечення;
3. Опис програмного та технічного забезпечення;
4. Реалізація клієнтської частини.

Об’єкт дослідження: дослідження способів моніторингу процесу набуття/позбавлення звичок.

Предмет дослідження: розробка інформаційної системи для автоматизації моніторингу прогресу у набутті чи позбавленні звичок.

РОЗДІЛ 1. ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1. Опис предметного середовища

Існує багато різних технологій для написання програмного забезпечення, які зазвичай використовуються залежно від типу проекту, мов програмування, розміру та складності проекту, вимог до продукту та інших факторів. Вибір може залежати від існуючих знань розробників, їхнього досвіду та попередніх проектів. Для великих, складних проектів можуть використовуватися різні технології та мови програмування для різних компонентів проекту. Багато розробників використовують фреймворки, такі як React, Angular та Vue.js, які допомагають прискорити розробку та полегшують тестування. Незалежно від вибору, важливо використовувати кращі практики програмування та забезпечення якості, щоб забезпечити ефективність.

Система «Habit Rabbit» представлена як односторінковий застосунок (SPA). Загалом, застосунки поділяються на SPA та PWA. Для чіткого розуміння дамо визначення обом поняттям та вкажемо на переваги обраного виду.

PWA (Progressive Web App) – багатосторінковий веб-застосунок. Під час використання такого ресурсу здійснюється завантаження кожної із сторінок сервера окремо.

SPA (Single Page Application) – це веб-застосунок, який працює в одній вкладці браузера і не вимагає перезавантаження сторінки під час взаємодії з користувачем. Зазвичай такі застосунки будуються з використанням фреймворків або бібліотек, таких як Angular, React або Vue.js (у даному випадку – React).

Переваги односторінкових застосунків:

1. Швидкість: під час перемикання вкладок SPA завантажується лише окремі елементи вмісту. Фрейм сторінки завантажується один раз.
2. Безперервний UX: робота в SPA схожа на використання native app (нативний застосунок), а не веб-сайту, завдяки миттєвому перемиканню сторінок.
3. Кросплатформеність: можливість роботи з будь-яким пристроєм та операційною системою.

4. Автономний режим: використовується кешування для роботи без підключення до Інтернету.
5. Підтримка: легше контролювати та виправляти баги.
6. Оновлення: SPA можуть оновлюватися безпосередньо розробниками [1].

1.2. Стек технологій

Після врахування усіх факторів було прийнято рішення скористатися таким стеком технологій:

Архітектура рішення — Clean Architecture. Чиста архітектура — це кращий спосіб організувати програми помірної та високої складності. Вона допомагає нам відокремлювати залежності і тримати їх ізольованими від бізнес-логіки і доменної моделі застосунку (рис. 1.1).

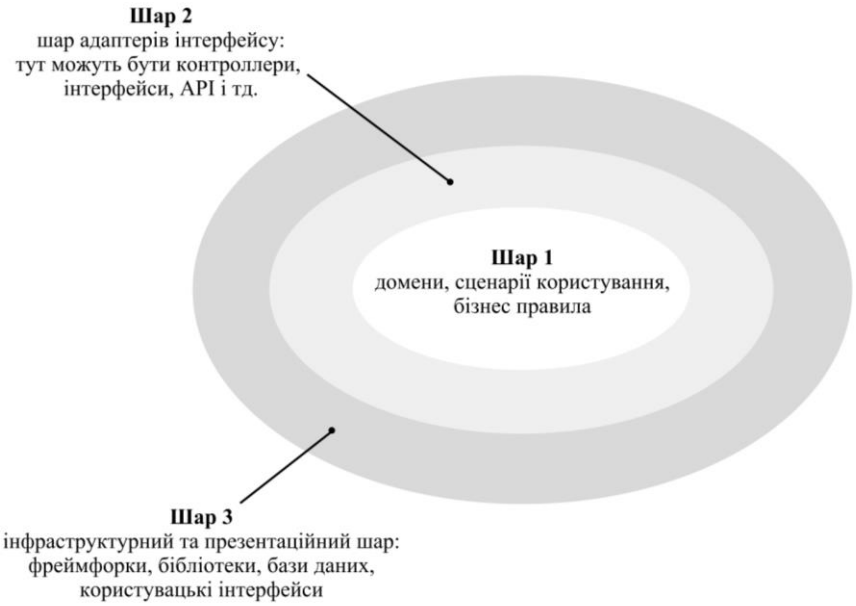

Рис. 1.1. Зображення ідеї чистої архітектури

Основна ідея «чистої архітектури» полягає в тому, щоб зробити рішення адаптивним, зберегти варіанти використання логіки програми незалежно від інтерфейсу та зовнішніх фреймворків. Підсумовуючи, ми можемо окреслити наступні результати чистої архітектури:

1. Незалежна від інтерфейсу користувача: використовуючи чисту архітектуру, ми повинні мати можливість легко змінювати інтерфейс або рівні презентації, не змінюючи рівень програми тощо. Інтерфейс користувача може бути з будь-якої зовнішньої структури або інтерфейсу користувача консолі, будь-якої мережі і його можна замінити без зміни інших рівнів або решти системи.

2. Незалежна від бази даних: архітектура має бути достатньо гнучкою, щоб міняти базу даних, не впливаючи на варіанти використання програми та сутності. Рішення може перемикати набір даних на MS SQL, MySQL, Oracle, MongoDB або щось інше.

3. Незалежна від зовнішніх бібліотек: бізнес-правила мають бути незалежними від зовнішніх сторін.

4. Незалежна від фреймворку: основні правила застосування повинні бути незалежними від існування фреймворків, бібліотек на майбутнє. Ми можемо включити фреймворки, але як інструменти.

5. Тестується: архітектура повинна відповідати тестуванню основної програми та бізнес-випадкам і правилам без інтерфейсу користувача, бази даних, веб-сервера чи будь-якого зовнішнього компонента.

Принципи чистої архітектури можуть бути застосовані до будь-якого рішення або програмного забезпечення, незалежно від того, які парадигми програмування він використовує. Насправді, CA може бути використана будь-якою іншою архітектурою, яка слідує “чистим” принципам. Дотримання чистої архітектури при проектуванні і розробці рішення робить його перевіреним і більш незалежним від бібліотек, фреймворків, баз даних і забезпечує кращу ремонтопридатність в довгостроковій перспективі [2].

Серверна частина – ASP.NET Core.

ASP.NET Core – нова версія веб-фреймворку ASP.NET, в основному орієнтована на запуск на платформі .NET Core.

ASP.NET Core – це безкоштовний, відкритий та кросплатформенний фреймворк для створення хмарних додатків, таких як веб-додатки, програми IoT та мобільні сервери.

Переваги використання цього фреймворку:

1. Підтримка декількох платформ: ASP.NET Core програми можуть працювати на Windows, Linux і Mac.
2. Швидко: ASP.NET Core дозволяє включати пакети, які нам потрібні для нашого застосування. Це зменшує конвеєр запиту та покращує продуктивність та масштабованість.
3. IoC Container: він включає в себе вбудований IoC контейнер для автоматичного введення залежностей, що робить його ремонтопридатним і перевіреним.
4. Інтеграція з Modern UI Frameworks: дозволяє використовувати і управляти сучасними фреймворками інтерфейсу, такими як AngularJS, ReactJS, Umber, Bootstrap тощо, використовуючи Bower (менеджер пакетів для Інтернету).
5. Хостинг: веб-додаток ASP.NET Core може розміщуватися на декількох платформах з будь-яким веб-сервером, таким як IIS, Apache тощо.
6. Спільне використання коду: дозволяє побудувати бібліотеку класів, яку можна використовувати з іншими фреймворками .NET. Таким чином, одна база коду може бути розділена між фреймворками.
7. Side-by-Side App Versioning: ASP.NET Core працює на .NET Core, який підтримує одночасний запуск декількох версій додатків [3].

База даних – MS SQL Server, Entity Framework.

Клієнтська частина – ReactJS.

React – відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту веб-сторінки, з якими стикаються в розробці односторінкових застосунків. Розроблена американською корпорацією Meta у 2013 році.

Переваги використання фреймворку:

1. Легко освоїти та використовувати: ReactJS набагато легше вивчати та використовувати. Він має достатню кількість документації, посібників і навчальних ресурсів. Будь-який розробник із досвідом JavaScript може легко зрозуміти та почати створювати веб-додатки за допомогою React за кілька днів. Це частина V (view part) у моделі MVC (Model-View-Controller). Вона не є повнофункціональною, але має перевагу відкритої бібліотеки- інтерфейсу користувача JavaScript (UI), яка допомагає виконувати завдання кращим чином.

2. Створення динамічних веб-додатків стає легшим: створити динамічну веб-програму саме з рядками HTML було складно, оскільки це вимагає складного кодування, але React JS вирішив цю проблему та спростив її. Він забезпечує менше кодування та надає більше функціональних можливостей. React використовує JSX (розширення JavaScript), яке є особливим синтаксисом, що дозволяє використовувати лапки HTML і синтаксис тегів HTML для відтворення окремих підкомпонентів. Фреймворк також підтримує побудову машиночитаних кодів.

3. Багаторазові компоненти: веб-програма ReactJS складається з кількох компонентів і кожен компонент має власну логіку та елементи керування. Ці компоненти відповідають за виведення невеликого багаторазового фрагмента HTML-коду, який можна повторно використовувати будь-де, де це потрібно. Багаторазовий код допомагає спростити розробку та підтримку програм. Ці компоненти можна вкладати в інші компоненти, щоб дозволити створювати складні програми з простих будівельних блоків. ReactJS використовує механізм на основі віртуальної DOM для заповнення даними в HTML DOM. Віртуальний DOM працює швидко, оскільки змінює лише окремі елементи DOM замість того, щоб щоразу перезавантажувати повний DOM.

5. Підвищення продуктивності: ReactJS покращує продуктивність завдяки віртуальному DOM. DOM є кросплатформним і програмним API, який займається HTML, XML або XHTML. Більшість розробників зіткнулися з проблемою при оновленні DOM, що уповільнило продуктивність програми. ReactJS вирішив цю проблему, ввівши віртуальний DOM. React Virtual DOM повністю існує в пам'яті і є

представленням DOM веб-браузера. Завдяки цьому, коли ми пишемо React-компонент, ми не писали безпосередньо в DOM. Замість цього ми пишемо віртуальні компоненти, які реагують, перетворяться в DOM, що призведе до більш гладкої і швидкої продуктивності.

5. Наявність JavaScript бібліотеки: сьогодні ReactJS вибирає більшість веб-розробників. Це тому, що фреймворк пропонує дуже багату бібліотеку JavaScript. Бібліотека JavaScript надає більше гнучкості веб-розробникам.

6. Область застосування для тестування кодів: програми ReactJS надзвичайно легко тестувати. Фреймворк пропонує область, де розробник може тестувати та налагоджувати свої коди.

До того ж, наразі ReactJS широко користується популярністю за даними 2022 року [4] (рис. 1.2.):

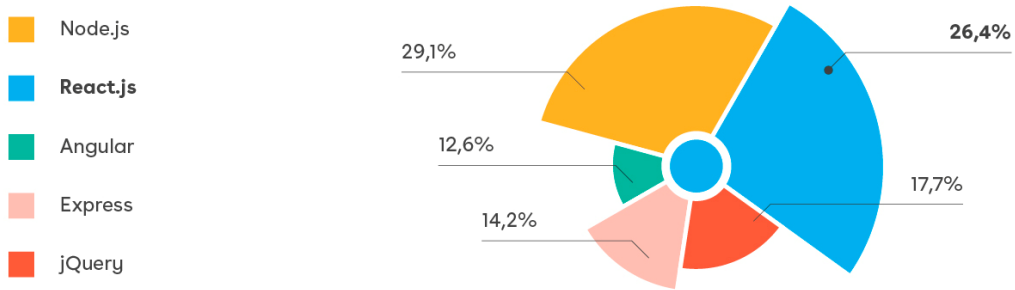

Рис. 1.2. Дані статистики використання популярних фреймворків

1.3. Аналіз продуктів найближчих конкурентів

Habitify

Це добре розроблений застосунок із простим, але привабливим інтерфейсом (рис. 1.3). Habitify представляє список звичок на день, і це дає змогу відзначати їх, коли здійснбється їх заповнення. Простий, але ефективний.

На додаток до основних функцій відстеження звичок, у Habitify також є темна тема, щоб зняти напругу з очей, і безліч крутих графіків і діаграм, які дозволять стежити за прогресом своєї звички.

Ціна: безкоштовно, преміум-версія за 4,99 доларів США на місяць.

Платформи: Android, iOS, Web.

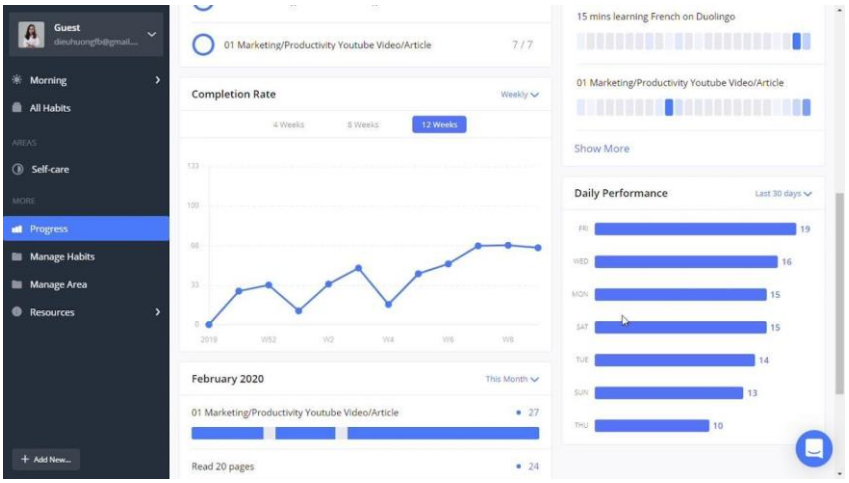

Рис.1.3. Вигляд застосунку Habitify

Coach.me

Додаток для відстеження звичок, який надає доступ до професійних тренерів і спільноти підтримки (рис. 1.4).

Відстеження своїх звичок є ефективним способом формування нових, але іноді потрібна додаткова мотивація. За своєю суттю Coach.me — це ефективний засіб відстеження звичок, який дозволяє перевіряти свої звички під час їх виконання. Однак справжня цінність полягає в аспектах спільноти та коучингу програми.

