ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:«Розробка Web-застосунку Mood journal трекеру емоцій та настрою для аналізу та самотерапії за допомогою технологій React, JS, NodeJS»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Анастасія ЗІНЧЕНКО
(підпис)

Виконала: здобувачка вищої освіти групи ПД-41

Анастасія ЗІНЧЕНКО

Керівник: Ірина ЗАМРІЙ
д.т.н., доцент

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«____» _____2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Зінченко Анастасії Євгеніївні

1. Тема кваліфікаційної роботи: «Розробка Web-застосунку Mood journal трекеру емоцій та настрою для аналізу та самотерапії за допомогою технологій React, JS, Node.js»
керівник кваліфікаційної роботи д.т.н., доцент, зав.кафедри ІПЗ Ірина Замрій,
затверджені наказом Державного університету інформаційно-комунікаційних технологійвід «27» лютого2024р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024р.

3. Вихідні дані до кваліфікаційної роботи: технічна література з питань, пов'язаних з розробкою програмного забезпечення, опис методів відстеження емоцій та настрою, технічна документація з описом бібліотек для відображення діаграм та технічна документація з використання геолокації у web-додатках.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної області.
2. Проектування застосунку для відстеження емоцій та настрою.

3. Програмна реалізація та опис функціонування застосунку для відстеження емоцій та настрою.

4. Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до додатку.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Даталогічна модель бази даних.
6. Карта сайту.
7. Екранні форми.
8. Апробація результатів дослідження

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Огляд існуючих трекерів емоцій та настрою	14.03-17.03.2024	
4	Проектування web-застосунку трекеру емоцій та настрою	18.03-24.03.2024	
5	Програмна реалізація застосунку	25.03-22.04.2024	
6	Тестування застосунку	23.04-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувачка вищої освіти _____ Анастасія ЗІНЧЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Ірина ЗАМРІЙ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 53 стор., 1 табл., 51 рис., 14 джерел.

Об'єкт дослідження – процес стеження за власними емоціями та закономірностями між ними.

Предмет дослідження – використання технологій React, JavaScript (JS) та Node.js для створення функціонального та ефективного web-інструменту для аналізу та самотерапії.

Мета роботи – збільшення методів для самоаналізу та самотерапії за допомогою web-застосунку з використанням бібліотеки React, мови програмування JS та платформи з відкритим кодом Node.js.

Для реалізації поставленої мети потрібно вирішити наступні завдання:

1. Проаналізувати вже існуючі застосунки, виявити їх переваги та недоліки, скласти таблицю порівняння.
2. На основі проаналізованих застосунків розробити функціональні та нефункціональні вимоги.
3. Дослідити інструменти розробки, розробити архітектуру та дизайн web-застосунка.
4. Розробити web-застосунок на основі обраних інструментів.
5. Протестувати web-застосунок.
6. Пройти апробацію на Науково-технічних конференціях.

Результат дослідження – wed-застосунок, який дозволить користувачеві більш детально дослідити взаємовплив різних факторів на психологічне здоров'я користувача.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми та її актуальність: Вже давно доведено, що емоції мають не аби який вплив не лише на здоров'я, а й на кар'єру, відносини з іншими людьми, мотивацію. Наші реації на різні фактори це не просто несуттєва зміна насторою – це емоційні стани, якими свідома людина має вміти керувати. Розвиваючи навичку дослідження власних емоцій ми стаємо більш уважними до себе та свого внутрішнього світу, починаємо розуміти справжні потреби, поведінку та цілі.

Об'єкт дослідження – процес стеження за власними емоціями та закономірностями між ними.

Предмет дослідження – використання технологій React, JavaScript (JS) та Node.js для створення функціонального та ефективного web-інструменту для аналізу та самотерапії.

Мета роботи – збільшення методів для самоаналізу та самотерапії за допомогою web-застосунку з використанням бібліотеки React, мови програмування JS та платформи з відкритим кодом Node.js.

Для реалізації поставленої мети потрібно вирішити наступні завдання:

1. Проаналізувати вже існуючі застосунки, виявити їх переваги та недоліки, скласти таблицю порівняння.
2. На основі проаналізованих застосунків розробити функціональні та нефункціональні вимоги.
3. Дослідити інструменти розробки, розробити архітектуру та дизайн web-застосунка.
4. Розробити web-застосунок на основі обраних інструментів.
5. Протестувати web-застосунок.
6. Пройти апробацію на Науково-технічних конференціях.

Результат дослідження – wed-застосунок, який дозволить користувачеві більш детально дослідити взаємовплив різних факторів на психологічне здоров’я користувача.

Практичне значення результатів: Даний web-застосунок передбачає використання у якості персонального інструменту для самоаналізу та самотерапії.

Для розробки використовуютья такі інструменти, як мова програмування JavaScript, бібліотека web-інтерфейсів React, платформа з відкритим кодом Node.js та бібліотека Knex.js.

Практична значущість результатів: Розроблений web-додаток надає можливість користувачам аналізувати дані, які можуть впливати на їх емоційний стан та виявляти закономірності.

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Актуальність трекеру емоцій та настрою

У сучасному світі важливо розуміти не лише свої базові потреби, обов'язки, а й власні почуття та емоції. «Емоційний інтелект» є важливою частиною розсудливої, розважливої людина, адже це важлива характеристика особистості, яка виражається в її ефективності розуміння власних емоції.

Існує багато різних способів самотерапії та самопізнання. Це можуть бути вільні щоденники та записи (Freewriting), медитації, щоденник вдячності, арт-терапії. Але досить поширеним та водночас простим є ведення щоденнику настрою. Він дозволяє відслідковувати закономірності та дізнаватися, що викликає ті чи інші емоції. Навіть якщо ви досить довго перебуваєте в одному настрої, щоденник настрою допоможе знайти мінімальні коливання та врівноважити ваше повсякденне життя. Люди, які припускають, що можуть змінити свої емоції та почуття, відчувають емоційне благополуччя як у короткій, так і у довгостроковій перспективах.

На даний момент mood journal набув автоматизації у різних проявах: готові шаблони у Notion або Google sheets, веб-додатки, мобільні застосунки. Але не всі вони мають достатньо функціоналу для якісного та швидкого аналізу, запису та відстеження емоцій та настрою.

Для вирішення функціональних та програмних недоліків було розроблено web-застосунок Mood journal. Задля точного визначення засобів і методів покращення застосунку було проведено дослідження існуючих додатків: MoodTracker, Journey.cloud, eMoods. Результати аналізу розглянутих додатків буде наведено у таблиці.

Головною задачею була розробка web-застосунку із додатковими функціями, що дозволять провести більш глибокий аналіз чинників, які

впливають на емоційний стан.

Перш за все, одними з найвідоміших засобів відображення даних для аналізу є стовпчасті діаграми та графіки. Вони дозволяють відразу побачити статистику за певний період. Важливо зазначити, що не менш корисними є кільцеві діаграми, які чудово відображають порівняння настроїв, або ж відображення загальних даних, таких як: середня кількість годин сну, що найбільше впливає на настрій, яку емоцію користувач відчуває найчастіше. Доцільним є розуміння який вплив має оточення та розташування, тому відстеження історії геолокацій було б корисною функцією в застосунку. Крім цього дії або звички також мають певний вплив на емоційний стан, тому функція постановки цілей та їх досягнення є важливою складовою самотерапії.

Для вирішення потреб, які не входять у загальний перелік даних аналізу, у застосунок було додано вільне введення нотаток з можливість застосування хештегів. У подальшому вони будуть використовуватися у пошуку записів, що значно покращить знаходження важливих записів та деталей настрою.

Цільовою аудиторією застосунку є люди, які прагнуть краще зрозуміти свої емоції та настрої, виявити закономірності та дізнатися, що впливає на них. Також ті, хто страждає від депресії, тривожності та інших психологічних розладів. Їх лікування проходитиме значно легше, якщо вони матимуть можливість відслідковувати прийом ліків, зміни настрою та завантажувати ці дані для подальшого їх аналізу зі спеціалістом. Загалом цей застосунок не матиме обмежень, адже дослідження та самопізнання є доступним для будь-якого користувача, який прагне контролю та розуміння емоцій та настрою.

1.2 MoodTracker

Web-застосунок MoodTracker має простий інтерфейс, який дозволяє відстежувати важливі показники здоров’я[1]. Застосунок також має функцію відстеження прийому медикаментів. Однією з його особливостей є можливість налаштувати нагадування на електронну пошту. Також у ньому присутній форум

для пошуку однодумців та підтримки. Форум має розподіл по групах, які в свою чергу діляться на теми. Користувач має можливість поставити собі одну мету, яка періодично буде з’являтися при сході у свій обліковий запис. Статистика даних відображається у стовпчастих діаграмах або графах. Екранні форми застосунку MoodTracker наведено на рисунку 1.1 та рисунку 1.2

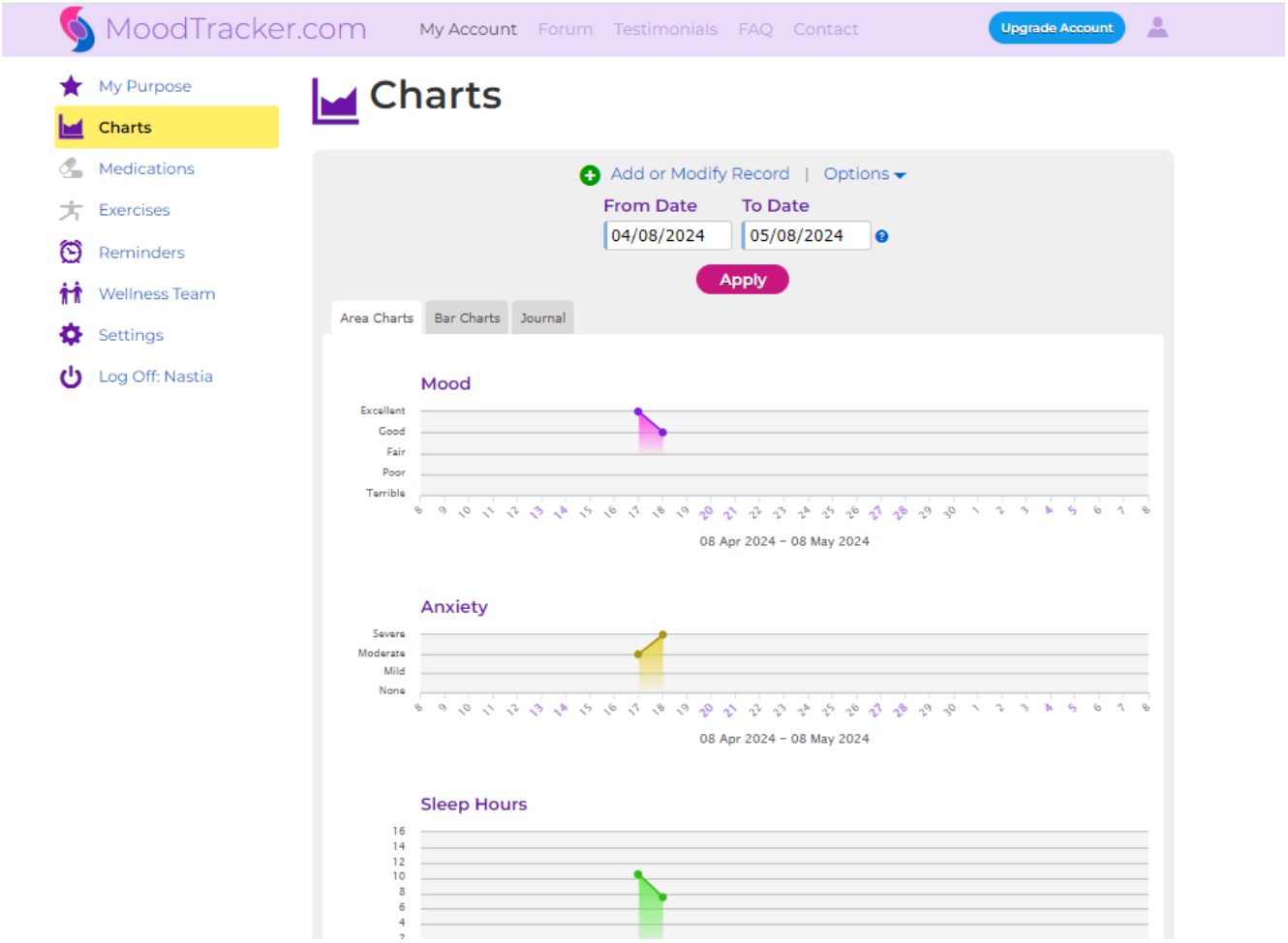


Рис. 1.1 Приклад сторінки з діаграмами застосунку MoodTracker

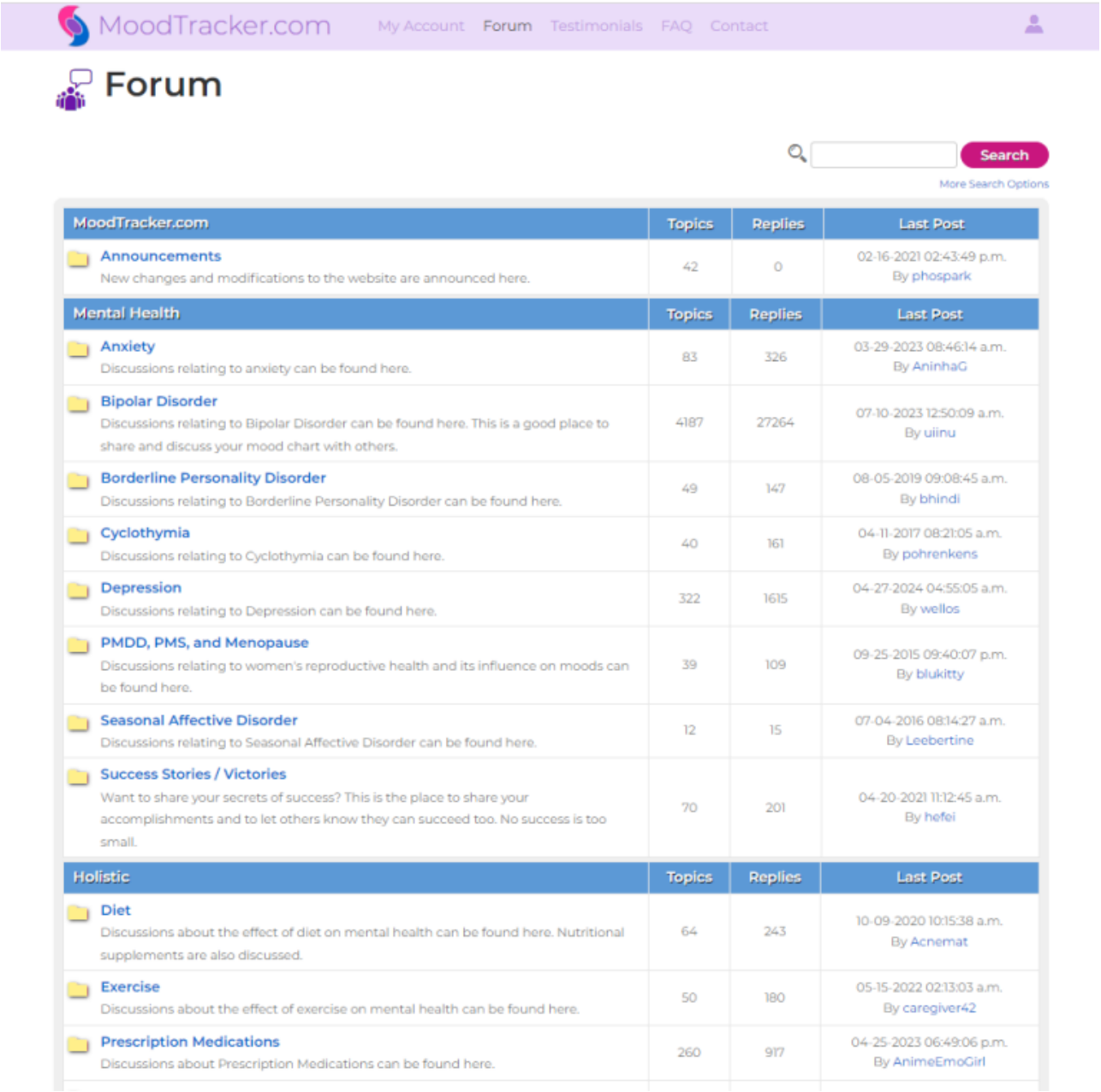


Рис. 1.2 Приклад форму застосунку MoodTracker

До недоліків даного застосунку можна віднести відсутність пошуку по записам, тому для знаходження потрібних даних доведеться витратити деякий час. Також тут відсутні можливості кастомізації, тому користувачеві дозволено відслідковувати лише базові параметри. Ще одним недоліком є те, що інтерфейс платформи є дещо застарілим і через це методи аналізу, запропоновані в MoodTracker, не є достатніми.

1.3 Journey.cloud

Застосунок Journey.cloud – це щоденник, який дозволяє вести свої записи з будь-якої платформи[2]. Його функціонал дозволяє користувач вести записи думок, відслідковувати емоції та ділитися фотографіями, спогадами з іншими користувачами. Для кращого розуміння настрою та емоцій є можливість використовуватидіаграму настрою. Застосунок також містить корисні програми та поради.

Journey.cloud має різноманітні інструменти редагування, як от стилізація шрифта, таблиць та списків. Всі вони доступні при створенні нового запису на вкладці «Шкала часу». Для користувачів також доступні готові шаблони нотаток, які за власним бажанням можна кастомізувати.

Наразі бета-тестування проходить їх нова функція Journey Odyssey AI – помічник у роботі з записами. Він може відповісти на питання пов'язані з вашими нотатками. Наприклад :«Скільки записів я написав у 2023 році? », «Підсумуйте, що я зробив минулого Різдва». Екранні форми застосунку Journey.cloud наведено на рисунку 1.3 та рисунку 1.4

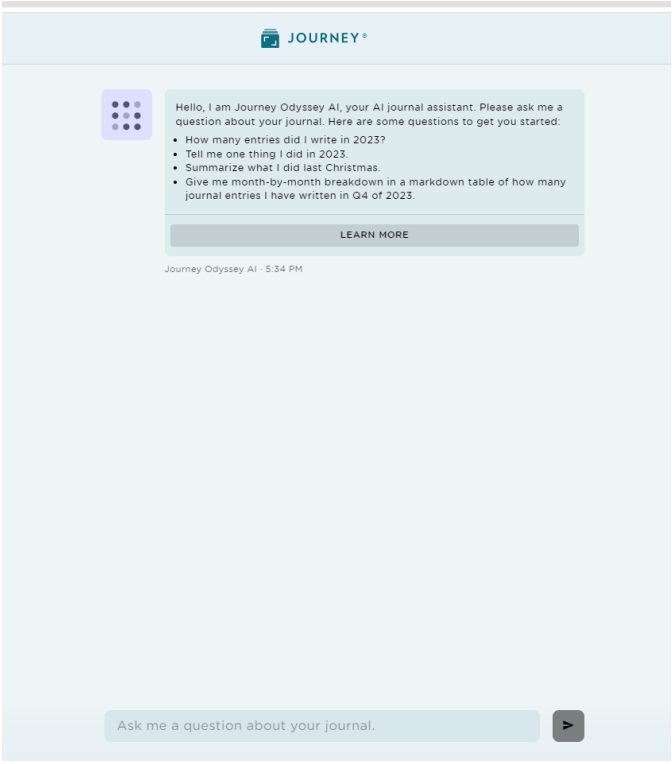


Рис. 1.3 Приклад сторінки з Journey Odyssey AI застосунку Journey.cloud

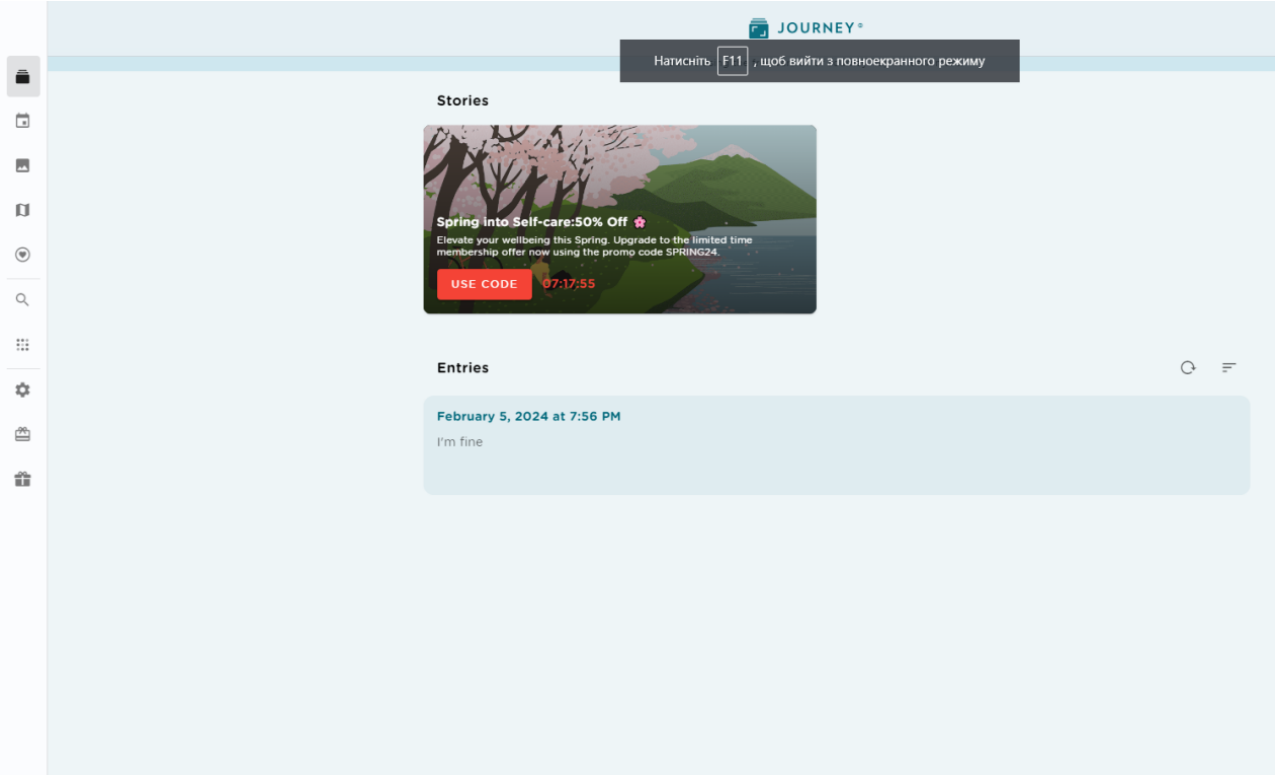


Рис. 1.4 Приклад сторінки з записами застосунку Journey.cloud

До недоліків застосунку можна віднести те, що в разі шифрування даних «end-to-end» при втраті паролю дані не можливо буде відновити. Також при такому шифруванні користувачі не зможуть використовувати Journey Odyssey AI, оскільки для швидкого пошуку цей інструмент використовує індексацію. Штучний інтелект цього застосунку також не дозволяє робити більше 10 запитань для безкоштовної та преміум версій, а 50 запитань доступні лише для власників абонементу. Враховуючи ці фактори, використання Journey Odyssey AI для пошуку записів є не найзручнішим інструментом.

1.4 eMoods

Застосунок eMoods перш за все призначений для людей які мають біполярний розлад, але його можуть використовувати також і інші користувачі

для моніторингу свого психологічного здоров'я[3]. Основними функціями web-застосунку є:

– відстеження настрою;
– відстеження симптомів(безсоння, психотичні симптоми, лікарські ефекти та інше);
– звіти у графіках з можливістю їх завантажити

Інтерфейс є досить простим, тому користувач має змогу почати працювати з eMoods без додаткового навчання. Крім цього ця платформа містить посилання на онлайн ресурси допомоги в разі загрози життю користувача або когось з його знайомих. До цих ресурсів також відноситься форум eMoods. Він містить статті на корисні теми з можливість залишити коментар та знайти однодумців. Екранні форми застосунку eMoods наведено на рисунку 1.5 та рисунку 1.6

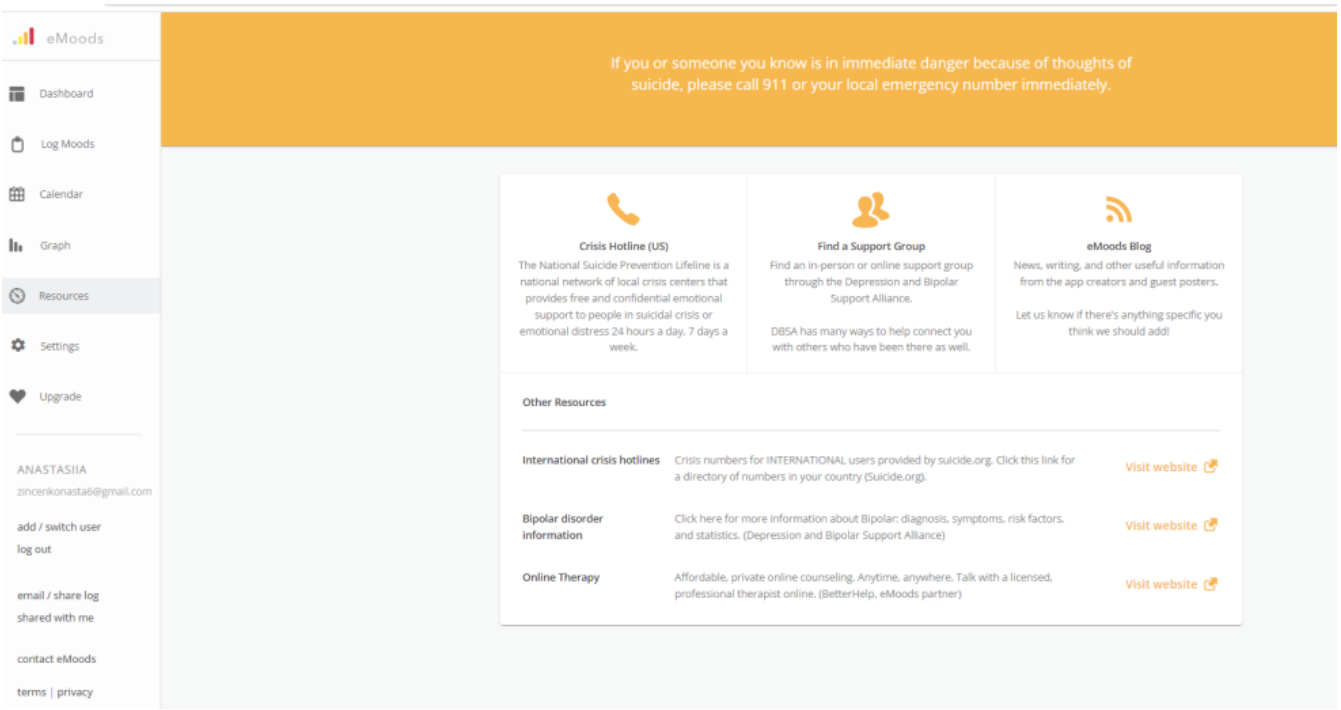


Рис. 1.5 Приклад сторінки з ресурсами застосунку eMoods

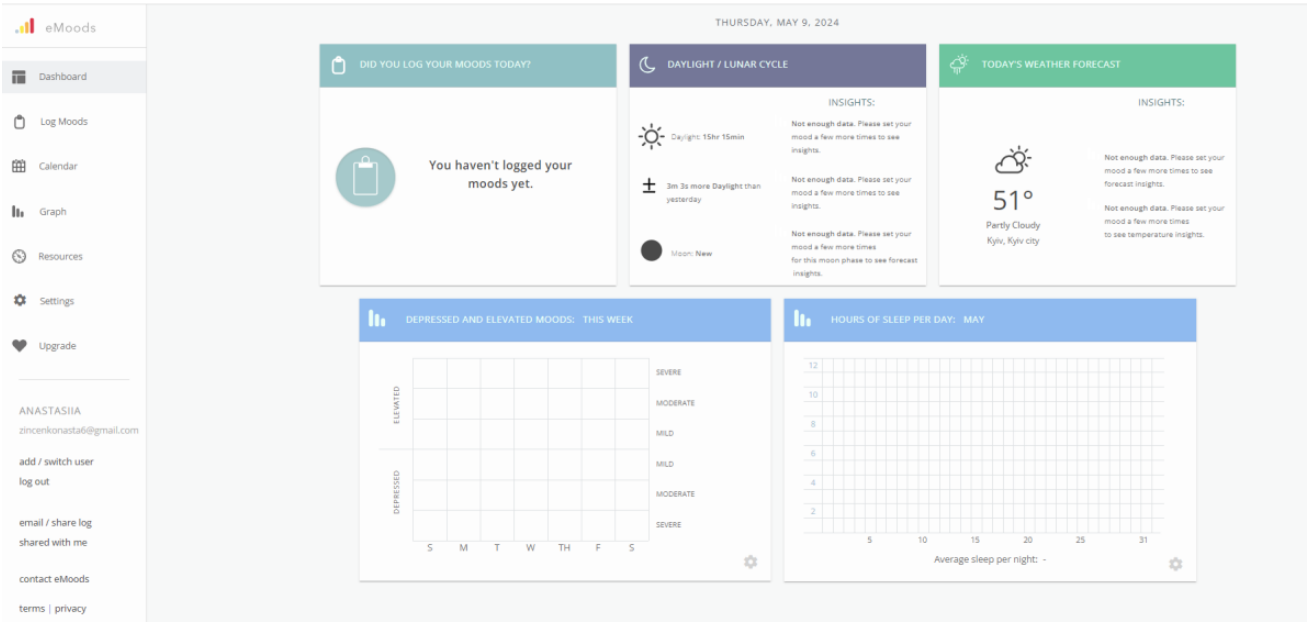


Рис. 1.6 Приклад панелі приладів застосунку eMoods

До недоліків платформи можна віднести те, що вона орієнтована перш за все на користувачів зі Сполучених Штатів Америки, тому така, на перших погляд, корисна функція, як онлайн ресурси допомоги не є працюючою для більшості українців. Також тут відсутня валідація введених даних у нотатках, тому користувач може зберігати як порожні рядки, так і рядки з пробілами на початку або кінці речення.

1.4 Таблиця порівняння аналогів

Таблиця 1.1

Порівняння існуючих програм-аналогів

Показник	**MoodTracker**	**Journey.cloud**	**eMoods**	**Mood journal**
Створення та редагування щоденних нотаток	+	+	+	+

Продовження таблиці 1.1

Порівняння існуючих програм-аналогів

Показник	**MoodTracker**	**Journey.cloud**	**eMoods**	**Mood journal**
Налаштування переліку емоцій, їх груп та кольорів	-	-	+	+
Статистика. Відображення графіку настрою та емоцій, їх підрахунок	+	-	-	+
Відображення даних у вигляді календаря	-	+	+	-
Можливість імпортування даних	-	-	+	+
Можливість додавати перелік ліків, вітамінів які приймав користувач	+(лише платна версія)	-	+	+
Можливість ставити цілі	-	-	-	+
Можливість структурувати нотатки за типами або темами	-	-	-	+
Прив’язка настроїв до геолокації	-	+(лише платна версія)	-	+

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Технічне завдання

Розробка програмного забезпечення для відстеження емоцій та настрою у вигляді веб-застосунку. Клієнтська частина повинна бути розроблена за допомогою бібліотеки React, керування стану застосунку відбуватиметься за допомогою менеджера станів Redux, інтерфейс користувача має бути розроблений з використанням бібліотеки Material UI, для управління навігації та маршрутів веб-сторінки застосовувати React Router. Серверна частина має бути розроблена з використанням Node.js, Knex.js для взаємодії з базою даних та Express для маршрутизації та обробки запитів.

У якості середовища розробки використати Visual Studio Code, для форматування коду інструмент Prettier та ESLint для контролю стилю коду на клієнтській частині. Git Hub буде використано для системи контролю версій.

2.2 Вимоги до програмного забезпечення

Дослідивши популярні трекери настрою такі як MoodTracker, Journey.cloud, eMood було сформовано основні функціональні вимоги, яким має відповідати застосунок, а саме:

1. Реєстрація та авторизація користувача, перевірка введених даних на валідність.
2. Створення та редагування щоденних записів з обов’язковим обранням настрою, валідація цих даних.
3. Відображення даних за період у вигляді діаграм та графіків.
4. Завантаження діаграм та графіків.
5. Зміна теми та мови застосунку.
6. Розрахунок підсумкових даних за весь час використання застосунку.

7. Можливість додавати, видаляти та змінювати геолокацію.

Нефункціональними вимогами є ті, які впливають на ефективність роботи застосунку та дозволяють оцінити його якість. До даних вимог відносяться:

1. Конфіденційність. Застосунок повинен бути захищений від витоку конфіденційних даних.
2. Доступність. Застосунок має бути створено за підходом Semantic HTML, що забезпечить доступність для користувачів з обмеженими можливостями.
3. Сумісність. Застосунок має бути доступним на різних пристроях та веб-браузерах.
4. Продуктивність. Застосунок повинен бути швидким та мати мінімальний час завантаження сторінок.
5. Масштабованість. Застосунок має мати можливість масштабуватися за потреби та ефективно працювати при збільшенні кількості користувачів.

2.3 Опис архітектури застосунку

Web-застосунок Mood journal передбачає клієнт-серверну модель архітектури.

Клієнтська частина - це розроблений інтерфейс користувача, який дозволяє взаємодіяти користувачеві з додатком. Ієрархію компонентів React зображено на рисунку 2.1

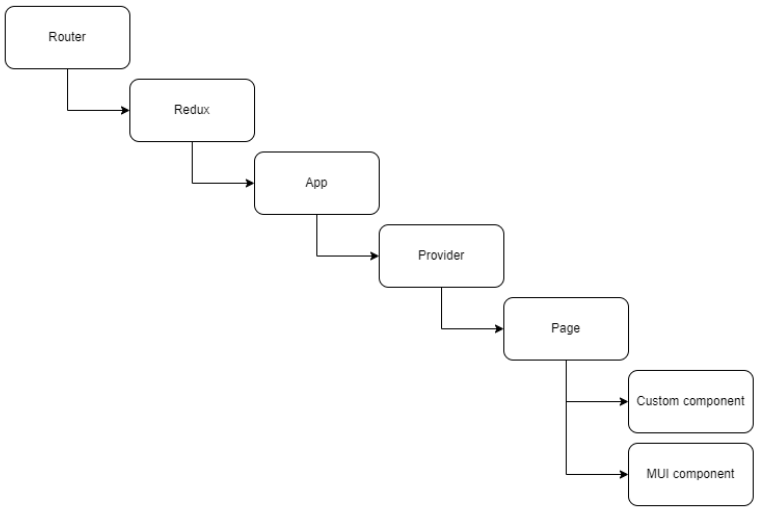


Рис. 2.1 Ієрархія компонентів React застосунку Mood journal

Верхнім рівнем ієрархіє є Router, який відповідає за маршрутизацію в застосунку та визначає який компонент має відображатися на основі поточного URL. Для його реалізації було використано бібліотеку React Router, яка дозволяє створювати односторінкові вебзастоснуки, де контент змінюється без перезавантаження сторінки. Настпуним рівнем є Redux, який відповідає за управління станами застосунку та є єдиним джерелом істини. Компонент App є головним у застосунку, адже містить структуру та логіку інтерфейсу користувача. Настпуний рівень ієрахніє це Provider, який розподіляється на MainProvider, StyleProvider та LocalProvider. MainProvider - це компонент, який забезпечує основні функції застосунку і саме він містить у собі два інші провайдери. StyleProvider відповідає за стилізацію застосунку та містить логіку темізації компонентів MUI за допомогою ThemeProvider та CssBaseline, який відповідає за нормалізацію стилів. LocalProvider відповідає за локалізацію застосунку та включає в себе логіку IntlProvider, який дозволяє налашувати поточний локаль та набір перекладів.Page відображає вміст конкретної сторінки. Він може включати в себе різні компоненти, такі як Custom component та MUI component. Custom component стилізуються на основі вже існуючих компонентів Material UI за допомогою утиліти styled(), дозволяючи перевикористовувати їх, або пропси sx, яка належиться до one-off customization, тобто швидкої зміни стилів одного екзепляру.