Цінною є функція спільноти. Для будь-якої поширеної звички, яку можна додати до свого списку, ймовірно, є ланцюжок обговорення, де можна зв’язатися з іншими користувачами Coach.me. Тут можна поділитися ідеями про те, як ефективно виконувати цю звичку, а також отримати підтримку та підбадьорення, коли важко.

Якщо потрібна більш персоналізована підтримка, можна відвідати розділ «Коучинг» програми. Тут можна отримати доступний коучинг для будь-якої звички. Тренер буде регулярно перевіряти процес, щоб переконатися, що користувач не

відстає від своєї звички. Якщо в учасника виникнуть проблеми, тренери можуть надати ідеї, як повернутися на правильний шлях.

Ціна: безкоштовно для відстеження звичок, 20 доларів на тиждень (і вище), щоб найняти тренера.

Платформи: Android, iOS, Web

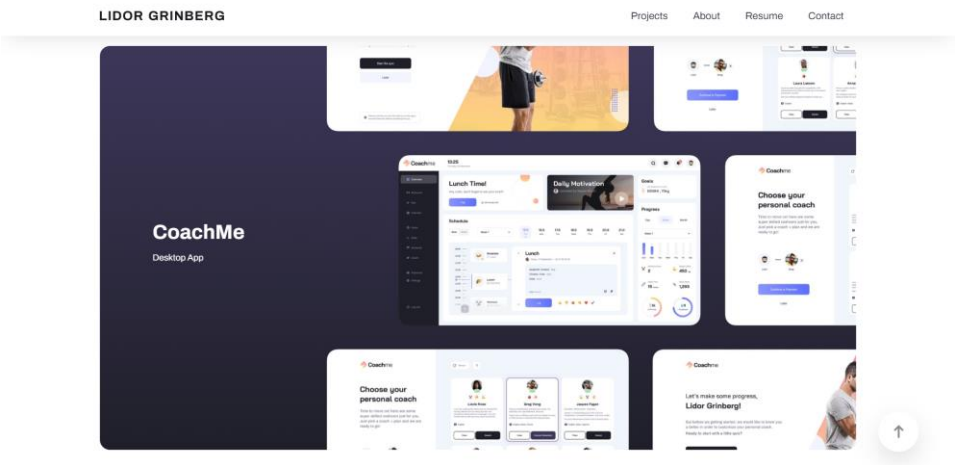

Рис. 1.4. Вигляд застосунку Coach.me

TickTick

TickTick сам по собі не є інструментом для відстеження звичок (рис.1.5). Швидше, це програма з повним списком справ, яка включає функції відстеження звичок.

Фактичні функції відстеження звичок у TickTick є основними, але це чудова програма, яку можна отримати, якщо потрібен один інструмент для відстеження як списку справ, так і звичок. Крім того, TickTick працює практично на будь-якому пристрої/платформі, яку можна придумати, навіть включаючи підтримку Apple Watch.

Ціна: безкоштовно, преміум-версія за 2,79 доларів США на місяць. Студенти та викладачі можуть отримати 25% знижки на рік TickTick Premium.

Платформи: Android, iOS, Mac, Windows, Web, Chrome, Firefox [5].

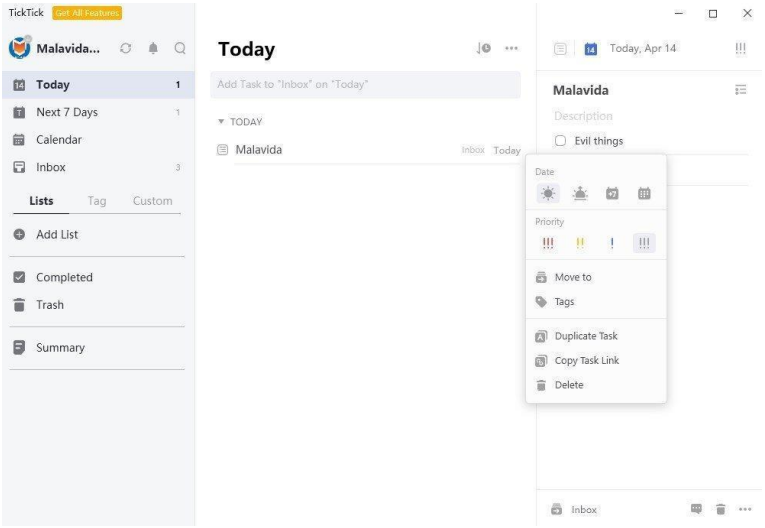

Рис. 1.5. Вигляд застосунку Tick Tick

1.4. Порівняння аналогів з трекером “Habit Rabbit”

Таблиця 1.1

Аналіз аналогів трекерів звичок

Критерії	Habitify	Coach.me	Tick Tick	Habit Rabbit
Інтуїтивно-зрозумілий інтерфейс	+	+	-	+
Сучасний дизайн	+	+	+	+
Цінова політика (безкоштовний)	умовно	умовно	умовно	+
Функціональність	+	+	-	+
Кросплатформеність	+	+	+	-
Відсутність реклами	-	-	-	+

Порівнюючи вищезазначені застосунки (бралися до уваги лише ті, які мають веб-версію), варто зазначити, що трекер звички Habit-Rabbit дещо поступається своїм конкурентам на даному етапі, бо деякі функції ще знаходяться в процесі розробки, а деякі заплановані у наступній версії, оскільки поточна версія має неповний функціонал та потребує доопрацювання. Перш за все, поки відсутня взаємодія з іншими користувачами веб-застосунку. Крім цього, відстежування прогресу за

більший проміжок часу не є таким обширним та інформативним як у застосунків-конкурентів.

Проте трекер вирізняється сучасним дизайном з гейміфікацією, простотою та лаконічністю користувацького інтерфейсу.

1.3. Постановка задачі

Постановка задачі є важливою частиною не лише кваліфікаційного проєкту, а й будь-якого іншого проєкту чи програмного продукту. Це визначення включає в себе окреслення мети і об'єктів проекту, а також визначення вимог та потреб користувачів, які повинен враховувати даний застосунок.

Крім цього, варто здійснити розробку ТЗ, що означає описати всі функціональні та нефункціональні вимоги до програмного продукту, вказати його архітектуру, технології та інструменти розробки (вказано у розділі 1).

Варто зазначити те, що у проєкті поняття “звичка” є взаємозамінним та синонімічним із словом “челендж”, оскільки процес формування звички є досить складним та займає певну кількість часу, що пропорційно проходженню челенджу.

Оскільки і мета, і об’єкт вказані у вступній частині, то зосередимо увагу на вимогах до застосунку:

Призначення застосунку

1. Застосунок призначений для полегшення процесу набування чи позбуття звичок користувача;
2. Трекер повинен забезпечувати підтримку системності у процесі роботи над звичками через встановлені користувачем параметри частоти виконання челенджів;
3. Habit Rabbit має допомагати особі відслідковувати прогрес для контролю цілей;
4. Застосунок має на меті заохотити користувача до виконання челенджів за допомогою елементів гейміфікації;

1. Реєстрація та автентифікація у модулі

Неавтентифікований користувач має доступ лише до сторінки автентифікації та реєстрації.

1.1. Реєстрація (якщо користувач не має акаунту) здійснюється при переході на гіперпосилання “Get started” на сторінці автентифікації:

1.1.1. Введення даних в поле “Username”;

1.1.2. Введення даних в поле “Email address”;

1.1.3. Введення даних в поле “Password”;

1.1.4. Кнопка “Register”;

1.1.5. Зберігання даних користувача після автентифікація.

1.2. Автентифікація здійснюється за допомогою введення логіну та паролю:

1.2.1. Введення даних в поле “Login”;

1.2.2. Введення даних в поле “Password”;

1.2.3. Кнопка “Login”;

1.2.4. Зберігання даних користувача після реєстрації.

2. Створення челенджів

Створювати челенджі може лише зареєстрований та попередньо автентифікований користувач.

2.1. Дія здійснюється через перехід за кнопкою “+” яка знаходиться у розділі “Challenges”, після чого відкривається модальне вікно:

2.1.1. Введення даних в поле “Title” (назва челенджу);

2.1.2. Введення даних в поле “Description” (детальніший опис);

2.1.3. Введення даних в поле “Units” (числове значення);

2.1.4. Вибір значення у компоненті select “Units” (години, сторінки, кілометри і тд.);

2.1.5. Вибір значення у компоненті select “Type” (позбутися чи набути звичку);

2.1.6. Вибір значення днів тижня для зазначення частоти виконання челенджу при натисканні на Toggle button;

2.1.7. Вибір значення кольору іконки кольору при натисканні на ColorPicker;

2.1.8. Вибір іконки при натисканні на IconPicker;

2.1.9. Вибір значення дати початку челенджу у DatePicker;

2.1.10. Вибір значення дати закінчення челенджу у DatePicker;

2.1.11. Додавання додаткових завдань (subtasks) при натисканні за кнопкою “+ Subtask” з введенням даних в поле “Titile”, “Units” та компоненті select “Units”;

2.1.12. Кнопка “Save”;

2.1.13. Збереження даних челенджу з відображенням на сторінці Challenges.

3. Редагування челенджів

Редагування челенджу відбувається у модальному вікні після натискання на Menu Item та вибору пункту “Edit”.

3.1. Введення даних в поля, які потрібно редагувати (з доступних);

3.2. Кнопка “Save”;

3.3. Збереження редагованих даних челенджу з відображенням на сторінці Challenges.

4. Видалення челенджів

Видалення челенджу відбувається у модальному вікні після натискання на Menu Item та вибору пункту “Delete”.

4.1. Кнопка “Cancel” та “Approve”;

4.2. Видалення челенджу на сторінці Challenges.

5. Робота із завданнями на день (Daily Taks)

Daily Taks — основна сторінка, з якою працюватиме користувач. Там і здійснюється позначення прогресу для кожного челенджу.

5.1. Кнопка “Done” для task з можливістю введення даних в поле або кнопкою для ітерацій;

5.2. Кнопка “Done” для subtask з можливістю введення даних в поле або кнопкою для ітерацій;

5.3. Кнопка “Undone” для task;

5.4. Кнопка “Undone” для subtask;

6. Перегляд профілю

Сторінка профілю призначена для перегляду інформації, досягнень користувача.

6.1. Кнопка “Delete”;

7. Перегляд челенджу

Перегляд челенджу відбувається у модальному вікні після натискання на Menu Item та вибору пункту “Details”.

7.1. Кнопка “Close”;

8. Перегляд прогресу

Перегляд загального прогресу відбувається при переході за посиланням “Challenges” у бічному меню.

Також прогрес по кожному челенджу можна переглянути перейшовши за переглядом челенджу індивідуально.

9. Гейміфікація

Гейміфікація відбувається у два етапи: присвоєння балів (points) та присвоєння статусів. Кількість отриманих користувачем балів виводиться у верхньому меню. Статус користувача можна переглянути у боковому меню під username.

Висновок до розділу 1

У результаті написання першого розділу, зроблено опис предметного середовища, а саме здійснено перелік технологій, які були використані для реалізації, тобто: React (для клієнтської частини), та ASP .Net Core, MS SQL Server, Entity Framework - для серверної. У розділі було описано загальні положення, переваги технологій та обраної архітектури рішення.

Підхід чистої архітектури допомагає відокремити залежності та ізолювати їх від бізнес-логіки програми і моделі домену. Основна ідея полягає в тому, щоб зробити рішення адаптивним і зберегти можливість використання програмної логіки незалежно від інтерфейсу та зовнішніх фреймворків. Цей підхід має кілька переваг, таких як незалежність від інтерфейсу користувача, баз даних, зовнішніх бібліотек, фреймворків і тестування. Чисту архітектуру можна застосувати до будь-якого рішення чи програмного забезпечення, а дотримання її принципів забезпечує її незалежність і можливість перевірки.

Робота над клієнською частиною здійснювалася за допомогою React. ReactJS — бібліотека JavaScript з відкритим вихідним кодом для створення інтерфейсів користувача, яка допомагає вирішити проблему часткового оновлення веб-вмісту. ReactJS є однією з найпопулярніших технологій серед розробників, адже її легко вивчити і застосовувати. Використання цієї технології дає змогу створювати динамічні веб-застосунки, що вимагають меншої кількості програмування і забезпечують більший функціонал. React надає можливість створювати повторно використовувані компоненти, що можна вкладати в інші компоненти для створення складних програм із простих блоків. Механізм на основі віртуальної DOM, який використовує ReactJS, допомагає швидко й ефективно наповнювати HTML DOM даними. Використання ReactJS допомагає спростити розробку та обслуговування додатків.

Також було здійснено опис та аналіз існуючих аналогів трекерів звичок. Серед головних конкурентів виділено ті програмні продукти, які мають веб-версію: Habitify, Coach.me та Tick Tick. Варто зазначити, що трекер звички Habit-Rabbit дещо

поступається своїм конкурентам на даному етапі, бо деякі функції ще знаходяться в процесі розробки, а деякі заплановані у наступній версії, оскільки поточна версія має неповний функціонал та потребує доопрацювання.

Варто зазначити те, що у проєкті поняття “звичка” є взаємозамінним та синонімічним із словом “челендж”, оскільки процес формування звички є досить складним та займає певну кількість часу, що пропорційно проходженню челенджу.

Крім цього, було сформоване технічне завдання для веб застосунку.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Аналіз предметної області

Концепція відстеження звичок передбачає систематичний запис і аналіз звичок для покращення якості життя та досягнення цілей. Основна ідея полягає в тому, що звички складаються з автоматично виконуваних повторюваних дій. Вони можуть бути корисними і водночас можуть завадити досягати бажаних результатів.

Аби позбутися негативних звичок та набути корисні, то важливо знати, які звички впливають на наші дії та поведінку. Для цього необхідно створити список звичок, яких ми хочемо позбутися або ж навпаки набути. Далі слід вести щоденний журнал звичок, нотуючи ті, які ми розвиваємо, і те, як часто ми їх розвиваємо. Для зручності широко використовуються трекери звичок у вигляді додатків чи веб застосунків.

Застосунки для відслідковування звичок дозволяють користувачам створювати список звичок, яких вони хочуть набути або ж позбутися. Для цього необхідно систематично вносити дані про виконання чи невиконання тих чи інших завдань.

Причиною вибору формату веб-застосунку було те, що застосунок буде орієнтований на людей, які більшість свого часу проводять за комп’ютером, працюючи чи навчаючись, і не хочуть зайвий раз відволікатись на телефон.