Серверну частину було побудовано на основі патерну Model-View-Controller, який дозволяє ефективно організувати логіку та доступ до даних веб-застосунку. Діаграму представлено на рисунку 2.2

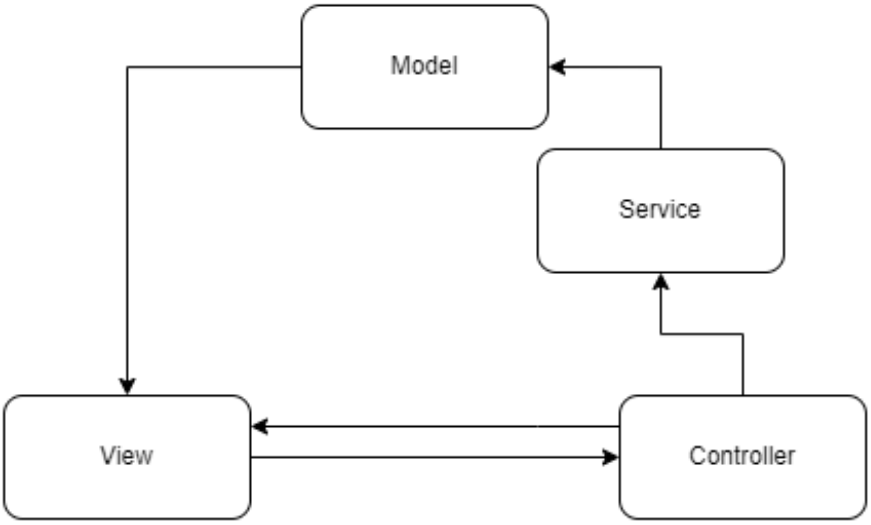


Рис. 2.2 Діаграма взаємодії MVC застосунку Mood journal

Controller відповідає за обробку HTTP запитів та керує потоком даних. Він викликає медоти з Service, який є прошарком між контролерами та моделлю, дозволяючи виокремити логіку доступу до даних від контролерів. Це дозволяє перевикористовувати код та полегшує обробку помилок. Model представляє структуру та логіку бази даних. У Controller відбувається взаємодія з нею, виконуючи операції читання, запису, оновлення та видалення. View представляє з себе клієнтську частину, яка відображає дані, отримані від сервера.

2.4 Діаграма використання

Діаграма використання – це вид діаграми, який дозвляє вказувати на очікувану поведінку застосунку. Вона допомагає проектувати систему з точки зору кінцевого користувача, який використовує її для досягнення своїх цілей. Дігграму використання веб- застосунку Mood journal зображено на рисунку 2.3

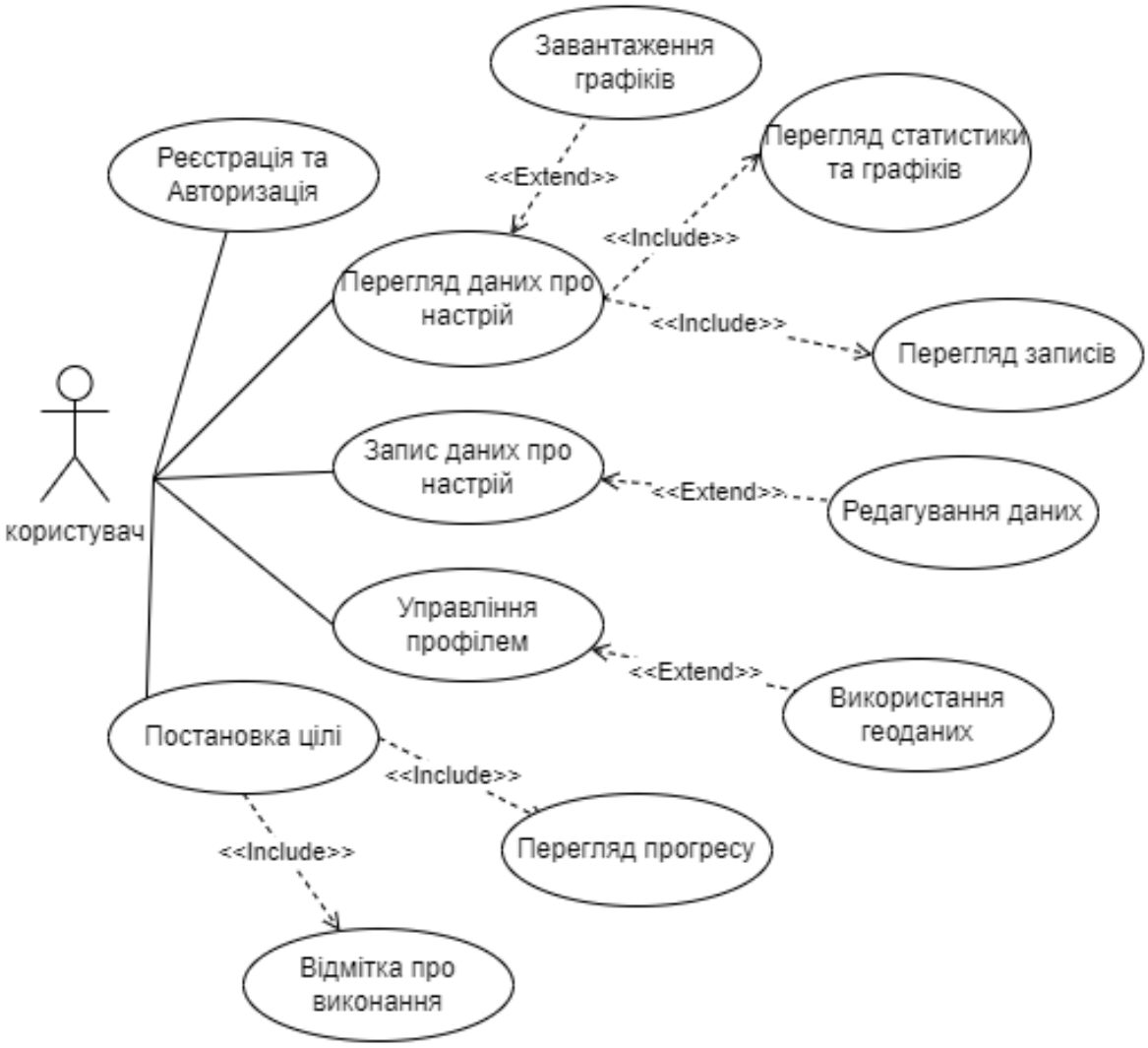


Рис. 2.3 Діаграма використання застосунку Mood journal

На даній діаграмі актором є користувач сайту.Випадки використання включають такий перелік дій: “Реєстрація та Авторизація”, “Запис даних про настрої”, який розширює “Редагування даних”, “Управління профілем”, “Перегляд даних про настрій”, який автоматично включає використання “Перегляд статистики та графіків”, “Перегляд записів” та його розширює “Завантаження графіків”, “Постановка цілі””, який автоматично включає використання “Перегляду прогресу” та “Відмітка про виконання”.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Інструменти і засоби розробки

3.1.1 Мова програмування JavaScript

JavaScript — це мова програмування, яка дозволяє реалізувати низку складних рішень у веб-документах[4]. Завдяки сучасним двигунам, таким як V8 для Google Chrome, SpiderMonkey для Firefox, Chakra для Microsoft Edge та JavaScriptCore для Apple обробка JavaScript-коду має високу швидкість виконання скриптів. Але також є небраузерні середовища які застосовують JavaScript як основну мову: Node.js, Apache CouchDB та Adobe Acrobat. Стандартами для JavaScript є специфікація мови ECMAScript (ECMA-262) і специфікація API інтернаціоналізації ECMAScript (ECMA-402).

Особливості та переваги мови JavaScript:

- Підтримка динамічної типізації. Це означає, що тип змінної визначається на основі значення, яке вона містить. Достатньо використовувати var, let або const, щоб оголосити змінну, не турбуючись про її тип.
- Підтримка об’єктно-орієнтованого програмування. У JavaScript підтримуються два важливі принципи ООП - інкапсуляція , успадкування та поліморфізм. Поліморфізм у JavaScript проявляється у здатності різних об’єктів відповідати на виклик методу різними способами. Інкапсуляція – це концепція, яка дозволяє зберігати внутрішні деталі об’єкта прихованими та відкривати лише публічний інтерфейс для взаємодії з об’єктом. А успадкування дозволяє створювати нові класи, які успадковують властивості та методи від існуючих класів[5].
- Функціональний підхід програмування. Функції підтримуються, як об’єкти першого класу, а об’єкти можна створювати за допомогою функцій конструктора і у результаті вони представлятимуть унікальний тип об’єкту.
- Інтерпретація. Це означає, що написаний JavaScript код буде оброблятися рядок за рядком.

- Асинхронні функції та проміси. Підтримка асинхронного програмування дозволяє надсилати асинхронні запити та значно скорочує їх час обробки.

3.1.2 React

React - це бібліотека JavaScript, яка була створена у 2013 році компанією Facebook і за цей час набула не аби якої популярності[6]. Вона дозволяє створювати складні інтерактивні інтерфейси за допомогою компонентів, які можна перевикористовувати. Компоненти бувають функціональні або класові і містять у собі логіку та стан.

Переваги React:

- JavaScript XML. Це розширення допомагає застосовувати HTML-подібний код у файлах з розширенням .js. Це дозволяє швидко створювати компоненти зі зрозумілим синтаксисом.
- Дані передається від батьківських компонентів до дочірніх у вигляді props, що дозволяє передбачувати їхню поведінку, а завдяки memo(для функціональних компонентів) та shouldComponentUpdate (для класових компонентів) дочірні не будуть оновлюватися зайвий раз.
- Віртуальний DOM. При зміні стану React робить копію реального DOM, оновлює її та робить порівняння з попередньою версією. Такий підхід дозволяє мінімізувати зайві оновлення та покращує продуктивність застосунку.

Для того, щоб максимально продуктивно використовувати React часто застосовуються додаткові бібліотеки. Найвідомішими є Redux та React Router.

Redux – це state менеджер, який дозволяє керувати станом застосунку, створена Даном Абрамовим та Ендрю Кларком у 2015 році.[7]

Концепції Redux:

- Єдине джерело правди. Стан застосунку зберігається у одному об’єкті, який має назву “store” і такий підхід дозволяє легко керувати станом та здійснювати глобальні зміні.

- Незмінний стан. Це означає, що він доступний лише для читання і може змінюватися лише за допомогою дій actions.
- Чисті функції. Вони гарантують, що вхідні дані завжди матимуть однаковий результат без побічних ефектів.

Redux також дозволяє зручно масштабувати проект, забезпечуючи чисту архітектуру. Також він дозволяє легко оптимізувати код, адже не має потреби виносити стан до найближчого батьківського компоненту і прокидати ці дані у вигляді пропсів, а достатньо їх винести у store і в разу потреби викликати за допомогою хуку useSelector.

React Router – бібліотека, яка керує станом URL та динамічно змінює контент компонентів на основі маршруту.[8]

Концепції React Router:

- Динамічність. Можливо створювати маршрути з параметрами, які є можливістю змінювати та таким чином впливати на URL.
- Programmatic Navigation. Використовуючи хуки useHistory або useNavigate можливо керувати навігацією через логіку застосунку.
- Вкладеність маршрутів. Для більш складних додатків є можливість застосувати вкладені маршрути, що робить їх більше гнучкими для використання.

3.1.3 Apexcharts

Apexcharts – це бібліотека, яка дозволяє створювати графік та діаграми для таких технологій, як Vue, React, Angular [9]. Простота у використанні та сучасний дизайн дозволяють ефектно візуалізовувати дані у користувацькому інтерфейсі. До основних можливостей бібліотеки належать:

- Велика варіативність графіків та діаграм. Apexcharts підтримує різні їх типи, а саме лінійні графіки, стовпчасті діаграми, кругові діаграми, теплові карти, радарні діаграми, графіки розсіювання та інші. Це дозволяє використовувати їх у різних проектах або різних візуалах.

- Інтерактивність. За замовчуванням діаграми та графіки містять різні варіанти взаємодії: кліки, наведення, масштабування та інші. Це забезпечує позитивний користувацький досвід та покращує взаємодії з даними.
- Адаптивність. Графіки та діаграми є повністю адаптивними, що значно полегшує розробку та дозволяє коректно відображати дані не залежно від екрану пристрою користувача.
- Кастомізація і налаштування. Ця бібліотека надає різноманітні можливості для конфігурації починаючи від зміни шрифтів і закінчуючи анімаціями.

Така гнучкість та простота у використанні без додаткових знань JavaScript або конкретного фреймворку робить Apexcharts чудовим інструментом для розробників, а привабливий результат для кінцевого користувача задовольняє будь-які його потреби в плані візуалізації даних.

```
var options = {
  series: [{
    name: "Desktops",
    data: [10, 41, 35, 51, 49, 62, 69, 91, 148]
}],
  chart: {
  height: 350,
  type: 'line',
  zoom: {
    enabled: false
  }
},
dataLabels: {
  enabled: false
},
stroke: {
  curve: 'straight'
},
title: {
  text: 'Product Trends by Month',
  align: 'left'
},
grid: {
  row: {
    colors: ['#f3f3f3', 'transparent'],
    opacity: 0.5
  },
},
xaxis: {
  categories: ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep'],
}
};

var chart = new ApexCharts(document.querySelector("#chart"), options);
chart.render();
```

Рис. 3.1 Приклад конфігурації базового лінійного графіку

3.1.4 Format.JS

Format.JS – це бібліотека JavaScript, яка застосовується для форматування даних та локалізації додатків[10]. Основним і найбільш уживаним компонентом якої є бібліотека Intl, яка дозволяє форматувати числа, дати та час, валюти і працювати з мовами. Для інтеграції у React є окрема технологія - React-Intl. Для інтеграції у застосунок потрібно спочатку огорнути головний компонент у “IntlProvider” та створити об’єкт з переліком мов та перекладів до них. Дані через пропсу local у це провайдер буде передаватися мова, яка і буде визначати мову застосунку.

```
 import React from 'react';
import { IntlProvider, FormattedMessage } from 'react-intl';
import { messages } from './messages';

const messages = {
    en: {
        welcome: "Welcome to our website!",
        description: "This is a sample description."
    },
    fr: {
        welcome: "Bienvenue sur notre site Web!",
        description: "Ceci est une description d'exemple."
    }
};

function App({ locale }) {
    return (
        <IntlProvider locale={locale} messages={messages[locale]}>
            <div>
                <h1><FormattedMessage id="welcome" defaultMessage="Welcome!" /></h1>
                <p><FormattedMessage id="description" defaultMessage="This is a sample description."
                    /></p>
            </div>
        </IntlProvider>
    );
}

export default App;

```

Рис. 3.2 Приклад базової локалізації застосунку

Format.JS надає чудові інструменти для міжнародного форматування та локалізації додатків. Таких підхід дозволяє отримати кращий користувацький досвід та адаптивний застосунок.

3.1.5 Material UI

Material UI – це популярна бібліотека для React проектів, яка надає доступ до використання компонентів, побудованих за принципом Material Design [11]. Її використання дозволяє створювати сучасні та зручні дизайни.

MUI містить налаштування теми застосунку, що забезпечує гнучкість у будь-якому проекті. Компоненти є повністю адаптивними, завдяки чому інтерфейс буде добре виглядати на різних пристроях.

Стилізація не обмежується налаштуванням теми, для створення власних елементів та компонентів, які будуть базуватися на принципах Material Design, доступна додаткова бібліотека Joy UI з відкритим кодом. Для ефективної кастомізації та створення індивідуальних проектів можна застосовувати MUI

System – набір утиліт CSS, який надає доступ до загальних компонентів по типу Box і Container і налаштовувати їх за допомогою пропси sx. Така функція корисна у ситуаціях коли потрібно створити не громіздкі компоненти без надлишкових констант.

Material UI є чудовим інструментом, який надає доступ до готових компонентів, застосування яких дозволить створити якісний дизайн сайту.

3.1.6 Node.js

Node.js – це кросплатформене середовище побудоване на основі JavaScript, яке дозволяє використовувати код для мережевих застосунків[12]. Він є популярним завдяки подвійно-орієнтованій моделі та архітектурі вводу-виводу, яка є неблокуючою. Таку функцію забезпечує Event Loop(цикл подій), який асинхронно обробляє події. Організація коду відбувається завдяки модулям від CommonJS, що дозволяє повторно використовувати код у різних частинах проєкту. Розширити функціонал можливо завдяки менеджеру пакетів npm.

Переваги Node.js:

- Швидкість. Сучасний рушій V8 оптимізовує та полегшує роботу Node.js-коду та забезпечує ефективне управління пам’яттю.
- Варіативність застосування. Node.js підходить для різних типів додатків: web-сервери, мікросервери, RTA додатки (ігри, чати), командний рядок.
- Масштабованість. Проєкти, створені за допомогою Node.js, забезпечують можливість ефективної обробки тисячі одночасних з’єднань зі зростаючою навантаженістю.

Виходячи з перелічених переваг, можна з точністю сказати, що даний інструмент дозволить створити надійний і легко масштабований проєкт, завдяки свої архітектурі та підтримці великій кількості додаткових модулів.

3.1.7 Knex.js

Knex.js – це SQL query-білдер для Node.js, який дозволяє працювати з різними реляційними базами даних[13]. Він підтримує роботу з популярними

базами даних: PostgreSQL, MySQL, SQLite, Oracle та Microsoft SQL Server. Однією з особливостей цієї бібліотеки полягає у тому, що запити пишуться за допомогою JavaScript, що дозволяє абстрагуватися від особливостей синтаксису різних баз даних і забезпечує розробників можливістю застосовувати один і той самий код у різних СУБД. Можливо перелічити такі переваги:

- Універсальність. Цей показник випливає з вищезгаданої особливості роботи з різними СУБД.
- Контроль версій та міграція. У Knex.js є спеціальний інструмент для налаштування конфігурації щоб полегшити міграцію – knexfile. Це дозволяє легко контролювати актуальність бази даних та коду.
- Підтримка транзакцій. Це дозволяє підтримувати складні операції та забезпечує цілісність баз даних.

3.2 Розробка шаблонів застосунку та прототипування

Для візуалізації вмісту та структури web-сайту використовується карта сайту. Вона показує які зв’язки містить між сторінками та як вони співвідносяться. Це забезпечує визначення пріоритетів та швидке виявлення прогалин до початку розробки, щоб кінцеві користувачі мали змогу користуватися сторінками як єдиним, цілісним потоком.

До початку роботи над картою було визначено перелік сторінок та вкладок:

- “Передстартова”;
- “Реєстрація”;
- “Авторизація”;
- “Панель інструментів”;
- “Статистика”;
- “Журнал”;
- “Налаштування”.

Наступним кроком було представлення цього переліку у вигляді карти сайту, яку зображено на рисунку 3.3.

Рис. 3.3 Карта сайту Mood journal

Інструментом для створення дизайну сайту було обрано Figma – сучасний хмарний інструмент для розробки дизайну сайтів. Він підтримує прототипування та створення систем, які дозволяють зберігати стилі та кольори. Figma також дозволяє працювати з векторною графікою, що робить його універсальним.

Отримавши карту сайту, наступним етапом було створення wireframes(каркасів) сайту на рисунку 3.4. Таке рішення дозволяє визначити скелет макету, а його простота у реалізації не забере багато часу. Це також дозволяє на ранніх етапах дослідити сторінки та вдосконалити їх концепцію.

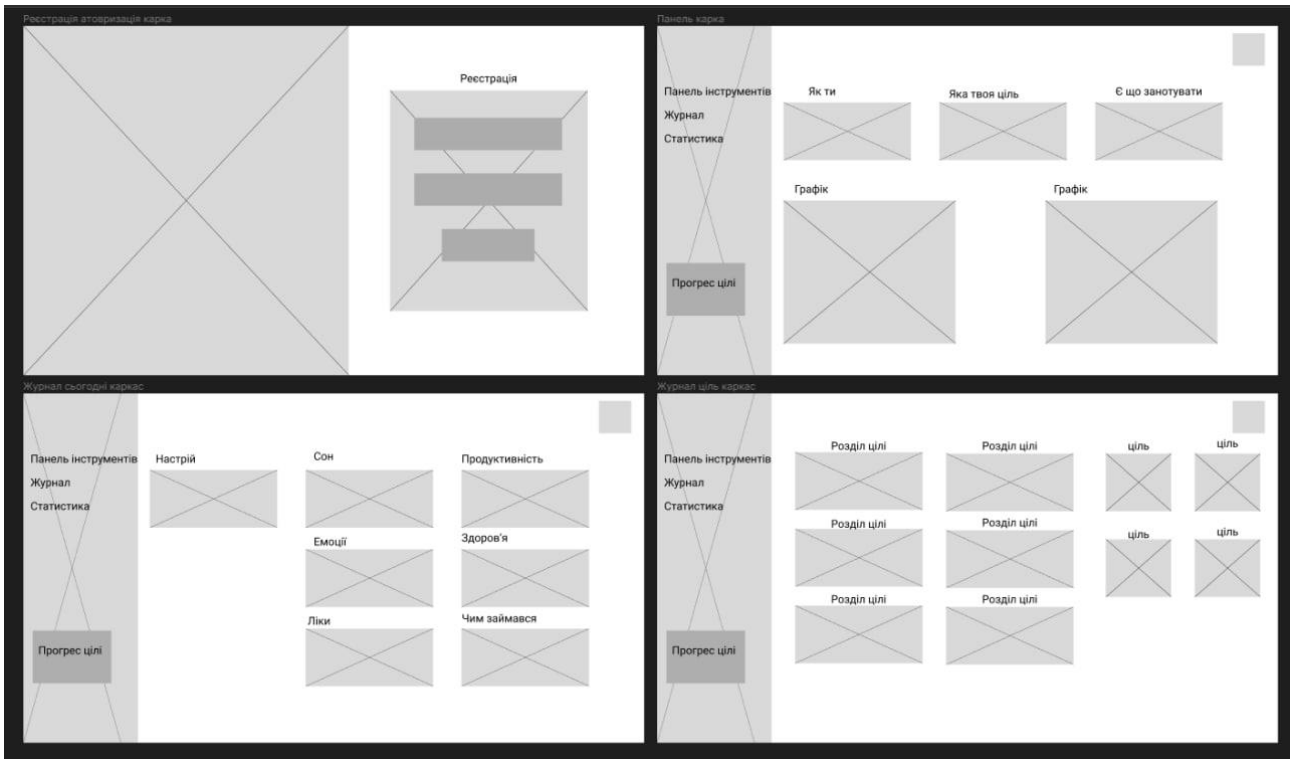


Рис. 3.4 Каркас сайту Mood journal

Даний каркас дозволив сформувати вимоги до наповнення сторінок:

- “Передстартова”: відображення кнопок-посилань на реєстрацію або авторизацію, анімація графічних елементів та заохочуючий напис.
- “Реєстрація” та “Авторизація”. Форма для введення даних, кнопка збереження даних.
- “Панель інструментів”. Ця сторінка є головною, тому має містити найбільш важливі інформацію. На ній було відображено введення швидких даних: настрій, короткий перелік емоцій, цілі, нотатки та графіки настрою за поточний тиждень.
- “Журнал” було розділено на дві категорії. Перша – “Сьогодні” яка буде містити детальний опис дня, який складатиметься з настрою, сну, емоцій, ліків, продуктивності, здоров’я та діяльності за день. Друга – “Нова ціль”, яка надаватиме можливість поставити цілі та визначити час їх досягнення.
- “Статистика” включає дві категорії: “Днів поспіль” та “Графік настроїв”. Перша дозволить дізнатися про загальну інформацію, таку

як середній час сну, кількість днів використання застосунку, нотатки. Друга відображатиме дані про настрій, їх загальну кількість та зміну.

- У каркасі також було визначено такі елементи: прогрес цілі, меню користувача з відображенням налаштувань сайту.

Для полегшення роботи з дизайном важливо створити UI Kit з переліком усіх елементів сайту. Так як у створенні застосовувалася бібліотека Material UI, то відповідно UI Kit містив готові компоненти, запропоновані від MUI.

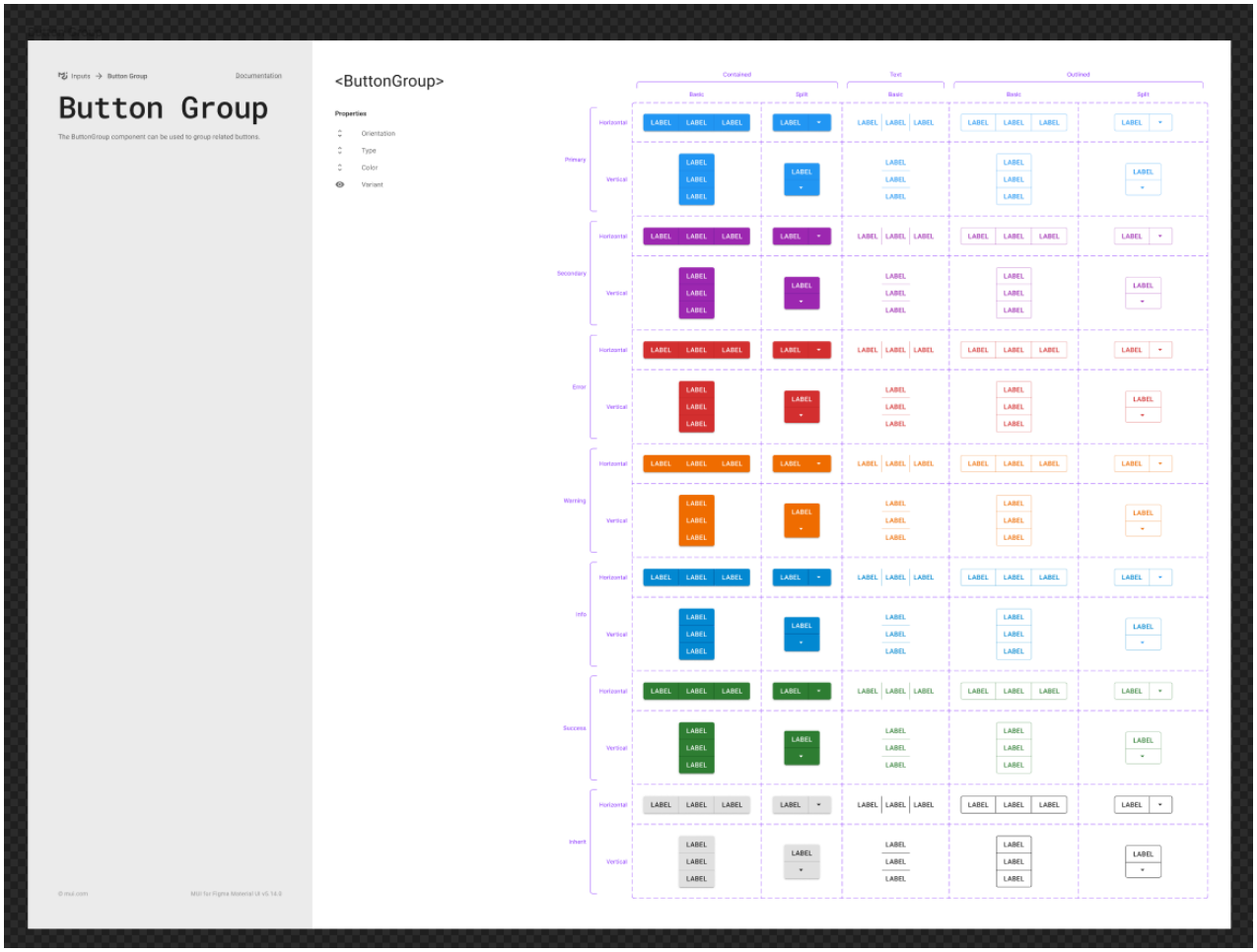


Рис. 3.5 UI Kit від Material UI

Для стилізації сайту було не достатньо базових налаштувань компонентів, тому було допрацьовано фірмові кольори та шрифти.

Roboto обрано в якості головного шрифту. Такі систем як Google, YouTube, Google Play використовують його, адже він є простим та зрозумілим. Фірмові кольори мають відповідати тематиці, тому вибір пав на пастельну палітру, яка б

крім головний кольорів містила б їх відтінки. У цьому підході застосувався принцип компліментарності – обрання кольорів і створення їх відтінків. У дизайні головним вважається колір powder blue, такий вибір було зроблено виходячи з того, що сині кольори заспокоюють та надають впевненості, другорядний колір honey dew є відтінком зеленого і створює асоціацію з безпекою, знижує дратівливість та втому.

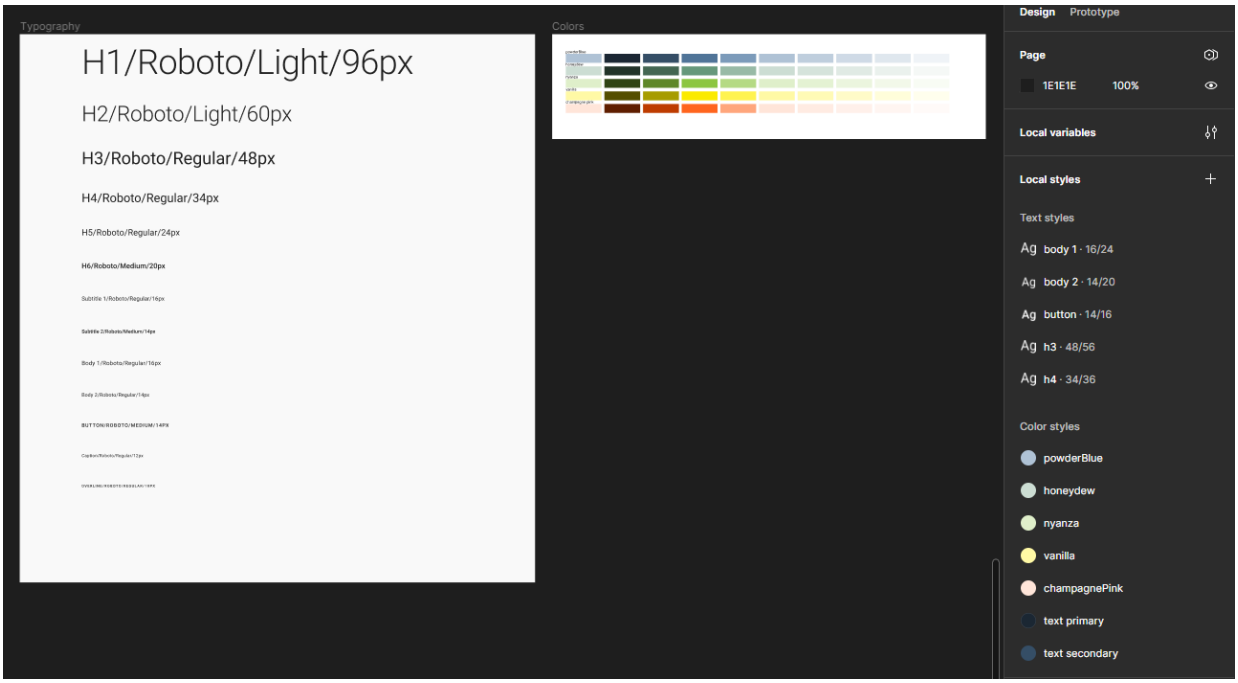


Рис. 3.6 Фірмові кольори та шрифти проекту Mood Journal

Пройшовши всі ці кроки стало можливо створити високорівневий дизайн сайту, який би точно відтворював всі елементи та компоненти.

Одними з перший було створено сторінки “Передстартова”, “Авторизація” та “Реєстрація”. Їх бачить користувач уперше, тому важливим є позитивне враження та гарний користувацький досвід. Тому на них було розміщено графічні елементи у фірмових кольорах проекту з анімацією їх появи. Це продемонстровано на рисунку 3.7.

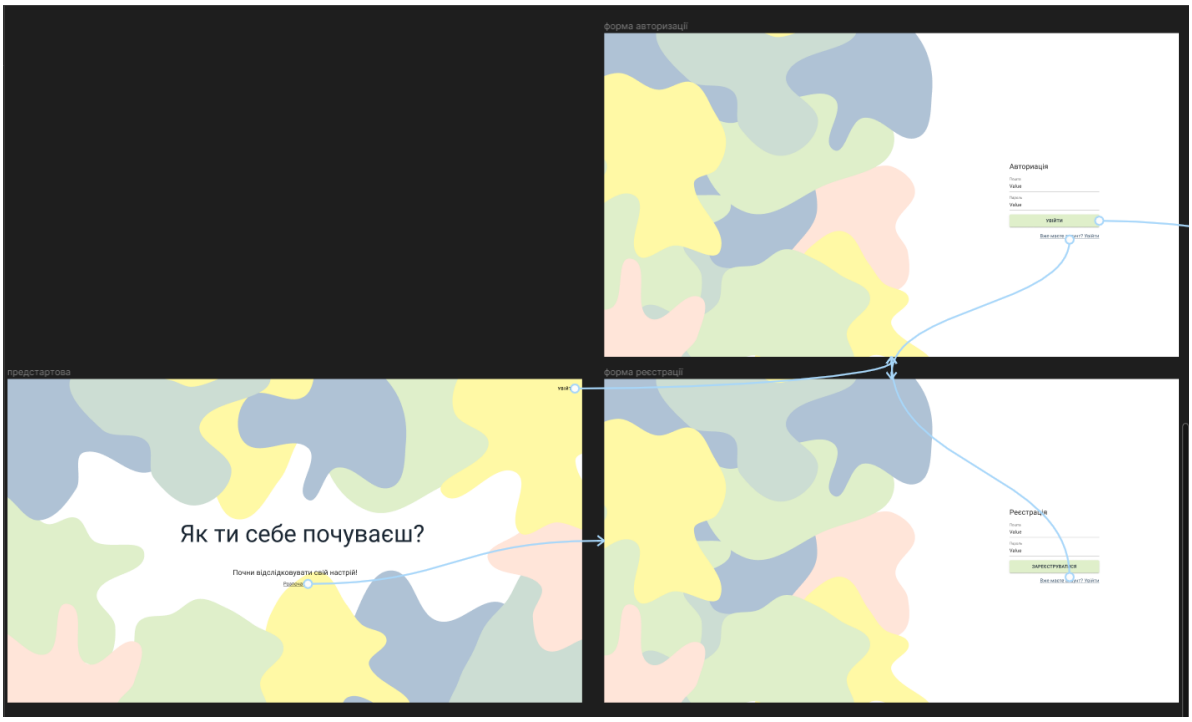


Рис. 3.7 Сторінки “Передстартова”, “Авторизація” та “Реєстрація”

Загалом було створено декілька фреймів з прототипування, для детального відображення шляху користувача та його взаємодії зі сторінками.

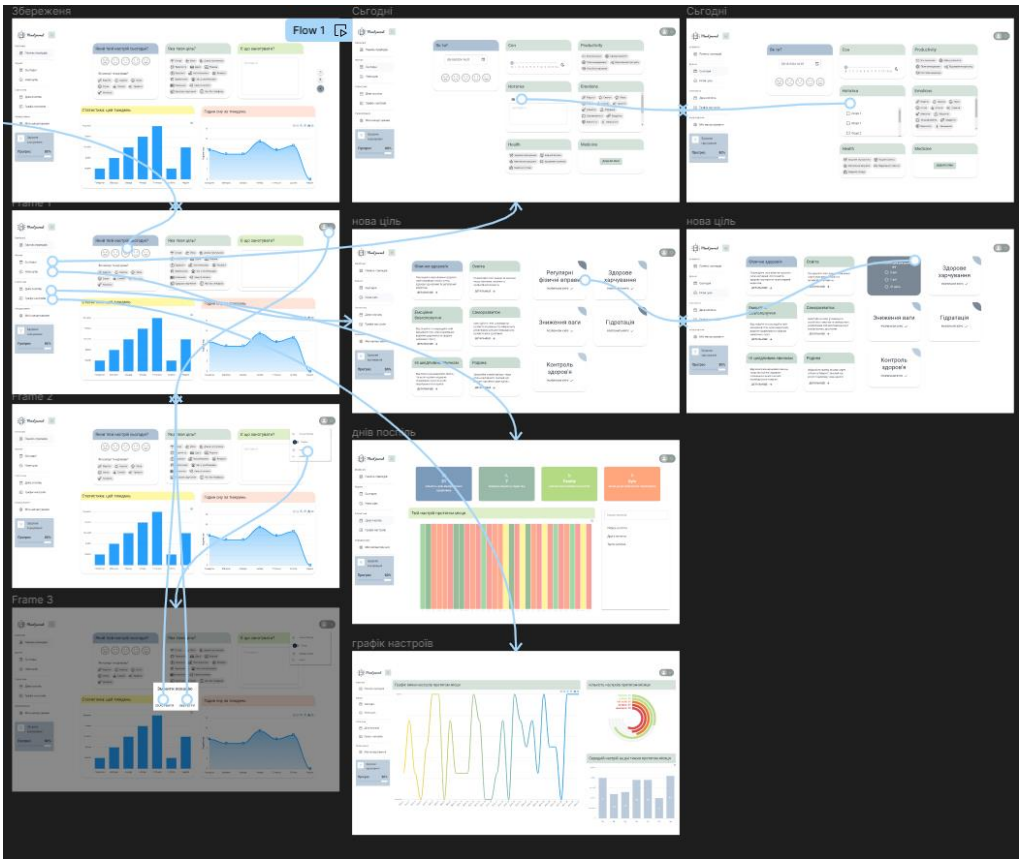


Рис. 3.8 Дизайн сайту Mood journal та його протипування

Результат, зображений на вищезгаданих рисунках повністю відповідає принципам Material Design. Графічні елементи є зрозумілими та привертають увагу користувача. Їх поведінка передбачувана і натискання на кожну кнопку або текстове поле викликає знайому йому дію. Кольори контрастні, а шрифти читабельні та різноманітні, мають чітку ієрархію.