Перевагою є те, що це зручно та доступно. Користувач може в будь-який момент редагувати дані, отримувати оповіщення і тд. Окрім цього, в нагоді стає можливість відслідковувати прогрес, аби розуміти стан справ та тенденцію покращення якості життя.

Для розуміння концепту веб-застосунку трекінгу звичок потрібно розглянути поняття, які тісно з ним переплітаються, а саме:

1. Звичка – автоматизована дія людини, виконання якої за певних умов стає предметом стійкої потреби та закріплюється в поведінці. Є неусвідомленою формою поведінки, оскільки її реалізація не потребує роздумів або свідомих зусиль особи й відбувається мимовільно.

2. Трекер звичок – це, зазвичай, таблиця або календар, в якому фіксується виконання завдань для здобування чи позбуття звичок. Повторення дій відбувається доти, доки ці дії не стануть звичними і автоматичними.
3. Веб-застосунок – розподілений застосунок, у якому в ролі клієнта виступає браузер, а сервера – веб-сервер.

2.2. Use case діаграма

Use case діаграма є основною формою системних/програмних вимог до нового програмного продукту на етапі створення. Use cases вказують очікувану поведінку (що), а не точний метод її здійснення (як). Use cases можна позначити як текстове, так і візуальне представлення (тобто діаграма випадків використання). Ключова концепція моделювання випадків використання полягає в тому, що вона допомагає нам розробити систему з точки зору кінцевого користувача. Це ефективний метод передачі системної поведінки в термінах користувача шляхом визначення всієї зовнішньої видимої поведінки системи.

Use case діаграма використання зазвичай проста. Вона не показує деталі випадків використання, а:

- лише підсумовує деякі відносини між випадками використання, користувачами та системами;
- не показує порядок виконання кроків для досягнення цілей кожного випадку використання;

Для ефективної роботи над проектом також було розроблено use case діаграму (рис.2.1). На ній представлено дві ролі: гість (незареєстрований чи неавтентифікований) та користувач.

З рисунку видно, що гість має доступ лише до двох функцій веб-застосунку:

- реєстрація;
- автентифікація.

Користувач має доступ до таких функцій:

- перегляд персональної інформації;

- додавання челенджу;
- редагування челенджу;
- видалення челенджу;
- перегляд завдань на день та позначення прогресу;
- перегляд статистики челенджу.

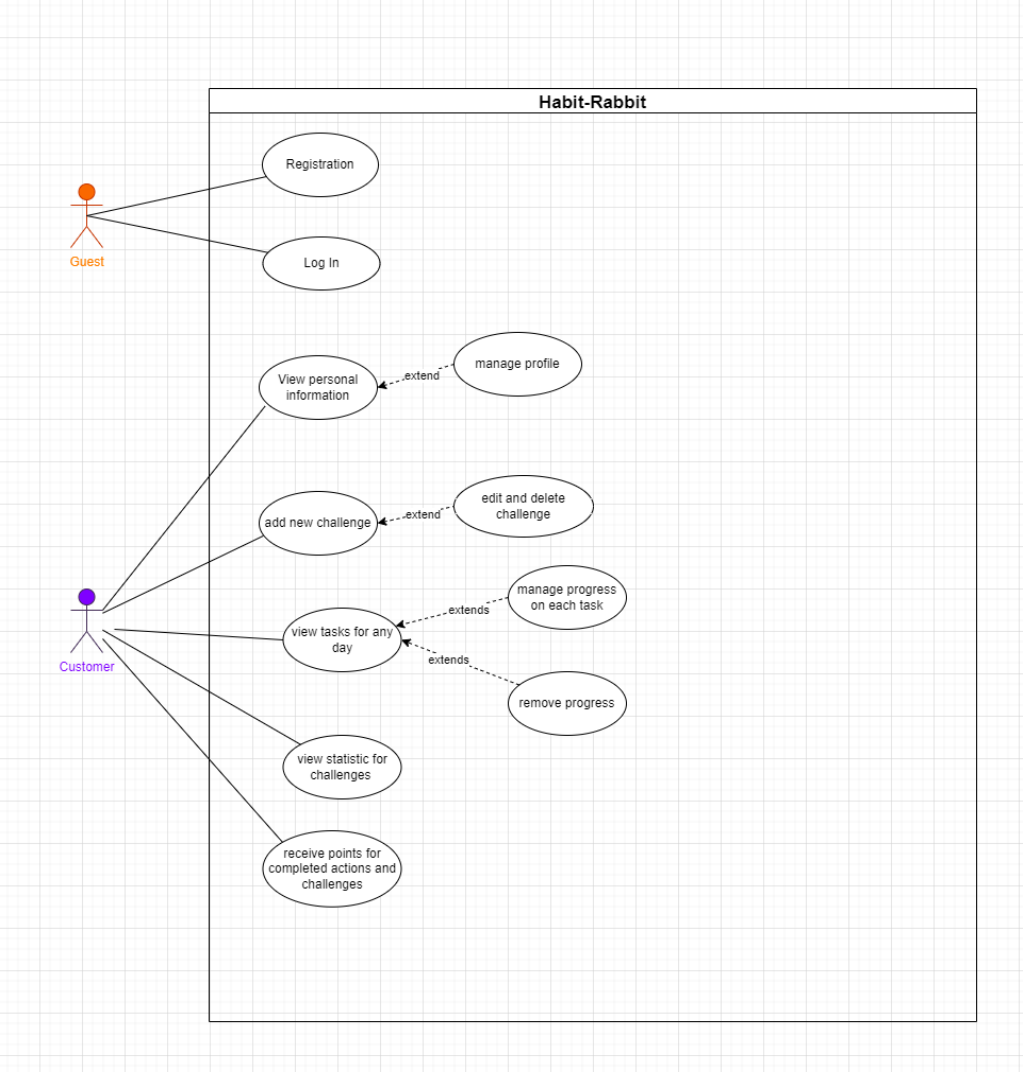

Рис. 2.1. Use case діаграма веб-застосунку

Висновоки до розділу 2

У результаті роботи над розділом 2 було виконано аналіз предметної області та продемонстровано роботу системи з точки зору користувача за допомогою Use case діаграми.

При аналізі предметної області було дано визначення базових понять для розуміння концепту веб-застосунку трекінгу звичок. Окрім цього було визначено, що аби позбутися негативних звичок та набути корисні, то важливо знати, які звички впливають на наші дії та поведінку. Для цього необхідно створити список звичок, яких ми хочемо позбутися або ж навпаки набути. Далі слід вести щоденний журнал звичок, нотуючи ті, які ми розвиваємо, і те, як часто ми їх розвиваємо. Для зручності широко використовуються трекери звичок у вигляді додатків чи веб застосунків.

За допомогою Use case діаграми розписано можливості доступу до функціоналу у ролі гостя (реєстрація та автентифікація) та користувача (перегляд персональної інформації, додавання челенджу, редагування челенджу, видалення челенджу, перегляд завдань на день та позначення прогресу, перегляд статистики челенджу). Use cases вказують очікувану поведінку (що), а неточний метод її здійснення (як). Use cases, можна позначити як текстове, так і візуальне представлення (тобто діаграма випадків використання). Ключова концепція моделювання випадків використання полягає в тому, що вона допомагає нам розробити систему з точки зору кінцевого користувача.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Visual Code

Visual Studio Code – це безкоштовне, відкрите, кросплатформне інтегроване середовище розробки програмного забезпечення (IDE), розроблене компанією Microsoft у 2015 році. VS Code підтримує багато мов програмування, таких як JavaScript, TypeScript, Python, C++, Go, Java, Ruby та інші.

Основні функції та переваги Visual Studio Code:

1. Зручність: Visual Studio Code – зручний редактор вихідного коду, що ідеально підходить для повсякденного використання. Він має підтримку сотень мов. VS Code допоможе бути продуктивними та швидкими у написанні коду, оскільки підтримує підсвічування (виділення) синтаксису, коригування відповідності дужок, автоматичний відступ, вибір блоку, фрагментів та багато іншого.
2. Підтримка завершення коду: для серйозного кодування часто використовуються інструменти з більшим розумінням коду, ніж просто блоками тексту. Visual Studio Code включає в себе вбудовану підтримку завершення коду IntelliSense, багате семантичне розуміння коду, навігацію та рефакторинг коду.
3. Відладка коду: коли програмування стає складним, то розробники використовують відладку. Visual Studio Code включає в себе інтерактивний інструмент відладки, тому є змога “проходитися” через вихідний код, перевіряти змінні, переглядати стеки викликів і виконувати команди в консолі.
4. Система контролю версій: VS Code має підтримку Git, щоб фахівці могли працювати з вихідним кодом, не залишаючи редактор, включаючи перегляд змін, які очікують на підтвердження. [6]

3.2. Rider

Для того, аби мати підключення до серверної частини, використовувався редактор коду **Rider** від JetBrains. Rider – крос-платформний IDE для .NET розробки на базі платформи IntelliJ і ReSharper. Райдер допомагає розробляти додатки .NET, ASP.NET,. NET Core, Xamarin і Unity на Windows, macOS або Linux. Він надає широкі можливості редагування та аналізу коду для мов, що використовуються в .NET розробці, такі як C #, VB.NET, F #, підтримує ASP.NET синтаксис Razor, JavaScript, TypeScript, XAML, XML, HTML, CSS, SCSS, JSON і SQL.

Переваги даного редактору:

1. Кросплатформений IDE: крім можливості запуску і налагодження різних додатків на різних операційних системах, сам Rider також підтримує кросплатформу і працює на Windows, macOS і Linux.
2. Підтримка різних .NET-проектів: Rider підтримує .NET Framework, нову платформу .NET Core і проекти на базі Mono. IDE дозволяє розробляти настільні додатки, .NET-сервіси та бібліотеки, ігри на рушії Unity, мобільні додатки Xamarin, ASP.NET веб-додатки та ASP.NET Core.
3. Швидкість і функціональність: Rider забезпечує понад 2200 перевірок коду, сотні контекстних дій та рефакторингів, запозичених у ReSharper, у поєднанні з передовими середовищами розробки на основі платформи IntelliJ. Незважаючи на безліч функцій, Rider є швидким і чутливим IDE [7].

3.3. Swagger

Для роботи з API використовували **Swagger**. Swagger – відкритий набір правил, специфікацій та інструментів для розробки та опису RESTful API. Фреймворк Swagger дозволяє розробникам створювати інтерактивну, машинну та людино-читабельну документацію API.

Специфікації API зазвичай включають таку інформацію, як підтримувані операції, параметри та виходи, вимоги до авторизації, доступні кінцеві точки та необхідні ліцензії. Swagger може генерувати цю інформацію автоматично з вихідного коду, попросивши API повернути файл документації з його анотацій. Swagger допомагає користувачам створювати, документувати, тестувати та споживати веб-сервіси RESTful.

Переваги Swagger:

1. Він має дружній інтерфейс користувача, який відображає план для API;
2. Документація зрозуміла як розробникам, так і не розробникам, таким як клієнти або менеджери проектів;
3. Підтримує створення бібліотек API більш ніж на 40 мовах;
4. Формат прийнятний в JSON і YAML, що дозволяє легке редагування;
5. Допомагає автоматизувати процеси, пов'язані з API [8].

3.4. Git

Для управління версіями використано Git. Git є найпопулярнішою і широко використовуваною сьогодні системою управління версіями. Спочатку розроблений як проект з відкритим вихідним кодом у 2005 році творцем операційної системи Linux. Величезна кількість розробників покладається на контроль версій Git для розробки декількох типів комерційних і некомерційних проектів.

Які можливості надає Git:

1. Відновлення файлів коду до попереднього стану;
2. Повернення проекту до попереднього стану;
3. Порівняння змін коду за певний період часу;
4. Вказує на того, хто останнім редагував певний фрагмент коду (допоможе у пошуці джерела помилки).

Переваги використання Git:

1. Продуктивність: нові зміни коду можна легко здійснити, гілки версій можна легко порівнювати та об'єднувати, а також оптимізувати код для кращої роботи.

2. Безпека: Git розроблений спеціально для підтримки цілісності вихідного коду. Вміст файлу, а також зв'язок між файлом і каталогами, тегами, комітами, версіями і т.д. захищені криптографічно за допомогою алгоритму під назвою SHA1, який захищає код і змінює історію від випадкових, а також шкідливих пошкоджень.

3. Гнучкість: ключовою метою дизайну Git є гнучкість, яку він пропонує для підтримки декількох видів нелінійних робочих процесів розробки та її ефективності в обробці як малих, так і великомасштабних проектів, а також протоколів [9].

3.5. Figma

Макети сторінок розроблялися за допомогою застосунку **Figma.** Figma – це інструмент для розробки користувальницьких інтерфейсів, базований на браузері, який дозволяє користувачам працювати разом для створення яскравих та інтерактивних прототипів. З моменту випуску в 2016 році Figma стала популярним інструментом як в індустрії веб-дизайну, так і в онлайн-спільнотах. Figma підтримує роботу в браузері, а також має десктопні додатки для ПК з операційними системами IOS та Windows.

Figma дозволяє користувачам співпрацювати в режимі реального часу для створення та прототипу користувацьких інтерфейсів та веб-додатків. Використовуючи інструменти векторного дизайну графіки, команди дизайнерів можуть створювати складні макети каркасів для веб-сайтів, які можуть бути змінені для оптимізації для екранів будь-якого розміру.

Основний функціонал Figma включає в себе:

1. Векторна графіка: функція Vector Figma дає можливість користувачам створювати якісний дизайн інтерфейсу через векторні ілюстрації та графіку.

2. Прототипування: собливості прототипування цього інструменту дозволяють перевіряти функціональність та дизайн інтерфейсу перед розробкою.

3. Спільна робота: графічний редактор надає змогу працювати над проектами з декількома співробітниками та оновлювати дизайни в режимі реального часу, маючи безперебійне спілкування в голосовому чаті.

4. Стилі та бібліотеки: Figma дає можливість встановлювати бібліотеки, щомістять повторювані елементи та стилі. Вони особливо корисні для підтримки однорідності(подібності) дизайну на всіх етапах проекту [10].

3.6. Material UI

За основу для дизайну взято Material UI в комбінації з шаблоном Minimal UI. Material UI (MUI) – це CSS фреймворк, який надає компоненти React поза коробкою і слідує за Material Design Google, запущеним у 2014 році. MUI дає можливість використовувати різні компоненти для створення інтерфейсу для веб і мобільних додатків. MUI містить рекомендації для типографіки, сітки, простору, масштабу, кольору, зображень і т.д.