3.3 Розробка бази даних

У якості бази даних було обрано MariaDB, яка створена на основі MySQL[14]. Забезпечуючи продуктивність та високу ефективність ця реляційна база даних дозволяє працювати з великим обсягом даних.

Перш за все, потрібно створити даталогічну модель бази даних, яка б відображала логічні зв’язки даних. Результати роботи зображено на рисунку 3.9.

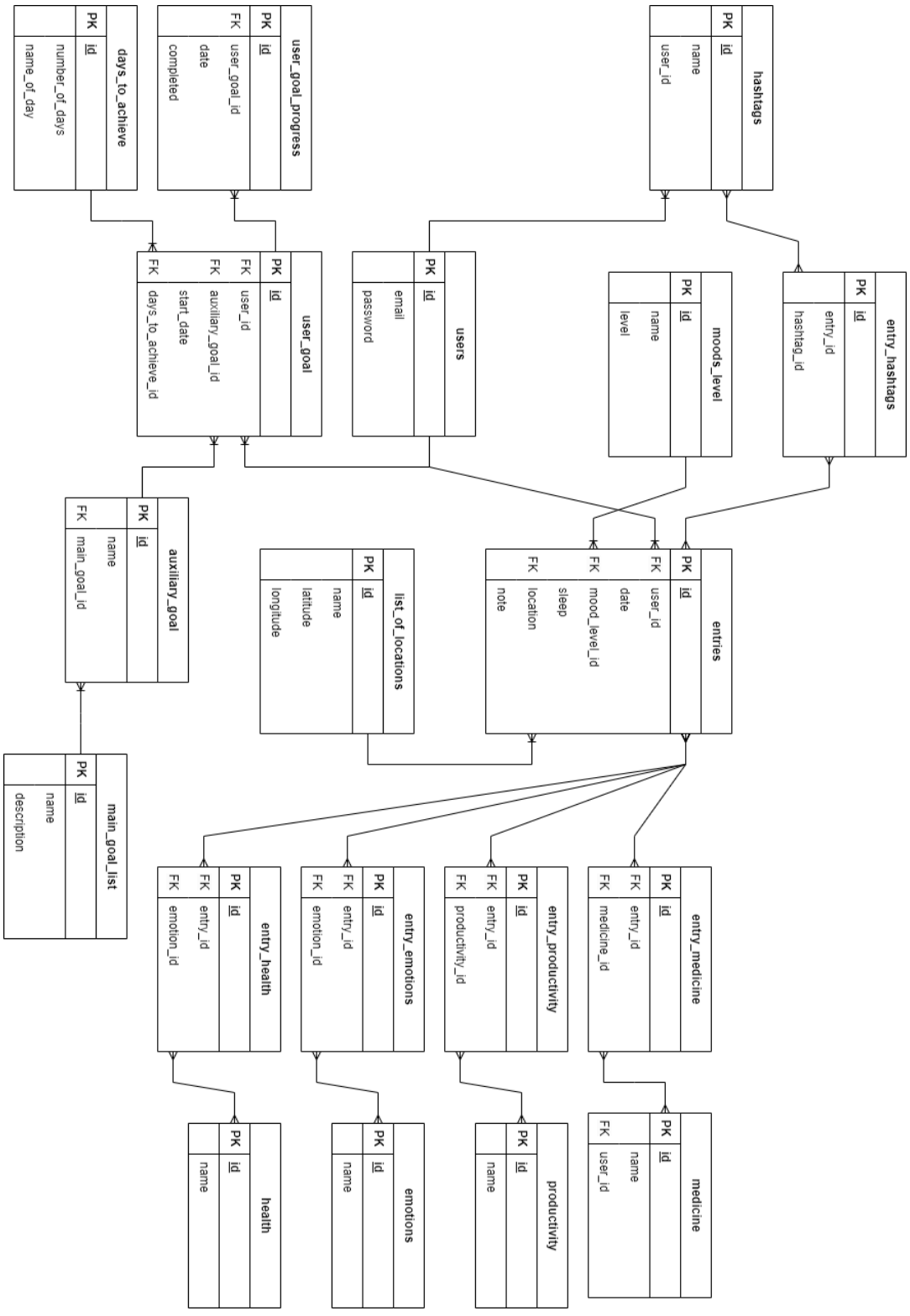


Рис. 3.9 Даталогічна модель бази даних проєкту Mood journal

Кожна таблиця представляє сутність, яка містить колонки. Також визначено первинні, які забезпечують унікальність, та зовнішні ключі, які використовуються для зв'язків таблиць. У даній моделі між таблицями представлено зв'язки один-до-багатьох та багато-до-багатьох.

Для роботи з базою даних обрано інструмент DBeaver. Він підтримує різні бази даних, такі як MySQL, MariaDB, PostgreSQL, Oracle, SQL Server, SQLite, тощо. Підключивши MariaDB та створивши базу даних, потрібно було створити таблиці. Для цього у DBeaver є інструмент додавання SQL-скриптів. Такий метод є швидким та надійним і не вимагає додаткових засобів.

```sql
CREATE TABLE entry_medicine (
    id INT PRIMARY KEY,
    entry_id INT,
    medicine_id INT,
    FOREIGN KEY (entry_id) REFERENCES Entry(id),
    FOREIGN KEY (medicine_id) REFERENCES Medicine(id)
);

CREATE TABLE entry_productivity (
    id INT PRIMARY KEY,
    entry_id INT,
    productivity_id INT,
    FOREIGN KEY (entry_id) REFERENCES Entry(id),
    FOREIGN KEY (productivity_id) REFERENCES Productivity(id)
);

CREATE TABLE entry_emotions (
    id INT PRIMARY KEY,
    entry_id INT,
    emotion_id INT,
    FOREIGN KEY (entry_id) REFERENCES Entry(id),
    FOREIGN KEY (emotion_id) REFERENCES Emotion(id)
);

CREATE TABLE entry_health (
    id INT PRIMARY KEY,
    entry_id INT,
    emotion_id INT,
    FOREIGN KEY (entry_id) REFERENCES Entry(id),
    FOREIGN KEY (emotion_id) REFERENCES Emotion(id)
);
CREATE TABLE main_goal_list (
    id INT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    description TEXT NOT NULL
);

CREATE TABLE auxiliary_goal (
```

Рис. 3.10 Створення таблиць у базі даних

У результаті було отримано базу даних, яка повністю відповідає раніше створеній моделі. Інтрумент DBeaver дозвляє переглянути діаграму бази даних, яку зображено на малюнку 3.11.

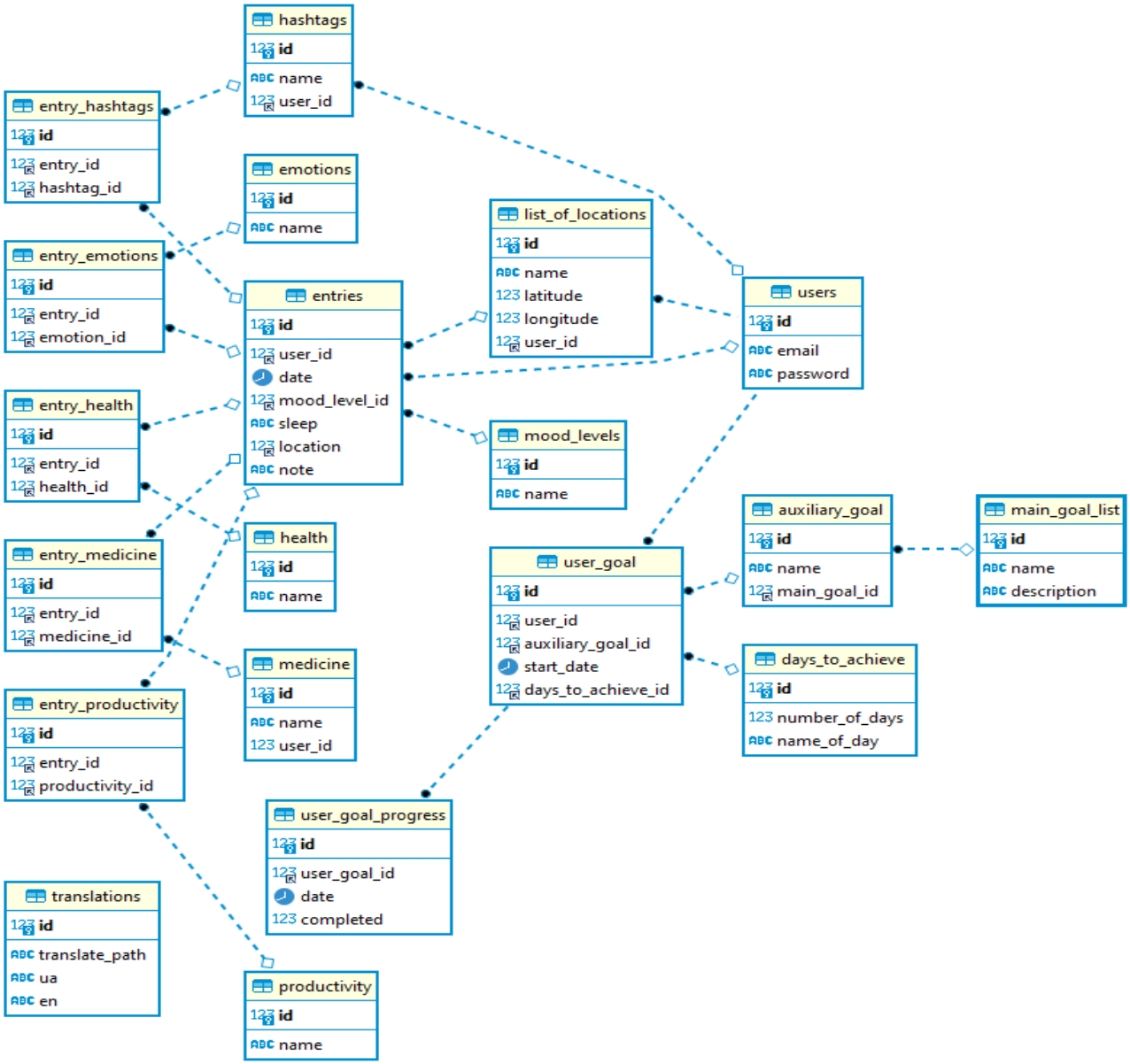


Рис. 3.11 Схема бази даних з інструменту DBeaver

3.3 Розробка серверної частини web-застосунку

Для створення серверної частини застосунку було обрано технології Node.js та Knex.js.

Перш за все, важливо налаштувати середовище розробки, тому було встановлено декілька пакетів для якісної роботи проєкту.

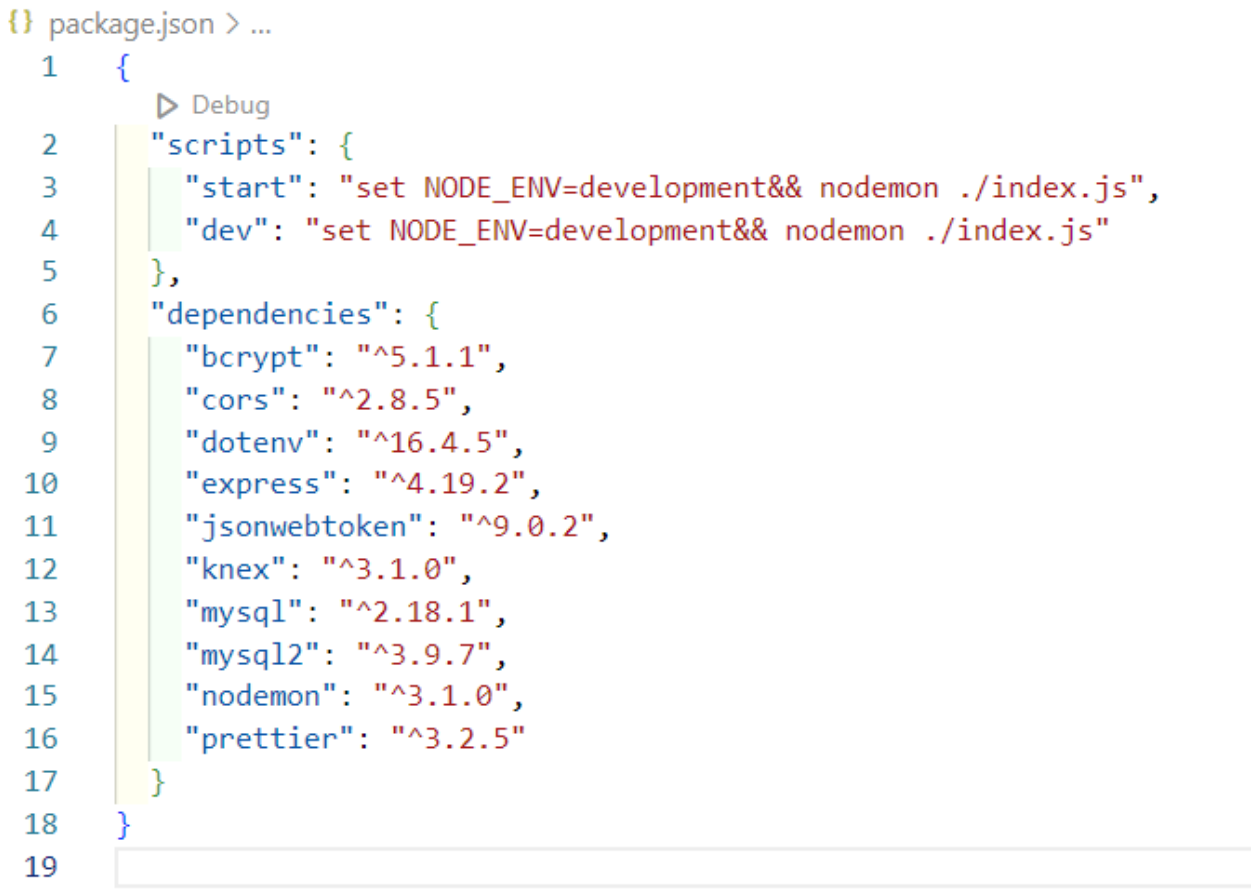


Рис. 3.12 Файл package.json серверної частини застосунку

Властивість “script” містить команди сценаріїв, які виконуються у різні моменти життєвого циклу проєкту, а “dependencies” представляє перелік пакетів та їх версій.

Далі важливо налаштувати підключення до бази даних, тобто конфігурацію Knex.js. (рис 3.13)

```
const dataBaseClient = process.env.DB_CLIENT || "mysql";
const host = process.env.DB_HOST || "localhost";
const dataBasePort = process.env.DB_PORT || 3306;
const databaseName = process.env.DB_NAME;
const databseUser = process.env.DB_USER;
const databasePassword = process.env.DB_PASSWORD;

const db = require("knex")({
  client: dataBaseClient,
  connection: {
    host: host,
    port: dataBasePort,
    database: databaseName,
    user: databseUser,
    password: databasePassword,
    dateStrings: "date"
},
  pool: {
    min: 0,
    max: 100,
    afterCreate: function (conn, done) {
      conn.query("show processlist  ", function (err) {
        if (err) {
          done(err, conn);
        } else {
          conn.query("show processlist  ", function (err) {
            done(err, conn);
          });
        }
      });
    }
  },

  acquireConnectionTimeout: 10000
});

module.exports = db;

```

Рис. 3.13 Конфігурація

Далі у файлі index.js було налаштовано базову конфігурацію серверу, налаштовано маршрутизацію та визначено порт на рисунку 3.14.

```
require('dotenv').config();
const express = require('express');
const app = express();
const routes = require('./src/routes/routes');

app.use(express.json());
app.use(process.env.API_PATH, routes);

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
    console.log(`Сервер запущено на порту ${PORT}`);
});

```

Рис. 3.14 Конфігурація серверу

Подальша робота полягала у створенні сервісів, які будуть зберігати логіку взаємодії з базою даних та використовуватися у контролерах. Кожен сервіс представляє з себе окремий модуль, який міститься базові CRUD-операції та додаткові операції, виходячи з логіки застосунку.

Get: Отримання всіх записів з таблиці на рисунку 3.15.

```
/**
 * Отримує всі записи з таблиці auxiliary_goal.
 *
 * @param {Object} [trx=db] - Транзакція бази даних або конфігурація бази даних за замовчуванням.
 * @returns {Promise<Array>} - Масив з усіма записами таблиці auxiliary_goal.
 * @throws {Error} - Помилка, якщо не вдалося отримати записи.
 */
const get = async (trx = db) => {
  try {
    const result = await trx("auxiliary_goal");
    return result;
  } catch (error) {
    console.log("can not get auxiliary_goal");
    throw error;
  }
};
```

Рис. 3.15 Функція get для таблиці auxiliary_goal

GetById: Отримання запису за ідентифікатором на рисунку 3.16.

```
19
20  /**
21   * Отримує запис з таблиці auxiliary_goal за вказаним ID.
22   *
23   * @param {Object} data - Об'єкт, що містить дані для пошуку.
24   * @param {number} data.id - Ідентифікатор запису.
25   * @param {Object} [trx=db] - Транзакція бази даних або конфігурація бази даних за замовчуванням.
26   * @returns {Promise<Object>} - Знайдений запис або undefined, якщо запис не знайдено.
27   * @throws {Error} - Помилка, якщо не вдалося отримати запис.
28   */
29  const getById = async (data, trx = db) => {
30    try {
31      const { id } = data;
32      const result = await trx("auxiliary_goal").where({ id });
33      return result[0];
34    } catch (error) {
35      console.log("can not getById auxiliary_goal");
36      throw error;
37    }
38  };
```

Рис. 3.16 Функція getById для таблиці auxiliary_goal

Create: Створення нового запису на рисунку 3.17.

```
40  /**
41   * Створює новий запис у таблиці auxiliary_goal.
42   *
43   * @param {Object} data - Дані для створення нового запису.
44   * @param {Object} [trx=db] - Транзакція бази даних або конфігурація бази даних за замовчуванням.
45   * @returns {Promise<number>} - Ідентифікатор створеного запису.
46   * @throws {Error} - Помилка, якщо не вдалося створити запис.
47   */
48  const create = async (data, trx = db) => {
49    try {
50      const result = await trx("auxiliary_goal").insert(data);
51      return result[0];
52    } catch (error) {
53      console.log("can not create auxiliary_goal");
54      throw error;
55    }
56  };
```

Рис. 3.17 Функція create для таблиці auxiliary_goal

UpdateById: Оновлення існуючого запису за ідентифікатором на рисунку 3.18.

```
/**
 * Оновлює запис у таблиці auxiliary_goal за вказаним ID.
 *
 * @param {Object} data - Дані для оновлення запису.
 * @param {number} data.id - Ідентифікатор запису, який потрібно оновити.
 * @param {Object} [trx=db] - Транзакція бази даних або конфігурація бази даних за замовчуванням.
 * @returns {Promise<number>} - Кількість оновлених записів (1 при успішному оновленні, 0 якщо запис не знайдено).
 * @throws {Error} - Помилка, якщо не вдалося оновити запис.
 */
const updateById = async (data, trx = db) => {
  try {
    const { id } = data;
    const updateData = { ...data };
    delete updateData.id;
    const result = await trx("auxiliary_goal").update(updateData).where({ id });
    return result;
  } catch (error) {
    console.log("Can't update auxiliary_goal");
    throw error;
  }
};
```

Рис. 3.18 Функція updateById для таблиці auxiliary_goal

Загалом було реалізовано 20 сервісів, кожен з яких відповідає таблиці з бази даних.

Так, як Mood journal містить авторизацію та реєстрацію, то було розроблено окремий контролер, який би відповідав за це зображено на рисунку 3.19 та рисунку 3.20.

```
const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");
const db = require("../db/dbConfig");

exports.register = async (req, res) => {
  const { email, password } = req.body;
  try {
    const hashedPassword = await bcrypt.hash(password, 10);
    const result = await db("users")
      .insert({
        email: email,
        password: hashedPassword
      })
      .returning(["id", "email"]);
    const token = jwt.sign({ id: result[0].id }, process.env.SECRET_KEY, { expiresIn: "1h" });
    res.status(201).send({
      user: result[0],
      token
    });
  } catch (error) {
    console.error(error);
    res.status(500).send("Something went wrong.");
  }
};

```

Рис. 3.19 Реєстрація користувача

```
exports.login = async (req, res) => {
  const { email, password } = req.body;
  try {
    const users = await db("users").where({ email: email });
    if (users.length > 0) {
      const user = users[0];
      const match = await bcrypt.compare(password, user.password);
      if (match) {
        const token = jwt.sign({ id: user.id }, process.env.SECRET_KEY, { expiresIn: "1h" });
        res.status(200).send({ token });
      } else {
        res.status(401).send("Authentication failed");
      }
    } else {
      res.status(404).send("User not found");
    }
  } catch (error) {
    console.error(error);
    res.status(500).send("Something went wrong.");
  }
};
```

Рис. 3.20 авторизація користувача

Для надійності збереження паролів було використано бібліотеку bcrypt, а для аутентифікації користувача сервер у результаті успішної обробки даних відправляє токен, який містить загальну інформацію про користувача.

Останній крок створення серверної частини включає в себе налаштування маршрутів.

Для обробки Cross-Origin Resource Sharing потрібно додатиmiddleware, який дозволить серверу відповідати на запити інших доменів.

```
15
16  app.use(cors());
17
```

Рис. 3.21 CORS Middleware

Після такого кроку можливо створювати різноманітні маршрути, які будуть збирати логіку з сервісів і відповідати на запити клієнта. Один з маршрутів зображено на рисунку 3.22.

```
  const { userId } = req.query;
  const emotions = await emotionsService.get();
  const moodLevels = await moodLevelService.get();
  const auxiliaryGoal = (await auxiliaryGoalService.get()).slice(0, 15);
  const todayEntry = (await entriesService.getByToday(userId)) || null;
  const todayEmotions = todayEntry ? await entryEmotionsService.getByEntryId({ entry_id: todayEntry.id }) : null;
  const health = await healthService.get();
  const hashtags = await hashtagsService.getByUserId({user_id: userId});

  res.json({ emotions, moodLevels, auxiliaryGoal, todayEntry, todayEmotions, health, hashtags });
});
```

Рис. 3.22 Код маршруту /dashboard

3.3 Розробка клієнтської частини web-застосунку

Клієнтська частина проєкту розроблялася за допомогою бібліотеки React та декількох допоміжних бібліотек. Головним компонентом є App.js, який містить всю логіку застосунку та запит на переклади.

```
import Routes from "routes";
import NavigationScroll from "layout/NavigationScroll";
import MainProvider from "provider/MainProvider";
import { useEffect } from "react";
import api from "api";
import { useDispatch } from "react-redux";
import { appSettingsActions } from "./redux/slices/appSettingsSlice";

const App = () => {
  const dispatch = useDispatch();
  useEffect(() => {
    const fetchTranslations = async () => {
      const result = await api.translations.get();
      dispatch(appSettingsActions.setTranslates(result));
    };

    fetchTranslations();
  });

  return (
    <NavigationScroll>
      <MainProvider>
        <Routes />
      </MainProvider>
    </NavigationScroll>
  );
};

export default App;
```

Рис. 3.23 Компонент App.js

Логіку стилізації та локалізації було винесено у окремі провайдери, що робить код більш читабельним та легшим до масштабованості, які зображено на рисунку 2.34 та рисунку 3.25.

```
import { IntlProvider } from "react-intl";
import { useSelector } from "react-redux";
import { selectLocale, selectTranslates } from "../redux/slices/appSettingsSlice";

const LocaleProvider = ({ children }) => {
  const locale = useSelector(selectLocale);
  const translates = useSelector(selectTranslates);

  return (
    <IntlProvider locale={locale === "ua" ? "uk" : locale} messages={translates ? translates[locale] : {}}>
      {children}
    </IntlProvider>
  );
};

export default LocaleProvider;
```

Рис. 3.24 Провайдер з налаштування локалізації

```
import { createTheme, CssBaseline, ThemeProvider } from "@mui/material";
import { useMemo } from "react";
import { useSelector } from "react-redux";
import globalTheme from "../styles/globalTheme";
import { selectTheme } from "../redux/slices/appSettingsSlice";

const StyleProvider = ({ children }) => {
  const style = useSelector(selectTheme);
  const theme = useMemo(() => createTheme(globalTheme(style)), [style]);

  return (
    <ThemeProvider theme={theme}>
      <CssBaseline />
      {children}
    </ThemeProvider>
  );
};

export default StyleProvider;
```

Рис. 3.25 Провайдер з налаштування теми

Для налаштування теми також створено файл глобальної темізації. Функція globalTheme повертає об’єкт на основі реживу mode(темний або світлий).Вона

використовує палітру кольорів, отриману від функції colorsPalette і налаштовує кольори для різних елементів теми відповідно до обраного режиму.

```
131  const globalTheme = (mode) => {
132    const colors = colorsPalette(mode);
133
134    return {
135      palette: {
136        mode: mode,
137        ...(mode === "dark"
138          ? {
139              background: {
140                default: colors.background
141              },
142              primary: {
143                main: colors.powderBlue.DEFAULT,
144                light: colors.powderBlue[600],
145                dark: colors.powderBlue[200],
146                contrast: colors.powderBlue[300],
147                extraLight: colors.powderBlue[800],
148                extraDark: colors.powderBlue[100]
149              },
150              secondary: {
151                main: colors.honeydew[300]
152              },
153              success: {
154                main: colors.nyanza[300]
155              },
156              text: {
157                primary: colors.powderBlue[900]
158              },
159              champagnePink: {
160                main: colors.champagne_pink[300],
161                light: colors.champagne_pink[700],
162                dark: colors.champagne_pink[400],
```

Рис. 3.26 Частина коду глобальної темізації

Для управління станом застосунку використовувався Redux. Функція configureStore створює store з налаштованими параметрами, об'єкт reducer містить ключі, які відповідають різним slice стану застосунку, і значення, які відповідають їхнім редукторам.

```
import { configureStore } from "@reduxjs/toolkit";
import { appSettingsReducer } from "./slices/appSettingsSlice";
import { dashboardReducer } from "./slices/dashboardSlice";
import { menuReducer } from "./slices/menuSlice/menuSlice";
import { userDataReducer } from "./slices/userDataSlice/userDataSlice";

export default configureStore({
  reducer: {
    appSettingsReducer,
    dashboardReducer,
    menuReducer,
    userDataReducer
  }
});
```

Рис. 3.27 Store

Для створення slice, стану, який відповідає за налаштування застосунку, використовується функція createSlice і . Для повернення початкового стану застосунку, тобто той, у який не було внесено жодних змін, використано initialState. Для повернення нового стану використовуються reducers – функції, які приймають поточний стан і повертають значення оновленого.Селектори дозволяють отримувати підмножини стану, наприклад, поточну тему, мову тощо.

```
6   const initialState = () => ({
7     theme: appConst.DEFAULT_THEME,
8     locale: appConst.DEFAULT_LOCALE,
9     currentUser: { isAuth: false, id: "" },
10    translates: { [appConst.DEFAULT_LOCALE]: {} }
11  });
12
13  const appSettingsSlice = createSlice({
14    name: "appSettingsSlice",
15    initialState: initialState(),
16    reducers: {
17      setAppSettings: (state, action) => {
18        state = action.payload;
19        return state;
20      },
21      setTheme: (state, action) => {
22        state.theme = action.payload;
23      },
24      setLocale: (state, action) => {
25        state.locale = action.payload;
26      },
27      setTranslates: (state, action) => {
28        state.translates = action.payload;
29      },
30      setCurrentUser: (state, action) => {
31        state.currentUser = action.payload;
32        state.currentUser.isAuth = true;
33      },
34      toggleColorMode: (state) => {
35        if (state.theme === themeConst.LIGHT_THEME) {
36          state.theme = themeConst.DARK_THEME;
37          localStorageUtil.setTheme(themeConst.DARK_THEME);
38        } else if (state.theme === themeConst.DARK_THEME) {
39          state.theme = themeConst.LIGHT_THEME;
40          localStorageUtil.setTheme(themeConst.LIGHT_THEME);
41        }
42      }
43    }
44  });
45
```

Рис. 3.28 Код appSettingsSlice

Щоб користувач міг змінювати сторінки та взаємодіяти з маршрутизацією застосунку важливо правильно створити маршрути. Для цього використовувалася бібліотека React Router. useRoutes — це хук, який використовується для створення маршрутизованих компонентів на основі конфігурації маршрутів. Ця функція приймає масив об'єктів маршруту.MainRoutes містить основні маршрути застосунку, а AuthenticationRoutes конфігурацію маршрутів для аутентифікації.

```
import { useRoutes } from "react-router-dom";

import MainRoutes from "./MainRoutes";
import AuthenticationRoutes from "./AuthenticationRoutes";

export default function ThemeRoutes() {
  return useRoutes([MainRoutes, ...AuthenticationRoutes]);
}
```

Рис. 3.29 Налаштування маршрутизації

Для взаємодії з сервером було розроблено утиліти, які б відповідали за виконання HTTP-запитів. Вони включають функцію для створення затримок під час запиту, стандарту функцію fetch та об'єкт client, який надає методи для здійснення запитів з різними HTTP-методами (GET, POST, PATCH, DELETE). Функція wait приймає один параметр delay і повертає Promise, який вирішується після заданого часу. Ця штучна затримка створюється для безпомилково виконання іншої операції

```
function wait(delay) {
  return new Promise((resolve) => {
    setTimeout(resolve, delay);
  });
}
```

Рис. 3.30 Функція затримки wait

Функція request приймає три параметри: url, method, data. Ця функція спочатку створює об'єкт options, який містить метод запиту і якщо передано дані, вони перетворюються у формат JSON і додаються до опцій разом із відповідними заголовками. Після чого відбувається затримка у 300 мілісекунд і виконується запит за допомогою fetch. У випадку успішної відповіді, вона парситься як JSON, якщо ні – викидається помилка.

```
10
11  function request(url, method = "GET", data = null) {
12    const options = { method };
13
14    if (data) {
15      options.body = JSON.stringify(data);
16      options.headers = {
17        "Content-Type": "application/json; charset=UTF-8"
18      };
19    }
20
21    return wait(300)
22      .then(() => fetch(SERVER_HOST + SERVER_PORT + SERVER_API_PATH + url, options))
23      .then((response) => {
24        if (!response.ok) {
25          throw new Error();
26        }
27
28        return response.json();
29      });
30  }
```

Рис. 3.31 Функція запиту request

Об'єкт client забезпечує зручний і інтуїтивно зрозумілий інтерфейс для виконання HTTP-запитів. Він включає методи: get(url), post(url, data), patch(url, data), delete(url). Кожен метод використовує функцію request з відповідними параметрами для здійснення запиту.

```
32  export const client = {
33    get: (url) => request(url),
34    post: (url, data) => request(url, "POST", data),
35    patch: (url, data) => request(url, "PATCH", data),
36    delete: (url) => request(url, "DELETE")
37  };
38
```

Рис. 3.32 Об'єкт client

Використовуючи об'єкт client, можна легко здійснювати запити до різних кінцевих точок API з підтримкою різних HTTP-методів.

```
dashboard: {
  getData: (userId) => client.get(`/dashboard?userId=${userId}`),
  createEntry: (data) => client.post("/create-entries", data),
  updateEntry: (data) => client.patch(`/entries?entryId=${data.entryId}`, data)
}
```

Рис. 3.33 Приклад використання client

На цьому базові налаштування проєкту закінчено і можна створювати компоненти. Для цього використано функціональні компоненти React, такий підхід є сучасним та надає змогу створювати кастомні хуки для взаємодії зі станом застосунку або винесення логіки. Метод життєвого циклу return повертає готовий до відображення елемент React. Для побудови цих елементів знадобилася бібліотека Material UI.

```
const TotalGrowthBarChart = ({ isLoading }) => {
  const [series, options] = useGetChartData();
  const intl = useIntl();

  return (
    <>
      {isLoading ? (
        <SkeletonTotalGrowthBarChart />
      ) : (
        <CardWrapper border={false} content={false}>
          <Grid container spacing={gridSpacing}>
            <TextWrapper item xs={12}>
              <TextWrapper container alignItems="center" justifyContent="space-between">
                <TextWrapper item>
                  <Grid container direction="column" spacing={1}>
                    <Grid item>
                      <Typography variant="h5" sx={{ p: 1 }}>
                        {intl.formatMessage({ id: "statisticThisWeek" })}
                      </Typography>
                    </Grid>
                  </Grid>
                </TextWrapper>
              </TextWrapper>
            </TextWrapper>
            <Grid item xs={12}>
              <Chart series={series} options={options} height={380} type="bar" />
            </Grid>
          </Grid>
        </CardWrapper>
      )}
    </>
  );
};

```

Рис. 3.34 Код компоненту TotalGrowthBarChart

```
import { useMemo } from "react";
import { useSelector } from "react-redux";
import { dashboardDataSelect } from "../../../../redux/slices/dashboardSlice";
import getIconByName from "utils/getIconByName";
import { useIntl } from "react-intl";

const useGetEmotionsData = () => {
  const { emotions } = useSelector(dashboardDataSelect);
  const intl = useIntl();
  return useMemo(
    () =>
      emotions
        ? emotions
            .map(({ id, name }) => ({
              id,
              label: intl.formatMessage({ id: `emotion.${name.toLowerCase()}` }),
              icon: getIconByName(name)
            }))
            .slice(0, 7)
        : [],
    [emotions, intl]
  );
};

export default useGetEmotionsData;
```

Рис. 3.35 Кастомний хук зборки даних для компоненту Chip

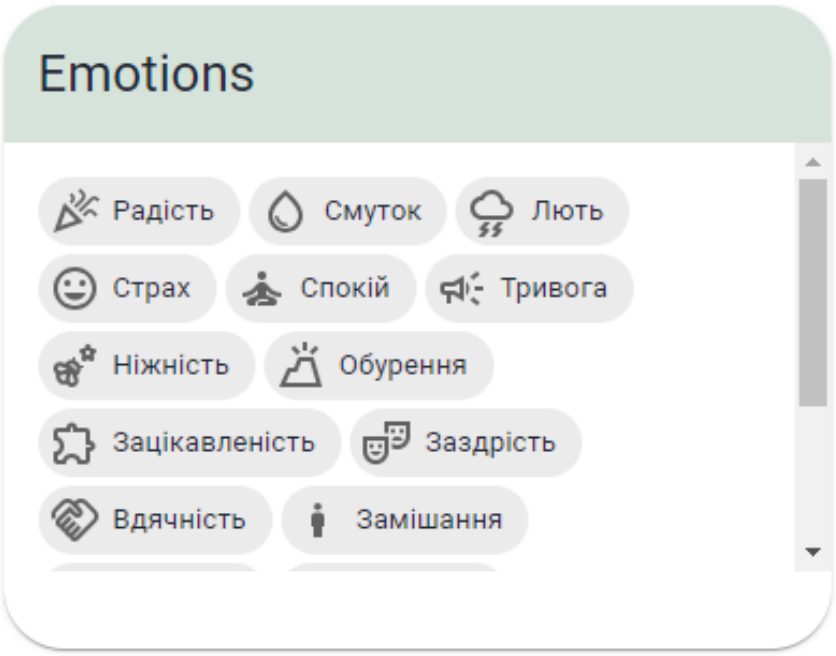


Рис. 3.36 Вигляд компоненту Emotions у інтерфейсі користувача

4 ТЕСТУВАННЯ РОЗРОБЛЕНОГО САЙТУ

4.1 Кросбраузерність

Користувачі мають мати можливість використовувати застосунок, який буде коректно працювати не залежно від браузеру. Кожен браузер має свої особливості інтерпретації web-сторінок, а різні їх версії підтримувати різні технології. Бібліотека Material UI розроблена з урахуванням кросбраузерності, що забезпечує повну сумісність з різними сучасними браузерами. MUI містить в собі скидання стандартних стилів браузера та їх нормалізацію за допомогою CssBaseline, а більшість компонентів побудовано з використанням технологій Flexbox та Grid, які підтримуються у всіх платформах. У ході тестування було протестовано вигляд сайту у чотирьох браузерах.