Переваги Material UI:

1. Попередньо розроблені компоненти інтерфейсу користувача: MUI постачає кілька попередньо розроблених компонентів, які можна додати до свого засосунку, що забезпечить привабливий, простий у використанні та візуально привабливий дизайн та швидку реалізацію.

2. Material Design – це продумана система дизайну, яка описує значення і випадки використання компонентів. З Material, наприклад, можна використовувати Material Icons, тобто вибрати значки, які відповідають системі дизайну.

3. Регульована тема: MUI надає прості теми встановлення та налаштування для використання та глобальної реалізації для всіх доступних компонентів, що робить його високофункціональним та динамічним. Таким чином, колір теми компонента, інформація про палітру, властивості та деякі інші стилі можуть бути налаштовані, що означає, що компоненти будуть подібними (однорідними).

4. Добре структурована документація: MUI має зрозумілу та добре структуровану документацію, яка включає в себе посібники з прикладами коду для практики.

5. Широка підтримка: MUI має велику підтримку для виправлення помилок і проблем, завдяки своїй постійно оновлюваній бібліотеці [11].

Висновки до розділу 3

У цьому розділі дійснено опис засобів розробки та їхніх переваг (які стали причиною для використання цих продуктів під час роботи над веб-застосунком).

Підсумовуючи, варто зазначити, що для написання коду було використано Visual Studio Code, для запуску серверної частини та внесення деяких правок - Rider, для роботи з API - Swagger, для контролю версій - Git, для розробки дизайну - Figma, а Material UI став фреймворком, з якого використовувалися деякі готові React компоненти та стилі.

Для роботи з клієнтською частиною переважно використовувався Visual Studio Code – зручний редактор вихідного коду з підтримкою значної кількості мов програмування. Редактор допомагає бути продуктивним та швидким у написанні коду. Цьому сприяє підсвічування синтаксису та автоматичний відступ. Також Visual Studio Code має підтримку завершення коду IntelliSense й інструмент відладки, який дозволяє розробникам здійснювати перевірку змінних, переглядати стеки викликів та виконувати команди в консолі. Тому, VS Code є потужним інструментом для розробки програмного забезпечення.

Git є найпопулярнішою системою управління версіями, що надає користувачам багато можливостей, серед яких: відновлення файлів коду до попереднього стану, повернення проекту до попереднього стану, порівняння змін коду за певний період часу та вказує на того, хто востаннє редагував якийсь фрагмент коду. Використання Github надає також безліч переваг, серед яких: продуктивність, безпека, гнучкість. Усе це дозволяє розробникам ефективно працювати з різними проектами.

Для роботи з дизайном комплексно використовувалися Material UI та Minimal UI Kit. Material UI – це бібліотека компонентів для створення інтерфейсів веб та мобільних додатків відповідно стандартам Material Design. Дана бібліотека має декілька переваг, серед яких: наявність попередньо розроблених компонентів, продумана система дизайну, налаштовувані теми, добре структурована документація та широка підтримка. Розробники застосовують Material UI, щоб

пришвидшити розробку і надати своїй програмі стандартизований і привабливий дизайн.

Для розробки макетів використано Figma – інструмент для розробки користувацьких інтерфейсів, що став надзвичайно популярним у веб-дизайні. Figma дозволяє створювати складні макети для веб-сайтів та оптимізувати їх до екранів різного розміру, використовуючи векторну графіку. Застосунок дає можливість колективно працювати над проектами з кількома розробниками та надає перевагу оновлення дизайну в режимі реального часу. Крім цього, приємним бонусом є функція безперебійного спілкування в голосовому чаті. Ще Figma дозволяє встановлювати бібліотеки, які містять повторювані елементи та стилі. Це забезпечує однорідність дизайну.

РОЗДІЛ 4. Реалізація клієнтської частини

Вирішено створити клієнтську частину окремим проектом для зручності розробки. Завдяки чистій архітектурі це не викликало жодних труднощів. Бо все, про що має знати клієнт, - це адреси, на які можна надсилати запити. В той же час, сервер взагалі нічого не знає про клієнта.

4.1. Дизайн

Для дизайну інтерфейсу було вирішено використати бібліотеку Material UI. Це безкоштовна бібліотека, в якій є готові реакт-компоненти, реалізовані в одному стилі Material. Також знайдено готовий шаблон Minimal UI Kit (рис. 4.1), який використано для проєкту.

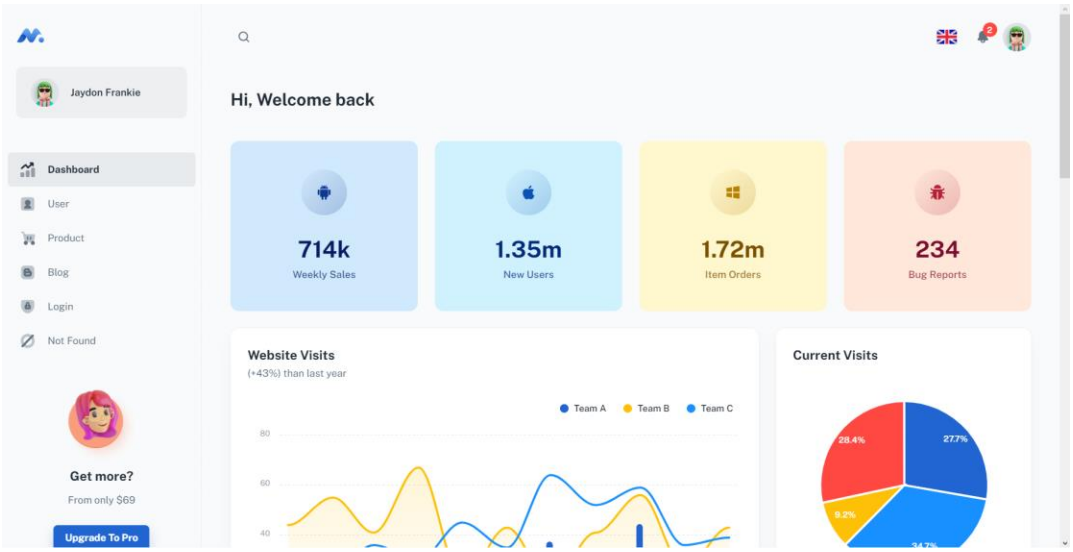

Рис. 4.1. Minimal UI Kit Dashboard

Він чудово підходить для застосунку Habit-Rabbit, так як виконаний у досить яскравих, та в той же час, м’яких кольорах, а до більшості елементів застосовано скруглення країв. Такий дизайн приємний для ока та заохочує щоразу повертатися в додаток. М’які кольори та заокруглення згладжують, і без того такий складний, процес набуття звички, а яскраві кольори піднімають настрій та перетворюють все на захоплюючу гру. В поєднанні зі зручним та продуманим функціоналом, дизайн

привабить кожного, перед ким постало непросте завдання - формування власного характеру (адже всі ми знаємо, що “посієш звичку - пожнеш характер”).

Для того, щоб наш продукт виділявся з-поміж інших, нами було розроблено унікальний логотип (рис. 4.2). Для цього було використано інструменти графічного редактору Figma.

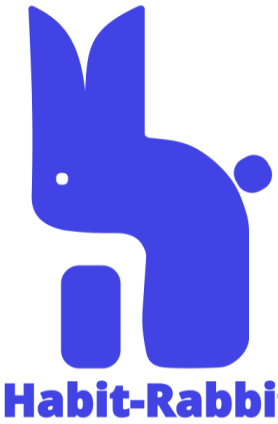

Рис. 4.2. Логотип Habit-Rabbit

Крім цього, у даному графічному редакторі було розроблено макети сторінок, згідно яких створено сторінки чи окремі компоненти проекту.

Головна сторінка, яка відображається після логування користувача. На розділі Daily tasks користувач відмічатиме прогрес по кожній із звичок. Розроблено дизайн для завдань, які мають subtasks чи не мають їх (рис. 4.3).

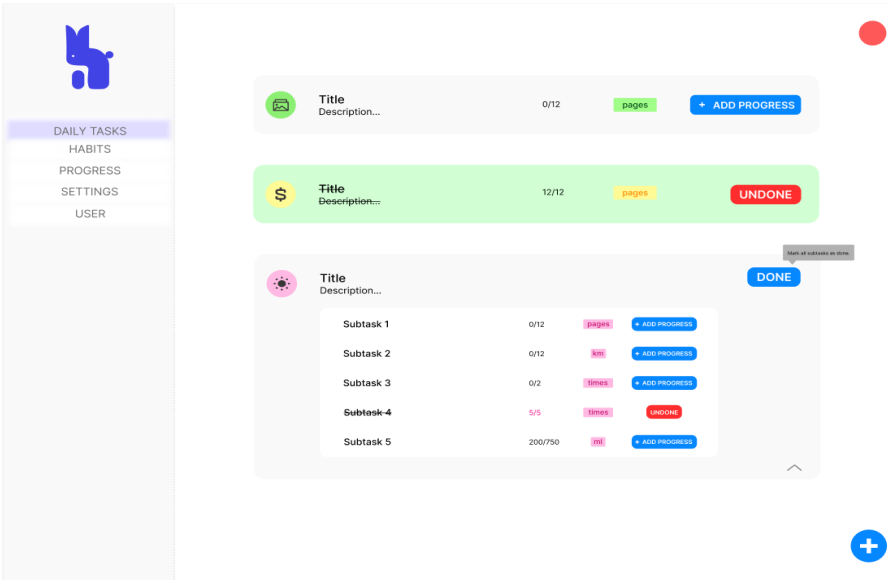

Рис. 4.3. Макет сторінки Daily Tasks

Сторінка челенджів, що містить перелік усіх челенджів, де виводяться основні дані (рис. 4.4).

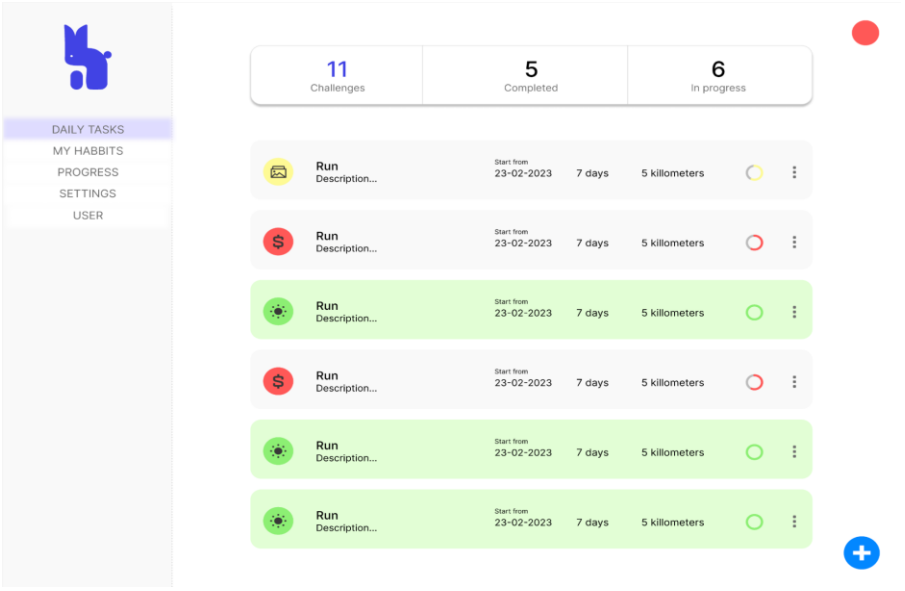

Рис. 4.4. Макет сторінки Challenges

Також розроблено дизайн модального вікна, яке використовується для створення звички (рис. 4.5):

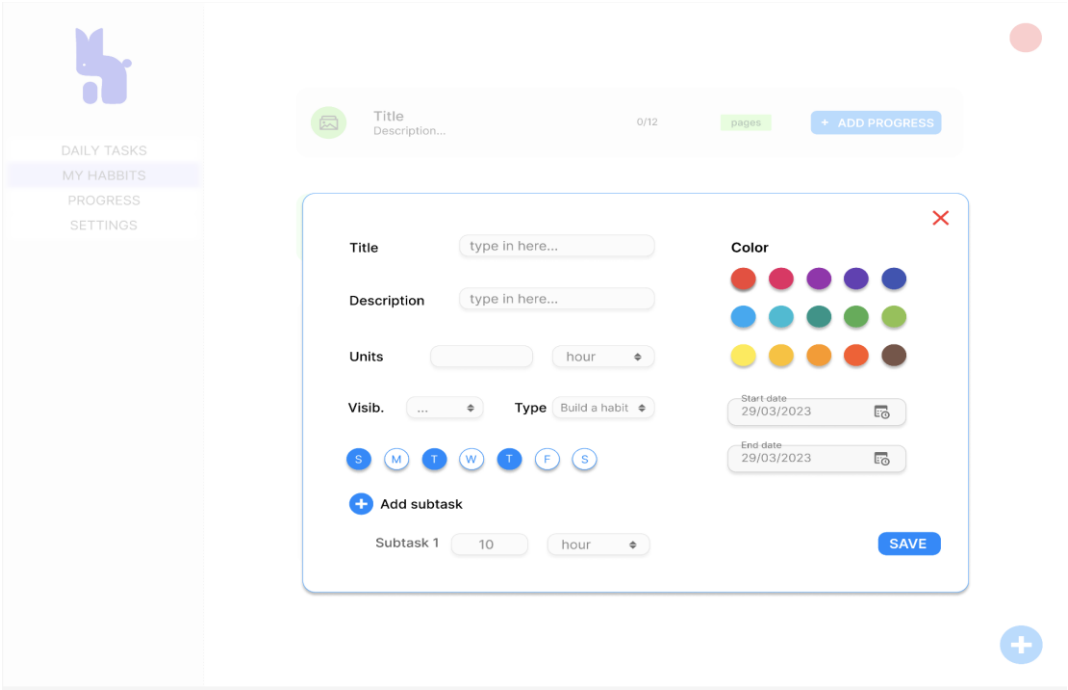

Рис. 4.5. Макет модального вікна для створення звички

Макет сторінки профілю користувача (рис. 4.6):