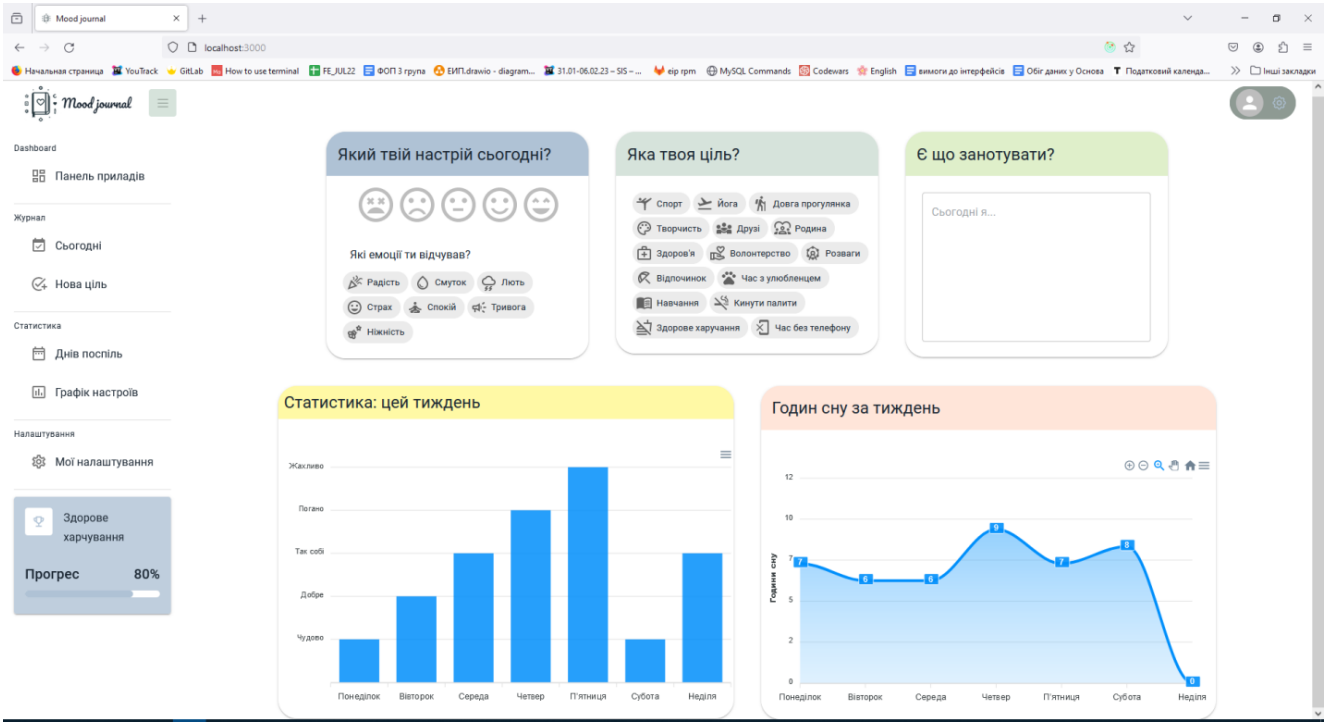


Рис. 4.1 Mozilla Firefox

Рис. 4.2 Google Chrome

Рис. 4.3 Opera

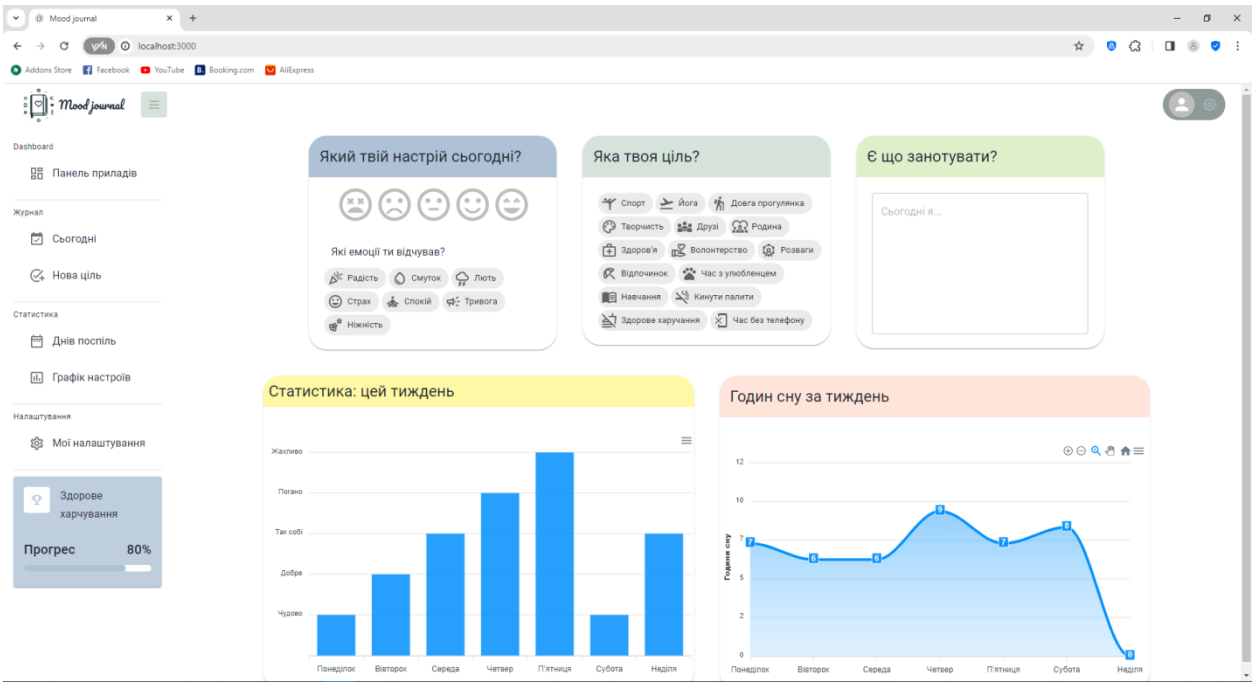


Рис. 4.4 Avast Secure Browser

4.2 Адаптивність

Адаптивність сайту є однією з базових вимог до сучасних сайтів. Вона означає можливість адаптації до різних розмірів екранів пристроїв. Це забезпечує позитивний користувацький досвід та збільшує обсяг цільової аудиторії. Основними вимогами є:

- Гнучкість макету – динамічна зміна розташування елементів та їх розміри.
- Адаптивна типографія для зручного читання тексту.
- Адаптивні зображення, тобто вони мають різну роздільну здатність.
- Touch-Friendly. Посилання, кнопки, форми введення даних легко застосувати на сенсорному екрані.

Застосунок Mood journal відповідає всім вищеперерахованим вимогам, тому забезпечує користувачів зручністю та приємним користувацьким досвідом.

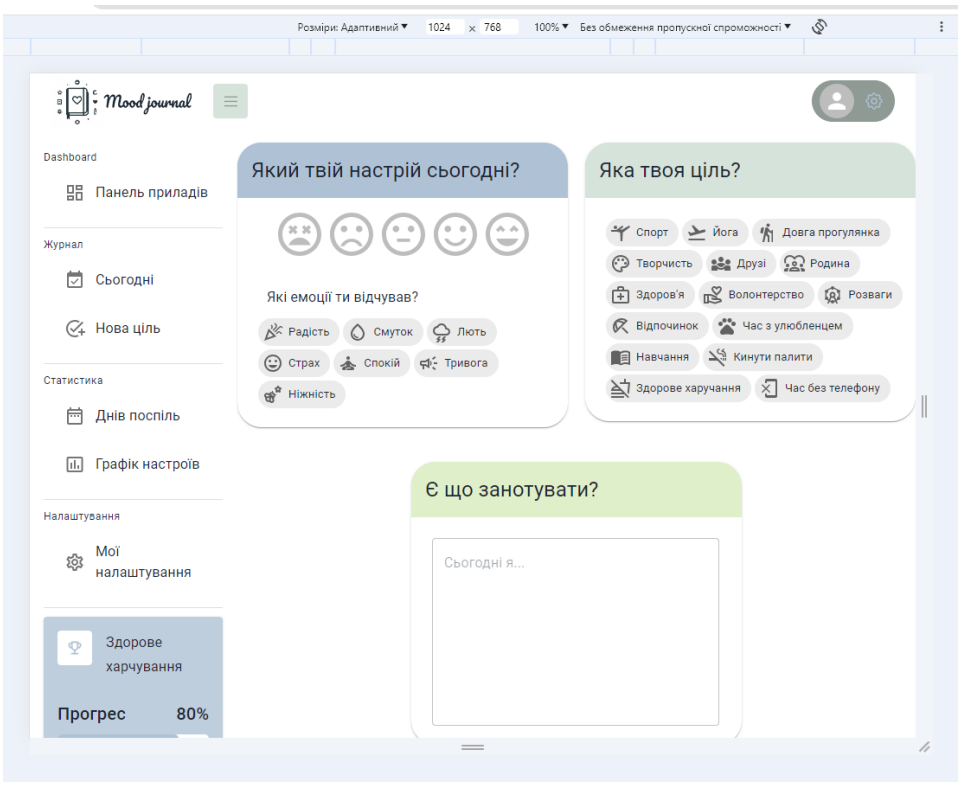


Рис. 4.5 Вигляд сайту на екрані з роздільною здатністю 1024 x 768

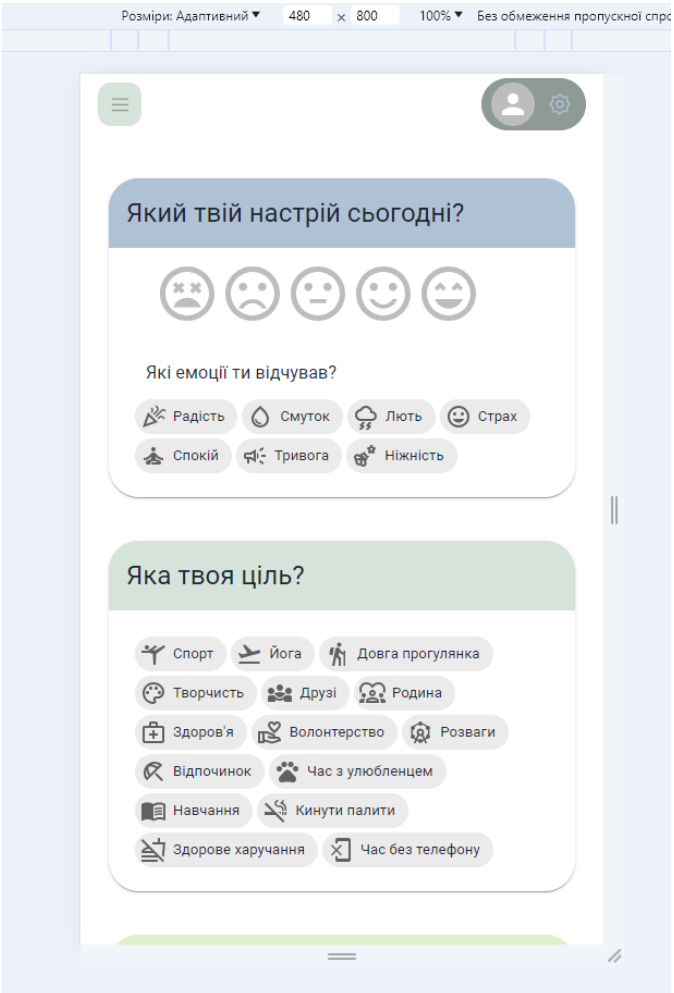


Рис. 4.6 Вигляд сайту на екрані з роздільною здатністю 480 x 800

ВИСНОВКИ

У результаті виконання дипломної роботи було розроблено web-застосунок для самоаналізу та самотерапії. Її актуальність полягає у тому, що розуміння емоцій та керування ними є невід'ємною частиною "емоційного інтелекту".

1. Проаналізувано вже існуючі застосунки та знайдено три аналоги, а саме MoodTraker, Journey.cloud, eMoods, виявлено їх переваги та недоліки, складено таблицю порівняння.
2. Розроблено технічне завдання, визначено функціональні та нефункціональні вимоги до застосунку.
3. Досліджено інструменти розробки та обрано: JavaScript, React, Redux, MUI, Format.JS, Apexcharts для клієнтської частини, Node.js, Knex.js для серверної з урахуванням вимог.
4. Розроблено web-застосунок для самоаналізу та самотерапії. У ході роботи було встановлено, що використання React, як бібліотеки для створення інтерфейсу користувача, дозволяє створити гнучкий та легко масштабований клієнтський застосунок. Крім цього, Node.js надає широкий спектр можливостей його використання та розширення функціоналу завдяки менеджеру пакетів. У результаті роботи розроблений web-застосунок відповідає функціональним та нефункціональним вимогам, збільшивши кількість методів для самоаналізу та самотерапії у вигляді створення кастомних хештегів для пошуку важливих записів, використання геолокації для визначення впливу розташування та постановка цілей.
5. Проведено тестування на кросбраузерність та адаптивність
6. Робота пройшла апробацію:

Зінченко А.Є., Замрій І.В. Визначення додаткових функцій веб-додатку Mood journal трекеру емоцій та настрою для аналізу та самотерапії: Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ», 24 квітня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С. 93-94.

Зінченко А.Є., Замрій І.В. Покращення продуктивності та ефективності веб-додатку Mood journal трекеру емоцій та настрою для аналізу та самотерапії: Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ», 24 квітня 2024 р., Київ, інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С. 227-228.

ПЕРЕЛІК ПОСИЛАНЬ

1. MoodTracker [Електронний ресурс] – Режим доступу до ресурсу: https://www.moodtracker.com/.
2. Journey.cloud [Електронний ресурс] – Режим доступу до ресурсу: https://journey.cloud/.
3. eMoods [Електронний ресурс] – Режим доступу до ресурсу: https://emoodtracker.com/.
4. JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript .
5. PRIYAM M. Exploring Object-Oriented Programming in JavaScript [Електронний ресурс] / MONDAL PRIYAM. – 2023. – Режим доступу до ресурсу: https://medium.com/@priyam_mondal/exploring-object-oriented-programming-in-javascript-25ebb1cc13c9 .
6. React [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/.
7. Redux [Електронний ресурс] – Режим доступу до ресурсу: https://redux.js.org/.
8. React Router [Електронний ресурс] – Режим доступу до ресурсу: https://reactrouter.com/en/main .
9. Apexcharts [Електронний ресурс] – Режим доступу до ресурсу: https://apexcharts.com/ .
10. Format.JS [Електронний ресурс] – Режим доступу до ресурсу: https://formatjs.io/ .
11. Material UI [Електронний ресурс] – Режим доступу до ресурсу: https://mui.com/ .
12. About Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/en/about.
13. Knex.js [Електронний ресурс] – Режим доступу до ресурсу: https://knexjs.org/
14. MariaDB [Електронний ресурс] – Режим доступу до ресурсу: https://mariadb.org/ .

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ
(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-застосунку Mood journal відстеження емоцій та настрою для аналізу та самотерапії за допомогою технологій React, JS, Node.JS

Виконала студентка 4 курсу
Групи ПД-41
Зінченко Анастасія Євгеніївна
Керівник роботи
Д.т.н, доц, завідувач кафедри ІПЗ Замрій Ірина Вікторівна

Київ – 2024

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** – – збільшення методів для самоаналізу та самотерапії за допомогою web-застосунку з використанням бібліотеки React, мови програмування JS та платформи з відкритим кодом Node.js.
- **Об'єкт дослідження** – процес стеження за власними емоціями та закономірностями між ними.
- **Предмет дослідження** – використання технологій React, JavaScript (JS) та Node.js для створення функціонального та ефективного web-інструменту для аналізу та самотерапії.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Проаналізувати вже існуючі застосунки, виявити їх переваги та недоліки, скласти таблицю порівняння.
2. На основі проаналізованих застосунків розробити функціональні та нефункціональні вимоги.
3. Дослідити інструменти розробки, розробити архітектуру та дизайн web-застосунка.
4. Розробка web-застосунка на основі обраних інструментів.
5. Тестування web-застосунка.
6. Пройти апробацію на Науково-технічних конференціях.

3

АНАЛІЗ АНАЛОГІВ

Показник	MoodTracker	Journey.cloud	eMoods	Mood journal
Створення та редагування щоденних нотаток	+	+	+	+
Налаштування переліку емоцій, їх груп та кольорів	-	-	+	+
Статистика. Відображення графіку настрою та емоцій, їх підрахунок	+	-	-	+
Відображення даних у вигляді календаря	-	+	+	+
Можливість імпортування даних	-	-	+	+
Можливість додавати перелік ліків, вітамінів які приймав користувач	+(лише платна версія)	-	+	+
Можливість ставити цілі	-	-	-	+
Можливість структурувати нотатки за типами або темами	-	-	-	+
Прив'язка настроїв до геолокації	-	+(лише платна версія)	-	+

4

ВИМОГИ ДО ДОДАТКУ

Функціональні:

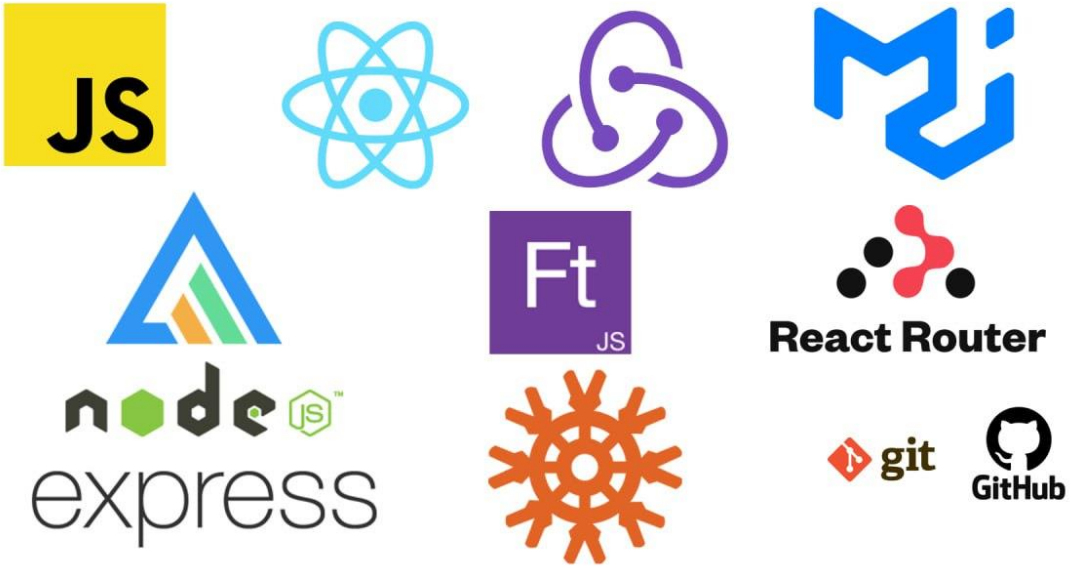
1. Реєстрація та авторизація користувача, перевірка введених даних на валідність.
2. Створення та редагування щоденних записів з обов'язковим обранням настрою, валідація цих даних.
3. Завантаження діаграм та графіків.
4. Розрахунок підсумкових даних за весь час використання застосунку.
5. Можливість додавати, видаляти та змінювати геолокацію.