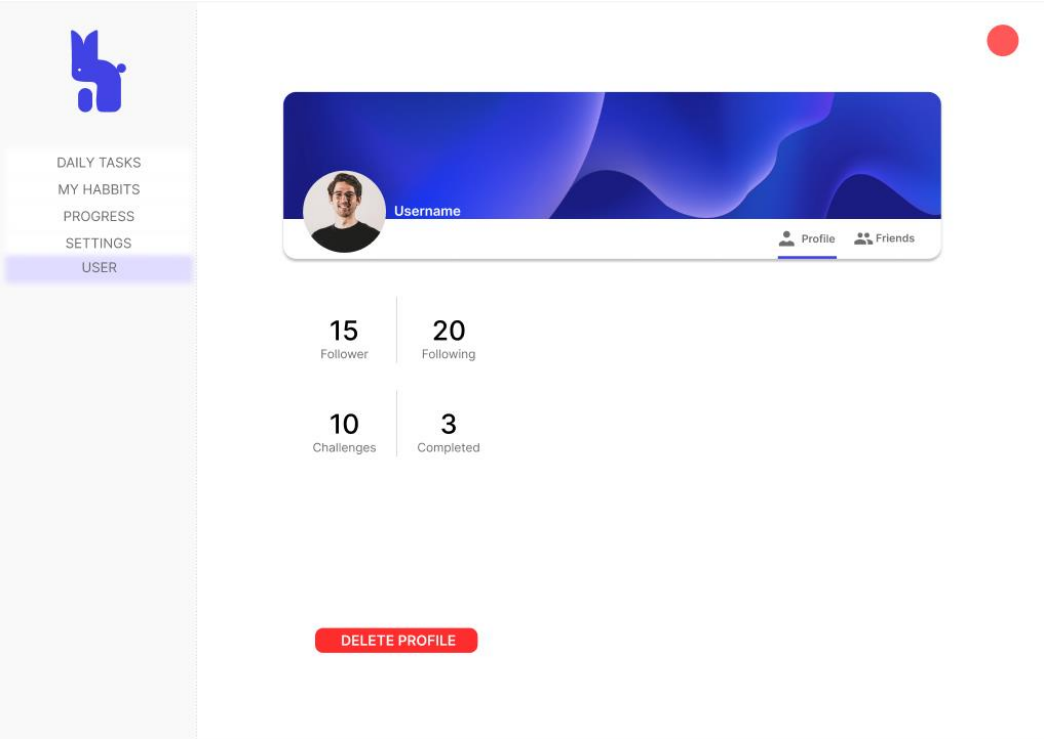

Рис. 4.6. Макет сторінки профілю

4.2. Архітектура проекту

Робота над трекером розпочалася з імплементації MUI та шаблону Minimal UI. Для взаємодії з пакетами було обрано **npm**. npm —менеджер пакетів для JavaScript, найбільший у світі реєстр програмного забезпечення. Розробники з відкритим вихідним кодом використовують npm для спільного використання та запозичення пакетів.

Як уже було зазначено, для створення користувацького інтерфейсу було обрано React JS. Ця бібліотека є надзвичайно гнучкою. Тому це хороший вибір, якщо передбачається, що продукт використовуватиметься на різних платформах.

Проект має свою архітектуру (рис. 4.7), яка дозволяє відокремлювати різні типи файлів один від одного.

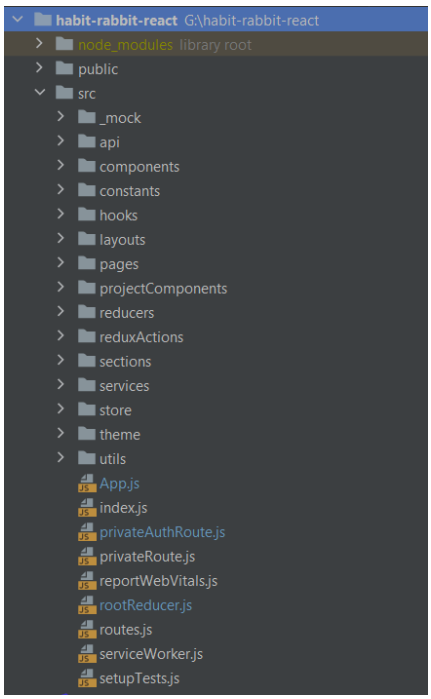

Рис. 4.7. Клієнтська частина проекту. Рівень UI

У проекті є дві основні папки: public та src. У public знаходиться файл index.html та зображення, які використовуються в проекті. Натомість, вся основна частина знаходиться в папці src. Для цієї папки розроблено певну архітектуру, яка складається з папок за призначенням. Окремо є папка для всіх компонентів, є папка для запитів на сервер, для сервісів, які є проміжною інстанцією між компонентами та http-запитами, для констант і т.д.

4.3. Реалізація автентифікації та реєстрації

Для цього використовувалися такі модулі:

- react-redux;
- redux;
- redux-thunk;
- react-router-dom.

Для реалізації було виконано такі кроки:

1. Спершу було створено папку **services** та файли authentication.js, token.js та user.js.

1.1. Authentication.js

Register() отримує значення, що задаються користувачем {username, email, password}. Далі надсилається запит на реєстрацію в API-інтерфейс, що на сервері, використовуючи модуль authenticationService. Запит успішний, якщо відповідь сервера обробляється у функції then, де встановлюється токен доступу, та здійснюється перехід на сторінку/dashboard/app, а сама сторінка оновлюється.

login() отримує значення, що задаються користувачем {username, password} & save JWT to Local Storage. Далі надсилається запит на вхід в API-інтерфейс на сервері, використовуючи модуль authenticationService. Запит успішний, якщо відповідь сервера обробляється у функції then, де встановлюється токен доступу, та здійснюється перехід на сторінку/dashboard/app, а сама сторінка оновлюється.

LogoutUser() здійснює виклик API-інтерфейсу сервера для виходу з системи, використовуючи модуль authenticationService. Запит успішний, якщо встановлюється значення виходу із системи та здійснюється очищення токенів. Викликається ця функція тоді, коли юзер натискає кнопку виходу із застосунку.

1.2. Token.js

Код у файлі описує клас tokenService, що має статичні методи для управління access та refresh токенами в Local Storage:

- **getLocalRefreshToken ()** отримує refresh токен в Local Storage;
- **getLocalAccessToken ()** отримує access токен в Local Storage;
- **setLocalRefreshToken (refreshToken)** зберігає refresh токен в Local Storage;
- **setLocalAccessToken (accessToken)** зберігає access токен в Local Storage;
- **deleteTokens ()** видаляє access та refresh токені в Local Storage.

1.3. User.js

Код отримує функцію getUserInfo (), яка отримує користувацьку інформацію з API за допомогою методу getUser () модуля usersService.

2. Далі послідовно було створено папки **ReduxActions** та **auth** та файли index.js та types.js.

2.1. Types.js

LOGOUT - константа, яка містить рядковий ідентифікатор "LOGOUT", що використовується для виявлення виходу з облікового запису у системі Redux.
SET_ACCESS - константа, яка відповідно призначена для виявлення встановлення доступу у системі Redux.

2.2. Index.js

Код файлу містить logout, що повертає подію з типом LOGOUT, яка диспечеризується, коли користувач вийшов із застосунку.

SetAccess, що повертає подію з типом SET_ACCESS та об’єкт payload, що містить токен. Ця подія диспечеризується, коли користувач входить у застосунок або оновлено токен доступу.

3. Після цього здійснювалася робота з Redusers, безпосередньо з файлом authReducer.js.

Reducer має початковий стан initialState, що має властивості role і isAuthUser (чи авторизований користувач).

У залежності від типу action, який приходить в Reducer через параметр action, State може бути змінений в залежності від типу action, що приходить в Reducer через параметр action.

Крім цього, Reducer також застосовує різні services, щоб працювати з access tokens та повідомляти про помилки.

4. Взаємодія із сторінками реєстрації, аутентифікації.

Розглядається приклад сторінки Login. У коді token застосовується таким чином: у функції onFinish, що здійснюється при успішному вході користувача, передається об'єкт values та історія браузера до login(), що описана в файлі authentication.js. Функція login() відправляє запит на сервер з об'єктом values, щоб перевірити правильність логіну та паролю. Якщо ж логін та пароль правильні, то сервер повертає access tocken. tokenService.deleteTokens() у функції useEffect виконується з метою видалення токенів, що можуть бути збережені в Local Storage перед входом користувача.

Візуалізація реалізованого коду автентифікації (рис. 4.8) та реєстрації (рис. 4.9):

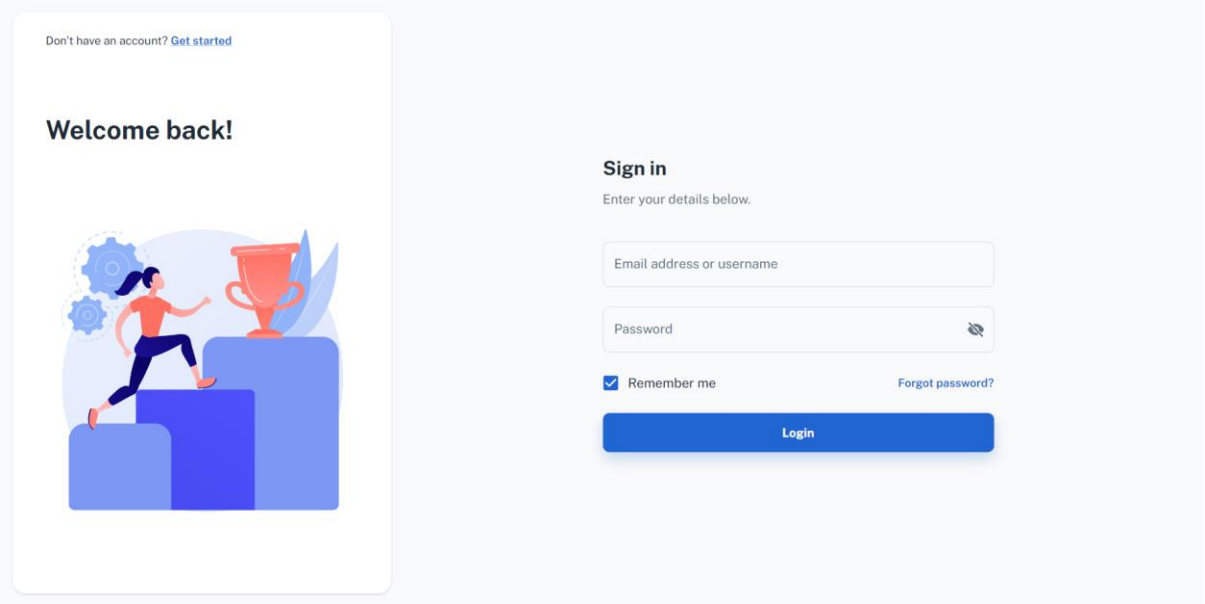

Рис. 4.8. Автентифікація користувача

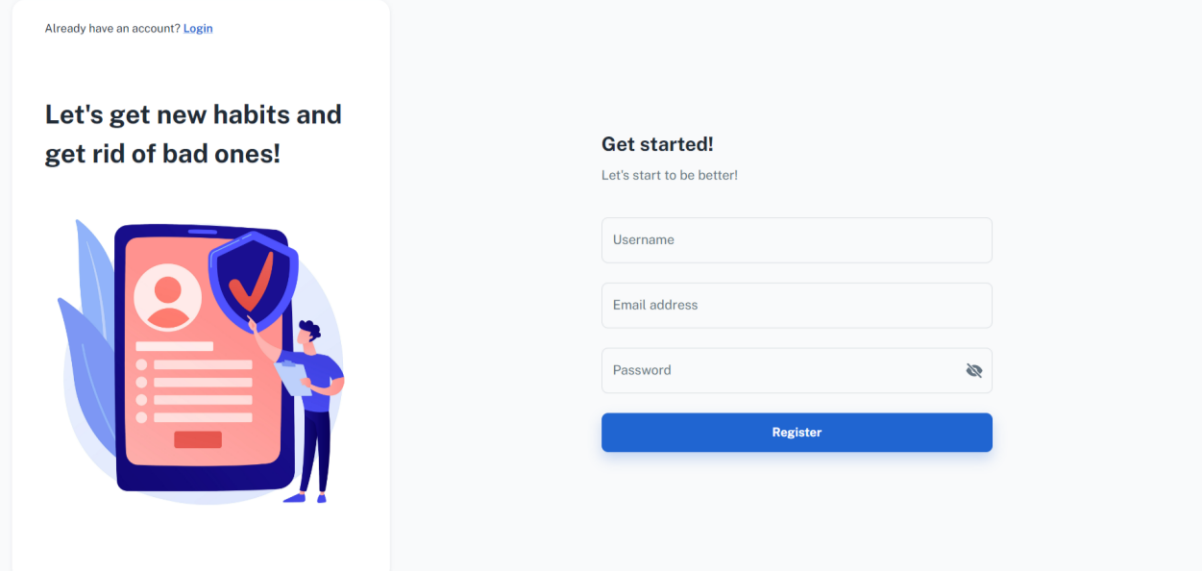

Рис. 4.9. Сторінка реєстрації

4.4. Створення челенджів

Заповнення даних для челенджу здійснюється у модальному вікні (рис. 4.10), тому першочергово було створено компонент AddChallenge, що для зручності складається з “менших” компонентів (аби код основного компоненту не був таким великим та загалом код читався легше): FieldsMain, IconRadios, MuiDateRagePicker, ToggleDays.

1. *FieldsMain* - компонент, що відповідає за наповнення модального вікна. Тобто, усі текстові поля, кнопки і тд.
2. *IconRadios* - компонент вибору іконок (за принципом Picker-a).
3. *MuiDateRagePicker* - компонент вибору дати початку та кінця челенджу.
4. *ToggleDays* - компонент вибору днів тижня, коли потрібно виконувати челендж (за принципом Picker-a).

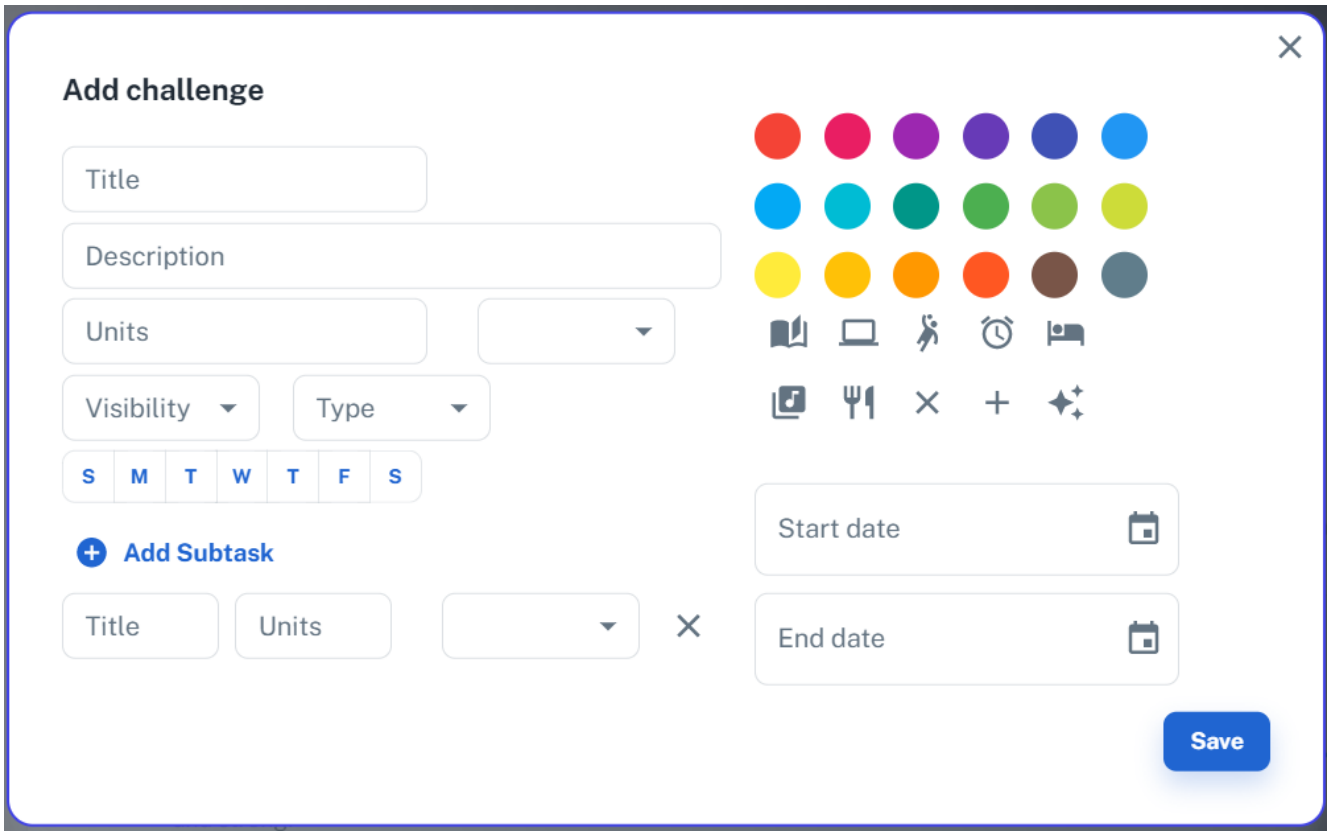

Рис. 4.10 Модальне вікно створення челенджу

Код компоненту AddChallenge визначає змінні стану, використовуючи hook useState(), а саме: color, startDate, endDate, addSubtask, toggleDays, selectedIcon та startDateError. Крім цього, визначає функції для обробки змін змінних стану, що включає handleChangeColor, handleChangeStartDate, handleChangeEndDate, onSubmit та showPickedIcon.

AddChallenge ще містить кнопку для відправки форми, що під час натискання викликає функцію onSubmit, щоб створити новий виклик, використовуючи hook useCreateChallenge. onSuccess() виводить повідомлення в консоль і здійснює перезавантаження сторінки, а onError() виводить повідомлення про помилку в консоль.

Для того, щоб усі поля були заповнені, потрібно отримати деякі значення із сервера. До прикладу, отримуємо значення Unit. За подібним принципом отримуємо Freaquency (лістинг 1).

```
import { useMutation } from "react-query";
import instance from "../configurations/configurations";
```

```
import { SERVER_URL } from "../../constants/api/urls";

const getUnitMutationFn = () => {
  return instance.get(`${SERVER_URL}/Unit`, false).then((res) => res.data);
};

export const useGetUnit = ({ onSuccess, onError }) => {
  const { mutate: getUnit } = useMutation(getUnitMutationFn, {
    mutationKey: 'getUnit',
    onSuccess,
    onError,
  });
  return { getUnit };
};
```

Лістинг 1. UseGetUnit

Далі розглянемо файл useCreateChallenge (лістинг 2). Цей хук повертає об'єкт з однією властивістю createChallenge, що є функцією, яка викликає мутацію. Мутація відправляє POST-запит до сервера з використанням змінної instance та передає об'єкт challenge як тіло запиту. При успішному виконанні функція createChallengeMutationFn повертає дані відповіді від сервера.

```
import { useMutation } from "react-query";
import instance from "../configurations/configurations";
import { SERVER_URL } from "../../constants/api/urls";

const createChallengeMutationFn = (challenge) => {
  return instance.post(`${SERVER_URL}/Challenge/create`, challenge, false).then((res) => res.data);
};

export const useCreateChallenge = ({ onSuccess, onError }) => {
  const { mutate: createChallenge } = useMutation(createChallengeMutationFn, {
    mutationKey: 'createChallenge',
    onSuccess,
```

```
  onError,
 });
 return { createChallenge };
};
```

Лістинг 2. Код файлу useCreateChallenge

4.5. Виведення челенджів

Кожен челендж відображається як картка з інформацією (назва, опис, дата початку, дата закінчення, кількість днів та Units, прогрес). Компонент використовує бібліотеку MUI для компонентів користувацького інтерфейсу. Із зовнішнього API компонент отримує дані виклику, використовуючи хук useGetChallenges. Хук приймає onSuccess() та onError().

За допомогою map() відображається список викликів відображається. Функція проходиться по масиву та відображає компонент Card для кожного челенджу (рис. 4.11).

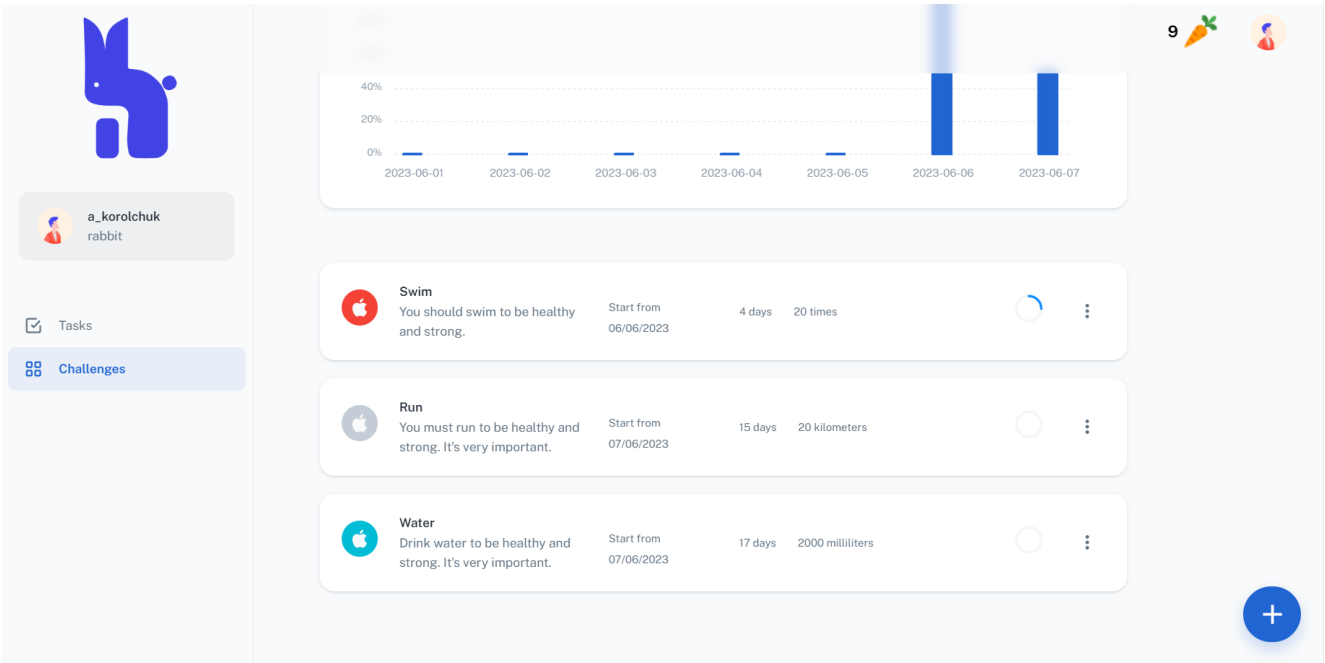

Рис. 4.11. Вивід челенджів

4.6. Виведення завдань на щодень

Виведення завдань на щодень відбувається при переході на пункт меню “Daily tasks” (рис. 4.12).

Використані хуки та функції:

1. useState використовується для управління станом поточного дня, списком завдань та тим, чи має бути відкритий діалог для додавання прогресу.
2. useEffect використовується для отримання списку завдань на поточний день під час монтажу компонента або при зміні поточного дня.
3. handleOpen() використовується для відкриття діалогового вікна для додавання прогресу.
4. користувацькі хуки: `useAddProgress` та `useRemoveProgress`, обробляють додавання та видалення прогресу для завдань.
5. markTaskAsDone() викликається, коли користувач натискає кнопку "Виконано" для завдання. Вона викликає функцію addProgressChallenge з власного хука, яка додає прогрес до завдання.

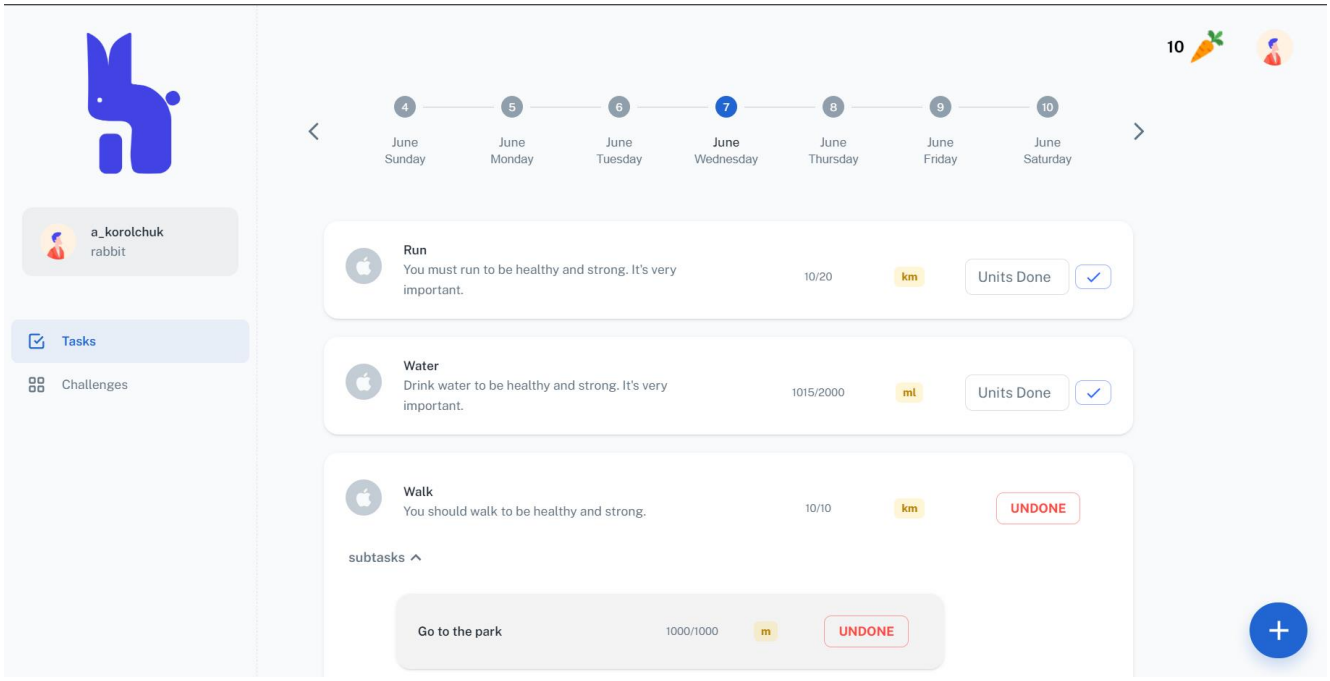

Рис. 4.12. Вивід завдань на день

4.7. Редагування челенджів

Заповнення даних для челенджу здійснюється у модальному вікні (рис. 4.13), тому першочергово було створено компонент EditChallenge, що для зручності складається з “менших” компонентів (аби код основного компоненту не був таким великим та загалом код читався легше) : FieldsMain, MuiDateRagePicker, ToggleDays.

Використані хуки та функції:

1. useGetChallenge - хук для отримання даних про завдання з API, що приймає об'єкти onSuccess та onError у якості аргументів. Зворотний виклик onSuccess встановлює стан з отриманими даними за умови, якщо дані отримані успішно. Відповідно, якщо виникла помилка, onError відображає її у консолі.
2. *useUpdateChallenge* - хук для оновлення даних завдання, що приймає об'єкти onSuccess та onError у якості аргументів. Зворотний виклик onSuccess закриває модальне вікно та перезавантажує дані. Відповідно, якщо виникла помилка, onError відображає її у консолі.

3. *useForm* - хук з бібліотеки react-hook-form, що використовується для управління даними форми, повертаючи об'єкт з методами та властивостями, що використовуються для реєстрації полів форми, обробки подання форми та скидання форми.
4. useMutation з бібліотеки react-query використовується для виконання мутацій (оновлення даних на сервері) в React компонентах з підтримкою стану завантаження (loading), помилок (error) та успішного завершення (success). Ця функція отримує два аргументи: mutationFn, який виконує мутацію, і options, який містить додаткові параметри.

5. mutationFn - це функція updateChallengeMutationFn, яка використовує Axios для відправки POST запиту на сервер з об'єктом challenge та повертає дані з відповіді сервера у форматі JSON. Параметри onSuccess та onError є колбеками, які викликаються після успішного виконання або помилки виконання запиту відповідно.

6. useUpdateChallenge - це обгортка для useMutation, яка дозволяє використовувати функцію updateChallenge в React-компонентах. Компонент, який викликає цю функцію, повинен передати об'єкт з колбеками onSuccess та onError. Код також використовує константи SERVER_URL та instance для налаштування взаємодії з сервером.

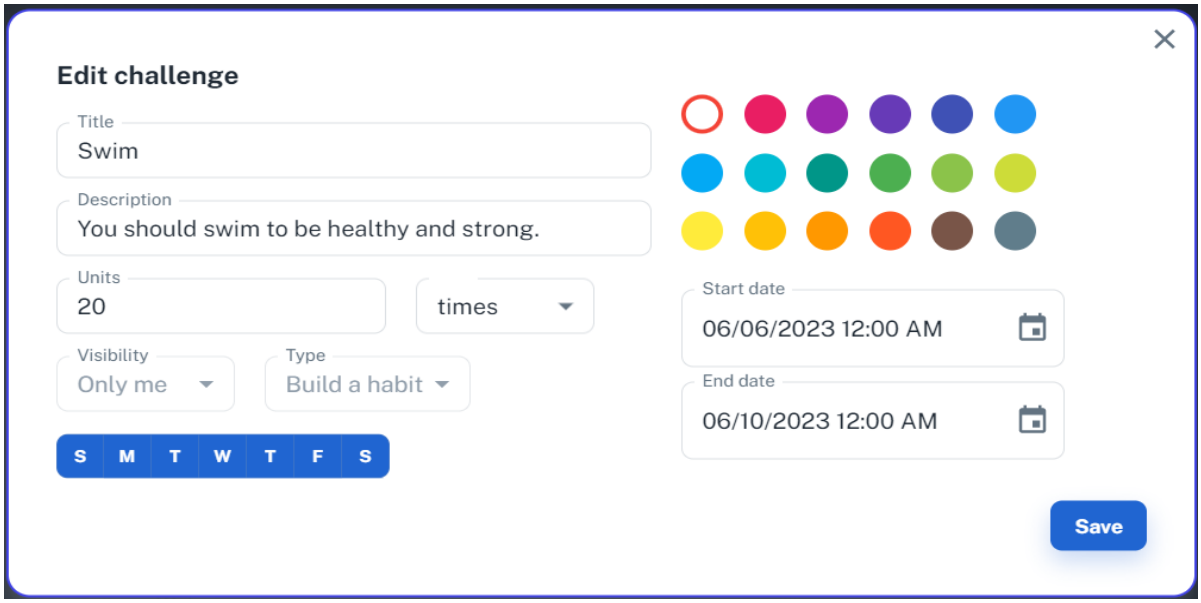

Рис. 4.13. Сторінка редагування челенджу

4.8. Перегляд челенджу

Цей компонент виконаний на основі компоненту редагування, але з неактивними полями, тому таким чином просто отримуємо дані челенджу, які можемо переглядати (див.рис.4.14.).

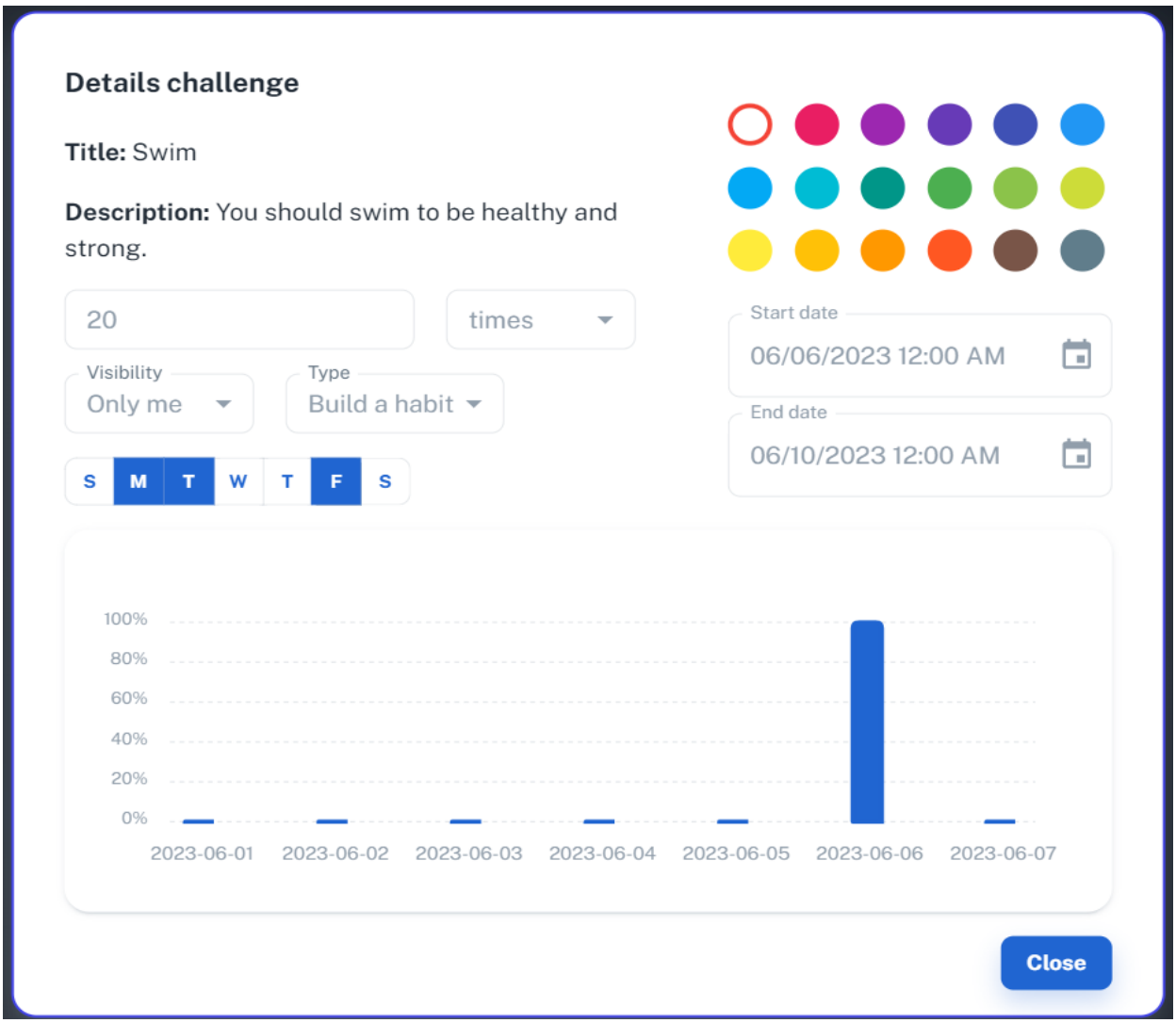

Рис. 4.14. Модальне вікно перегляду деталей челенджу

4.9. Видалення челенджу

Видалення челенджу відбувається при переході на пункт меню “Delete”, що знаходиться у картці. Для цього було створено компонент модального вікна “DeleteModal” (рис. 4.15).

Перелік використаних функцій та хуків:

- handleClickOpen() встановлює відкрите значення для відображення модального вікна.
- handleClose() закриває модальне вікно.
- handleDelete() видаляє обраний елемент, використовуючи useDeleteChallenge.

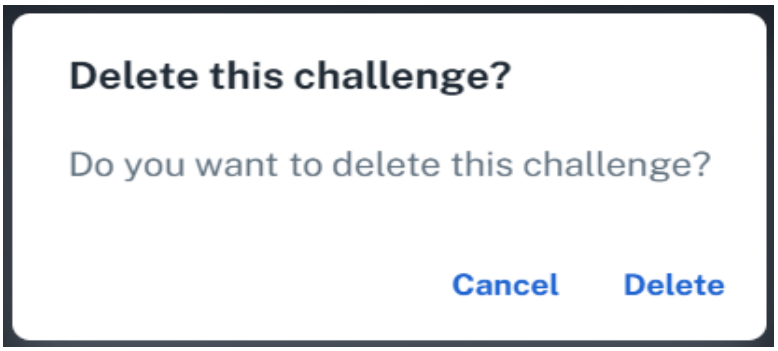

Рис. 4.15. Модальне вікно видалення

Код нижче містить функцію deleteChallengeMutationFn, що приймає ідентифікатор challengeId і повертає запит DELETE до сервера. Також містить useDeleteChallenge(), яка використовує useMutation з deleteChallengeMutationFn() і повертає об'єкт з deleteChallenge. useDeleteChallenge() дозволяє виклик мутації для видалення челенджу. Основна функція цього коду - створення зв'язку між клієнтською частиною та сервером.

```
import { useMutation } from 'react-query';
import instance from '../configurations/configurations';
import { SERVER_URL } from '../../constants/api/urls';

const deleteChallengeMutationFn = ({ challengeId }) => {
  return instance.delete(`${SERVER_URL}/Challenge/delete?challengeId=${challengeId}`, false).then((res) => res.data);
};

export const useDeleteChallenge = ({ onSuccess, onError }) => {
  const { mutate: deleteChallenge } = useMutation(deleteChallengeMutationFn, {
    mutationKey: 'deleteChallenge',
    onSuccess,
    onError,
  });
  return { deleteChallenge };
};
```

Лістинг 3. Код файлу useDeleteChallenge

4.10. Перегляд прогресу

Перш за все, варто зазначити, що користувач може здійснити перегляд прогресу як за окремим челенжом, так і загальний прогрес за найближчий період.

Перегляд загального прогресу відбувається в місці перегляду загальної кількості челенджів (рис. 4.16). Це можливо завдяки компоненту, який у стовпчасту діаграму отримує дані(у відсотках) про проходження завдань на щодень за датою. Показник “100%” означає те, що користувач виконав усі завдання на день.

Для розробки компоненту прогресу було використано код діаграм з шаблону Minimal UI та поміщено у створений компонент AppWebsiteTasks, що відображає графік за допомогою бібліотеки React ApexCharts.

Компонент AppWebsiteTasks приймає наступні пропси:

- title - заголовок компонента;
- subheader - підзаголовок компонента;
- chartData - дані для графіку;
- chartLabels - мітки для осі x графіку.

Цей компонент застосовується на сторінці Challenges, де виводиться перед списком челенджів.

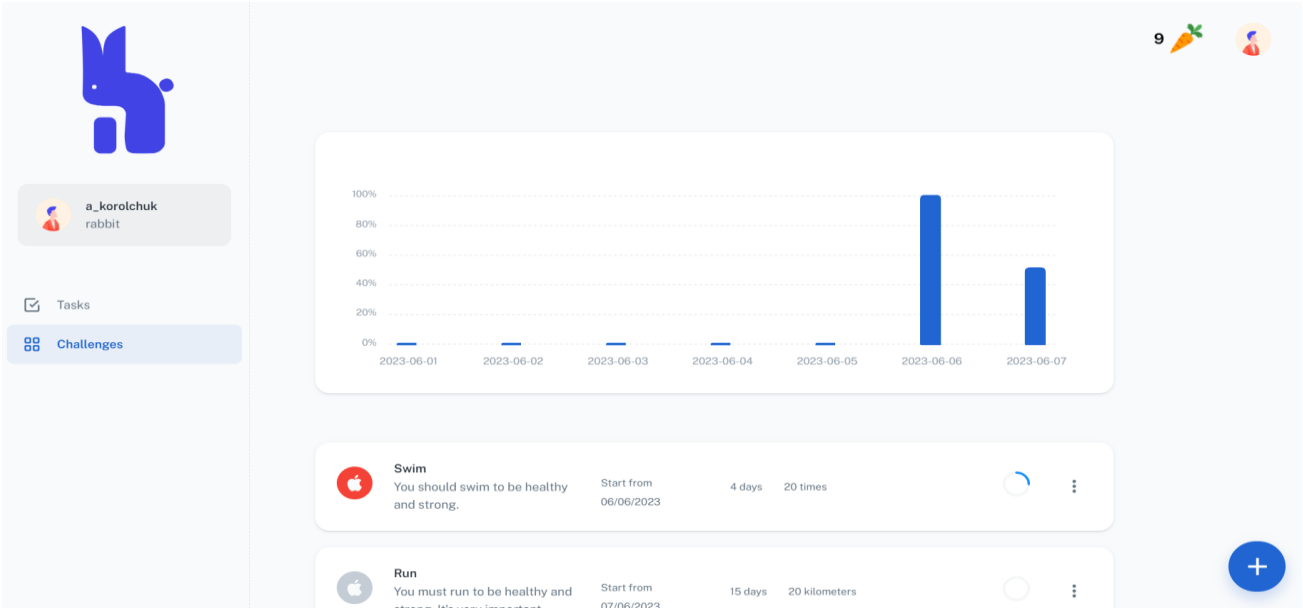

Рис. 4.16. Компонент перегляду загального прогресу за найближчий період

Перегляд прогресу за окремим челенджом відбувається при переході за пунктом меню “Details” за схожим механізмом, як і перегляд загального прогресу (рис. 4.17). При відкриванні форми з детальною інформацією, користувач може переглянути і прогрес завдяки компоненту, який у стовпчасту діаграму отримує дані(у відсотках) про проходження челенджу за датою. Показник “100%” означає те, що користувач виконав челендж того дня повністю.

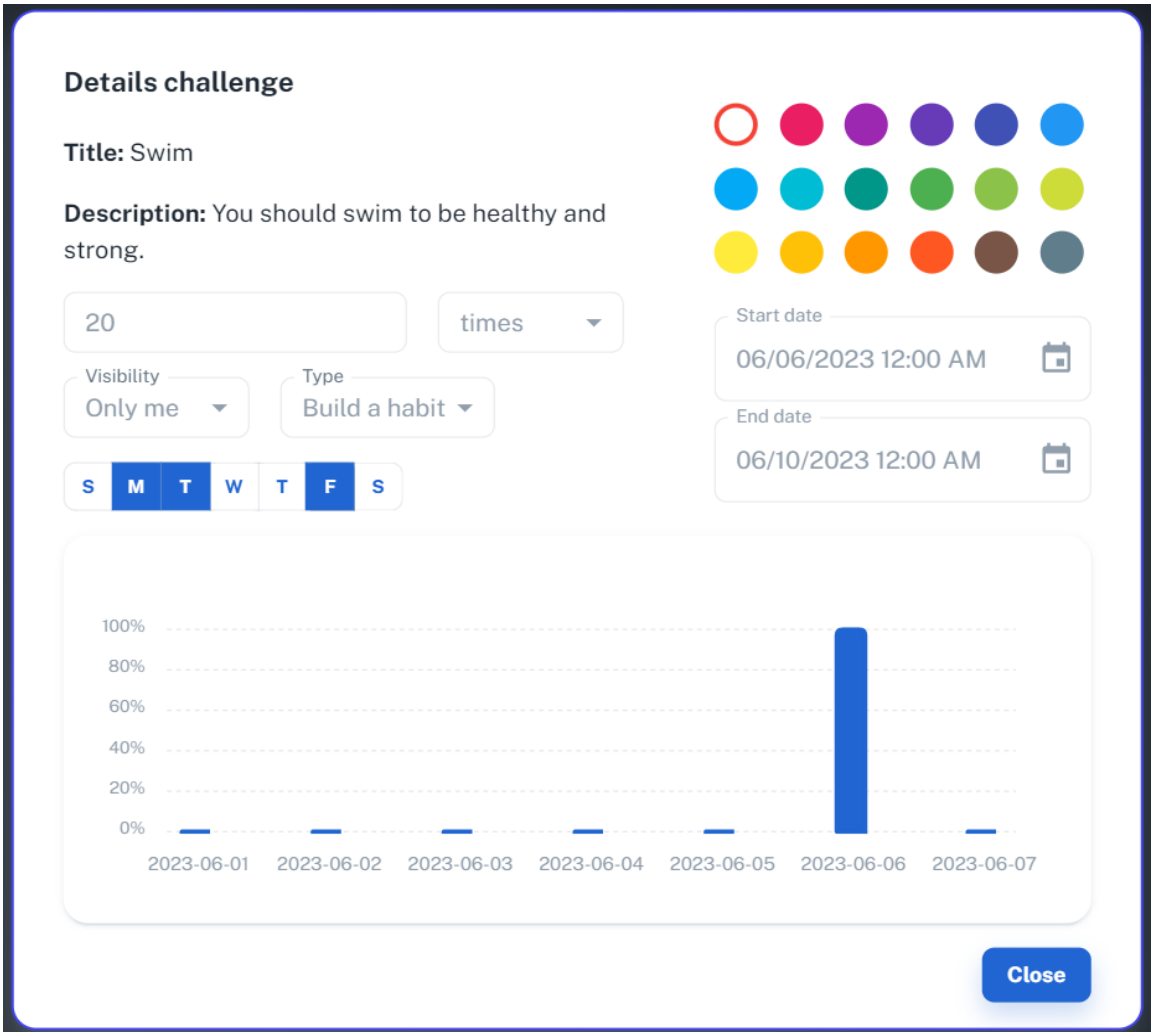

Рис. 4.17. Компонент перегляду прогресу за конкретним челенджом

4.11. Гейміфікація

Гейміфікація відбувається у два етапи: присвоєння балів (points) та присвоєння статусів.

Перш за все, нараховування балів за досягнення користувача (рис. 4.18). Це грає важливу роль у мотивуванні користувача використовувати застосунок та здобувати нові й нові звички або навпаки позбуватися їх. Система нарахування побудована на присвоєнні та втраті балів. Користувач може отримати бали за такі досягнення:

1 point - створення нового челенджу;

2 points - завершення завдання на день;

5 points - завершення челенджу;

При скасуванні прогресу бали віднімаються.

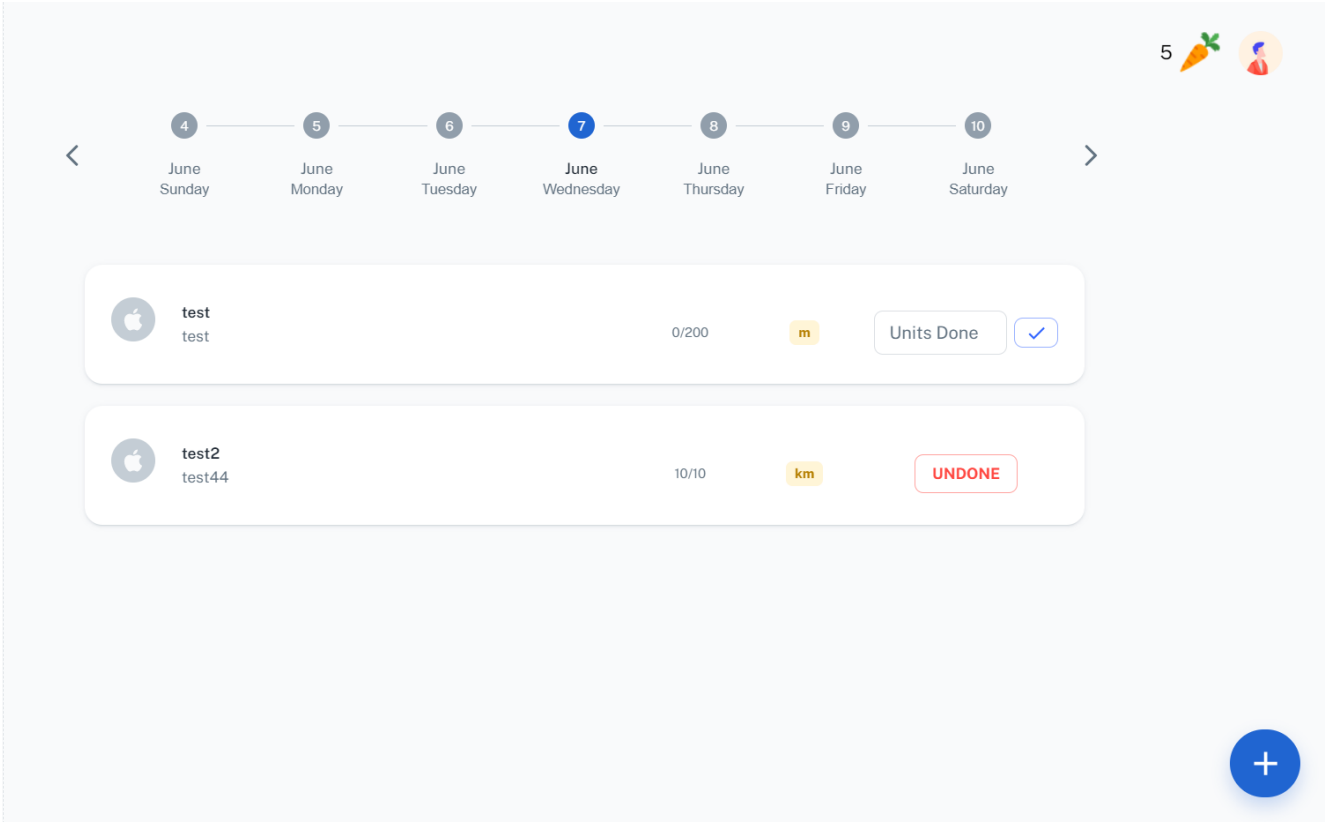

Рис. 4.18. Компонент перегляду балів користувача

У наступному етапі, в залежності від кількості балів за градацією, користувач отримує новий статус та фото профілю (аватарку) (рис. 4.18).

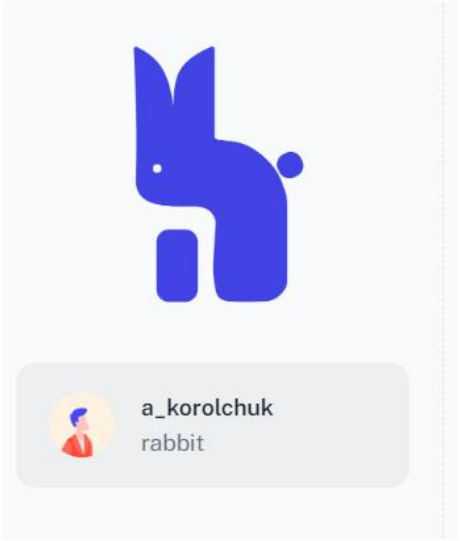

Рис. 4.18. Компонент перегляду статусу користувача

Висновки до розділу 4

У розділі 4 описану програмну реалізацію тренеру звичок «Habit Rabbit». Висвітлені основні етапи та елементи аутентифікації за допомогою access та refresh токенів. Для виконання цієї задачі було використано такі модулі, як react-redux, redux, redux-thunk, react-router-dom. Поетапно було виконано такі кроки:

1. Спершу було створено папку services та файли authentication.js, token.js та user.js.
2. Далі послідовно було створено папки ReduxActions та auth та файли index.js та types.js.
3. Після цього здійснювалася робота з Redusers, безпосередньо з файлом authReducer.js.
4. Взаємодія із сторінками реєстрації, аутентифікації.

Крім цього було здійснено опис основних компонентів, а саме закцентовано увагу на використанні хуків і функцій та ролі, яку вони виконували. Поетапно було виконано такі кроки:

1. Написання коду для модального вікна, за допомогою якого здійснюється створення челенджів;
2. Написання коду для сторінки виведення усіх поточних челенджів;
3. Написання коду для модального вікна, яке відповідає за видалення челенджів;
4. Написання коду для модального вікна, яке відповідає за редагування челенджів;
5. Написання коду для модального вікна, яке відповідає за перегляд челенджів;
6. Написання коду для сторінки виведення усіх завдань на день;
7. Написання коду для компоненту відслідковування прогресу за кожним челенжом;

8. Написання коду для компоненту відслідковування прогресу за всіма челенжами;
9. Написання коду для компоненту підрахунку балів з елементами гейміфікації.
10. Написання коду для компоненту присвоєння статусу.

Варто зазначити, що розроблено систему гейміфікації, задля покращення мотивації користувача якісно набувати звичку або позбуватись її. Вона полягає у підрахунку кількості зароблених балів та присвоєнні статусів на основі досягнень.

ВИСНОВОК

Під час виконання роботи, перш за все, було окреслене предметне середовище, зроблено опис технологій, які використовувалися у проекті: React (для клієнтської частини), та ASP .Net Core, MS SQL Server, Entity Framework - для серверної. У проекті використано підхід чистої архітектури, що допомагає відокремити залежності та ізолювати їх від бізнес-логіки програми і моделі домену. Цей підхід має кілька переваг, таких як незалежність від інтерфейсу користувача, баз даних, зовнішніх бібліотек, фреймворків і тестування. Для розробки клієнтської частини було використано бібліотеку ReactJS, що надає можливість створювати повторно використовувані компоненти, що можна вкладати в інші компоненти для створення складних програм із простих блоків. Механізм на основі віртуальної DOM, який використовує ReactJS, допомагає швидко й ефективно наповнювати HTML DOM даними. Використання ReactJS допомагає спростити розробку та обслуговування додатків.

Крім цього, було здійснено опис та порівняльний аналіз аналогів терекерів звичок, а саме Habitify, Coach.me, Tick Tick, із застосунком “Habit Rabbit” та складено технічне завдання даного веб-застосунку, а саме виконано опис функціональних та нефункціональних вимог до програмного продукту.

Далі було здійснено опис предметної області, що включає в себе опис понять та методу здобування звичок. Ще було розроблено Use Case діаграму, адже це ефективний метод передачі системної поведінки в термінах користувача шляхом визначення всієї зовнішньо видимої поведінки системи. За допомогою Use case діаграми розписано можливості доступу до функціоналу у ролі гостя (реєстрація та автентифікація) та користувача (перегляд персональної інформації, додавання челенджу, редагування челенджу, видалення челенджу, перегляд завдань на день та позначення прогресу, перегляд статистики челенджу).

Окрім цього було виконано опис програмного забезпечення, яке було обране для розробки системи. Це Visual Studio Code, для запуску серверної частини та внесення деяких правок - Rider, для роботи з API - Swagger, для контролю версій - Git, для розробки дизайну - Figma, а Material UI став фреймворком, з якого

використовувалися деякі готові React компоненти та стилі. При описі ПЗ було вказано на ряд переваг, завдяки яким було зроблено вибір на користь застосування цих продуктів при розробці трекеру звичок.

У завершальному розділі здійснено опис архітектури проекту, дизайну макетів сторінок та програмної реалізації основних функцій веб-застосунку. Значну частину опису складали хуки, які дозволяють використовувати стани та інші можливості React без застосування класів. У ході роботи було створено компоненти для створення, видалення, редагування та перегляду челенджів. Також для удосконалення процесу здобуття звички було розроблено компоненти для гейміфікації, що полягає у накопиченні балів за проходження челенджів та завдань на день, а також присвоєнні статусів за набрану кількість балів, що може вплинути на більшу вмотивованість користувача трекеру.

Отже, у ході виконання кваліфікаційної роботи було реалізовано клієнтську частину веб-застосунку для допомоги у формуванні та позбутті звичок з елементами гейміфікації, який слугуватиме хорошою підтримкою у процесі здобуття чи позбуття звичок. Варто зазначити, що трекер звички Habit-Rabbit не є досконалим та дещо поступається своїм конкурентам на даному етапі, бо деякі функції ще знаходяться в процесі розробки, а деякі заплановані у наступній версії, оскільки поточна версія має неповний функціонал та потребує доопрацювання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Стаття: Розробка односторінкових додатків SPA
URL:https://webcase.com.ua/uk/blog/razrabotka-odnostranichnyh-prilozhenij-spa-webcase/. (дата звернення: 11.03.2023 р.).
2. Article: Clean architecture-everything you need to know
URL: https://codilime.com/blog/clean-architecture/
(дата звернення: 11.03.2023 р.).
3. Article: ASP.NET Core Overview
URL:https://www.tutorialsteacher.com/core/aspnet-core-introduction
(дата звернення: 11.03.2023 р.).
4. Article: Pros and Cons of ReactJS
URL: https://www.javatpoint.com/pros-and-cons-of-react
(дата звернення: 11.03.2023 р.).
5. Article: The 12 Best Habit Tracking Apps in 2023
URL: https://collegeinfogeek.com/habit-tracker/
(дата звернення: 15.03.2023 р.).
6. Article: Why Visual Studio Code
URL: https://code.visualstudio.com/docs/editor/whyvscode
(дата звернення: 15.03.2023 р.).
7. Article: Rider
URL: https://www.jetbrains.com/rider/
(дата звернення: 15.03.2023 р.).
8. Article: Swagger
URL:https://www.techtarget.com/searchapparchitecture/definition/Swagger
(дата звернення: 15.03.2023 р.).
9. Article: What is Git? What benefits does Git offer?
URL:https://guide.quickscrum.com/git-guide/
(дата звернення: 15.03.2023 р.).
10. Article: What is Figma?

URL: https://www.nobledesktop.com/learn/figma/what-is-figma
(дата звернення: 15.03.2023 р.).

11. Article: What is Material UI?
URL: https://flatlogic.com/blog/what-is-material-ui/
(дата звернення: 15.03.2023 р.).

12. Article: What is Material UI?
URL: https://flatlogic.com/blog/what-is-material-ui/
(дата звернення: 15.03.2023 р.).

13. Лінк на репозиторій проекту:
URL: https://github.com/korolchuk111/habit-rabbit-react