Нефункціональні:

1. Сумісність. Додаток має бути доступним на різних пристроях та веб-браузерах.
2. Конфіденційність. Додаток повинен бути захищений від витоку конфіденційних даних.

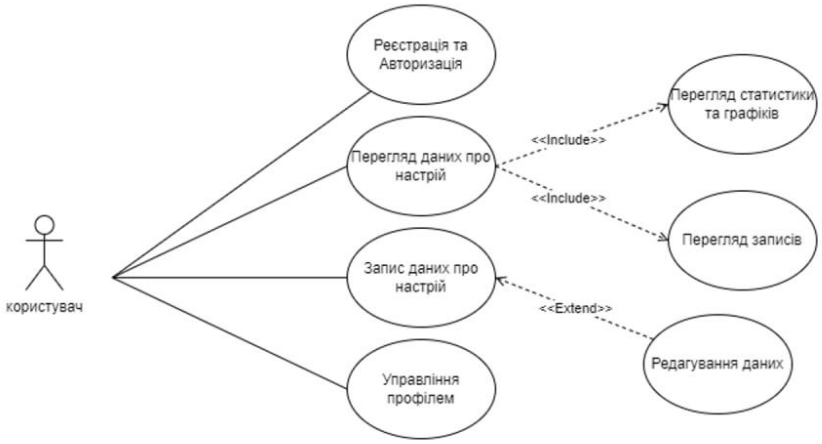
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



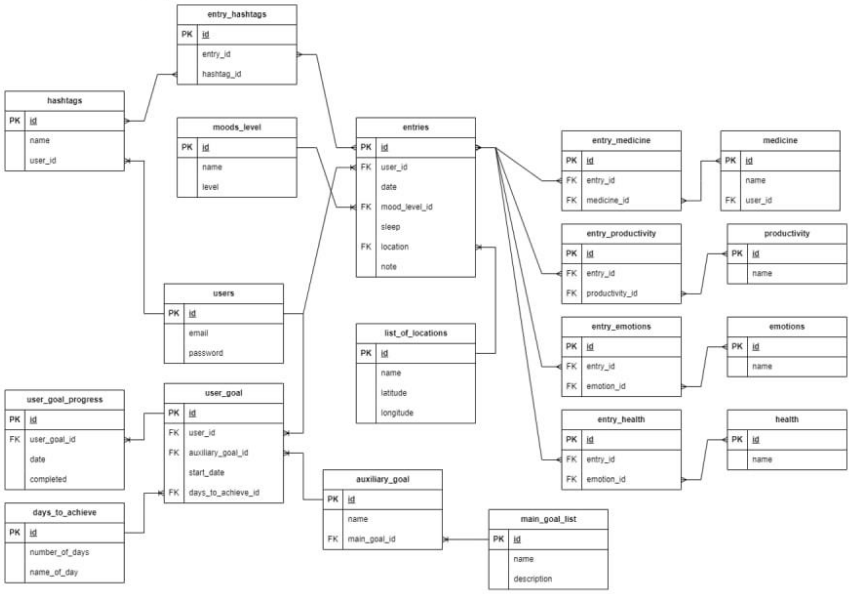
6

Діаграма варіантів використання



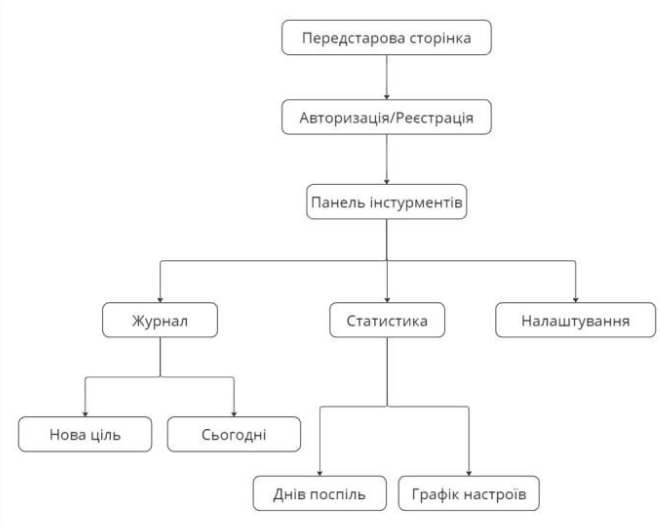
7

Даталогічна модель бази даних



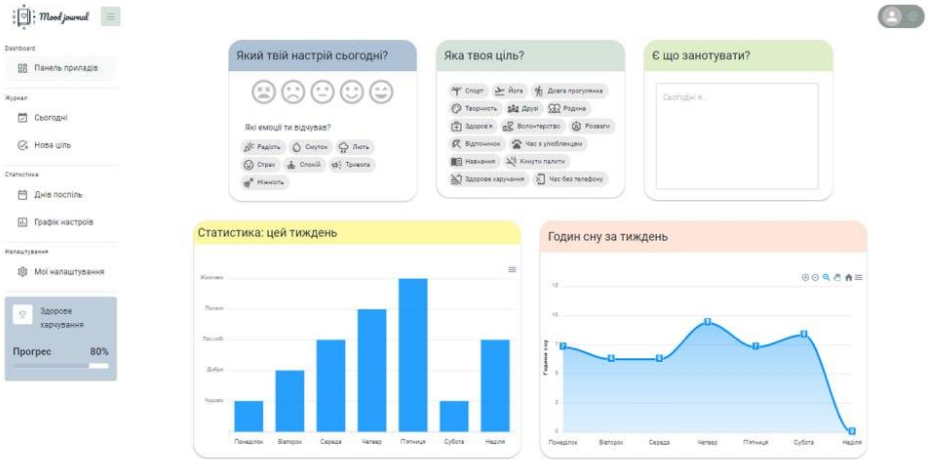
8

Карта сайту



9

Екранні форми



Панель приладів, світла тема

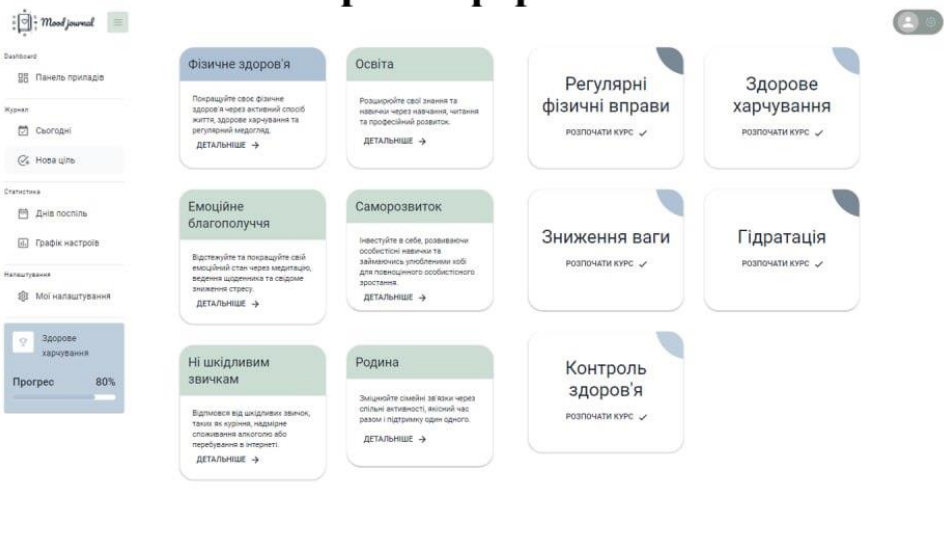
10

Екранні форми

Панель приладів, темна тема

11

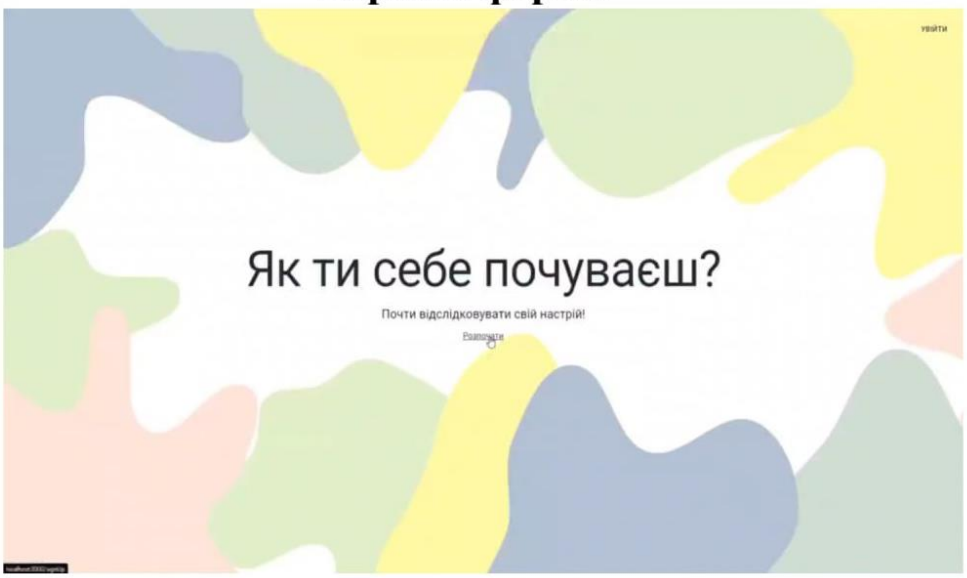
Екранні форми



Нова ціль, світла тема

12

Екранні форми



13

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Зінченко А.Є., Замрій І.В. Визначення додаткових функцій веб-додатку Mood journal трекеру емоцій та настрою для аналізу та самотерапії: Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ», 24 квітня 2024 р., Київ, Державний університет телекомунікацій. Збірник тез. К.: ДУТ, 2024. С. 93-94.
2. Зінченко А.Є., Замрій І.В. Покращення продуктивності та ефективності веб-додатку Mood journal трекеру емоцій та настрою для аналізу та самотерапії: Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ», 24 квітня 2024 р., Київ, Державний університет телекомунікацій. Збірник тез. К.: ДУТ, 2024. С. 227-228.

14

ВИСНОВКИ

1. Проаналізувано вже існуючі застосунки та знайдено три аналоги, а саме MoodTraker, Journey.cloud, eMoods, виявлено їх переваги та недоліки, складено таблицю порівняння.
2. Розроблено технічне завдання, визначено функціональні та нефункціональні вимоги до застосунку.
3. Досліджено інструменти розробки та обрано: JavaScript, React, Redux, MUI для клієнтської частини, Node.js, Knex.js для серверної.
4. Розроблено web-застосунок на основі обраних інструментів, який містить збільшену кількість методів для самоаналізу та самотерапії у вигляді створення кастомних хештегів для пошуку важливих записів, використання геолокації для визначення впливу розташування та постановка цілей.
5. Проведено тестування на кросбраузерність та адаптивність.

15

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

```
const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");
const db = require("../db/dbConfig");

exports.register = async (req, res) => {
  const { email, password } = req.body;
  try {
    const hashedPassword = await bcrypt.hash(password, 10);
    const result = await db("users")
      .insert({
        email: email,
        password: hashedPassword
      })
      .returning(["id", "email"]);
    const token = jwt.sign({ id: result[0].id },
process.env.SECRET_KEY, { expiresIn: "1h" });
    res.status(201).send({
      user: result[0],
      token
    });
  } catch (error) {
    console.error(error);
    res.status(500).send("Something went wrong.");
  }
};

exports.login = async (req, res) => {
  const { email, password } = req.body;
  try {
    const users = await db("users").where({ email: email });
    if (users.length >0) {
      const user = users[0];
      const match = await bcrypt.compare(password,
user.password);
      if (match) {
        const token = jwt.sign({ id: user.id },
process.env.SECRET_KEY, { expiresIn: "1h" });
        res.status(200).send({ token });
      } else {
        res.status(401).send("Authentication failed");
      }
    } else {
      res.status(404).send("User not found");
    }
  } catch (error) {
    console.error(error);
    res.status(500).send("Something went wrong.");
  }
};


const express = require("express");
const app = express();
const cors = require("cors");
const emotionsService =
require("../services/emotionsService");
const authController = require("../controllers/authController");
const moodLevelService =
require("../services/moodLevelService");
const db = require("../db/dbConfig");
const transformTranslations =
require("../utils/transformTranslations");
const auxiliaryGoalService =
require("../services/auxiliaryGoalService");
const entriesService = require("../services/entriesService");
const usersService = require("../services/usersService");
const entryEmotionsService =
require("../services/entryEmotionsService");
const healthService = require("../services/healthService");
const hashtagsService = require("../services/hashtagsService");

app.use(cors());

app.get("/dashboard", async (req, res) => {
  const { userId } = req.query;
  const emotions = await emotionsService.get();
  const moodLevels = await moodLevelService.get();
  const auxiliaryGoal = (await
auxiliaryGoalService.get()).slice(0, 15);
  const todayEntry = (await entriesService.getByToday(userId))
|| null;
  const todayEmotions = todayEntry ? await
entryEmotionsService.getByEntryId({ entry_id: todayEntry.id
}) : null;
  const health = await healthService.get();
  const hashtags = await
hashtagsService.getByUserId({user_id: userId});

  res.json({ emotions, moodLevels, auxiliaryGoal, todayEntry,
todayEmotions, health, hashtags });
});

app.post("/register", authController.register);
app.post("/login", authController.login);

app.get("/translations", async (req, res) => {
  try {
    const translationsFromDB = await
db("translations").select("translate_path", "ua", "en");

    const translationsObject =
transformTranslations(translationsFromDB);

    res.json(translationsObject);
  } catch (err) {
    console.error(err);
    res.status(500).send("Server error");
  }
});

app.post("/create-entries", async (req, res) => {
  try {
    const { userId, moodLevelId, sleep, note, date, emotionIds,
healthIds, medicineIds, productivityIds, hashtags } =
      req.body;

    const isOurUser = await usersService.getById({ id: userId
});

    if (isOurUser) {
      let createEntry;

      if (userId && moodLevelId && date) {
        createEntry = await entriesService.create({
          user_id: userId,
          mood_level_id: moodLevelId,
          date,
          note,
```

```
        sleep: sleep || null
       });
     }

     const newEntry = createEntry && (await
entriesService.getById({ id: createEntry }));

     if (newEntry) {
       await Promise.all(
         emotionIds?.map(async (emotion) => {
           const emotionData = {
             emotion_id: emotion,
             entry_id: newEntry.id
           };
           console.log(emotionData, "emotionData");
           const newEmotionEntry = await
entryEmotionsService.create(emotionData);
           console.log(emotionData, "emotionData");
           console.log(newEmotionEntry, "newEmotionEntry");
         })
       );
     }

     if (hashtags) {
       await Promise.all(
         hashtags?.map(async (hashtag) => {
           const hashtagData = {
             name: hashtag,
             user_id: userId
           };
           const newHashtag = await
hashtagsService.create(hashtagData);
           console.log(newHashtag, "newHashtag");
         })
       );
     }

     res.json({ newEntry });
   } else {
     res.status(401);
   }
 } catch (err) {
   console.error(err);
   res.status(500).send("Server error");
 }
});

app.patch("/entries", async (req, res) => {
 try {
   const { entryId } = req.query;
   const { userId, moodLevelId, sleep, note, date, emotionIds,
healthIds, medicineIds, productivityIds, hashtags } =
     req.body;

   const isOurUser = await usersService.getById({ id: userId
});

   if (isOurUser) {
     let createEntry;

     if (userId && moodLevelId && date) {
       createEntry = await entriesService.updateById({
         id: entryId,
         user_id: userId,
         mood_level_id: moodLevelId,
         date,
         note,
         sleep: sleep || 0
       });
     }

     const newEntry = createEntry && (await
entriesService.getById({ id: entryId }));

     if (newEntry) {
       await Promise.all(
         emotionIds?.map(async (emotion) => {
           const emotionData = {
             emotion_id: emotion,
             entry_id: newEntry.id
           };
           await
entryEmotionsService.updateEntryId(emotionData);
         })
       );
     }

     if (hashtags) {
       await Promise.all(
         hashtags?.map(async (hashtag) => {
           const hashtagData = {
             name: hashtag,
             user_id: userId
           };
           const newHashtag = await
hashtagsService.create(hashtagData);
           console.log(newHashtag, "newHashtag");
         })
       );
     }

     res.json({ newEntry });
   } else {
     res.status(401);
   }
 } catch (err) {
   console.error(err);
   res.status(500).send("Server error");
 }
});

const routes = app;
module.exports = routes;

const dataBaseClient = process.env.DB_CLIENT || "mysql";
const host = process.env.DB_HOST || "localhost";
const dataBasePort = process.env.DB_PORT || 3306;
const databaseName = process.env.DB_NAME;
const databseUser = process.env.DB_USER;
const databasePassword = process.env.DB_PASSWORD;

const db = require("knex")({
 client: dataBaseClient,
 connection: {
   host: host,
   port: dataBasePort,
   database: databaseName,
   user: databseUser,
   password: databasePassword,
   dateStrings: "date"
 },
 pool: {
   min: 0,
   max: 100,
   afterCreate: function (conn, done) {
     conn.query("show processlist  ", function (err) {
       if (err) {
         done(err, conn);
```

```
      } else {
        conn.query("show processlist  ", function (err) {
          done(err, conn);
        });
      }
    });
  }
 },

 acquireConnectionTimeout: 10000
});

module.exports = db;

const transformTranslations = (translations) => {
 const result = {
  ua: {},
  en: {},
 };

 translations.forEach(translation => {
  result.ua[translation.translate_path] = translation.ua;
  result.en[translation.translate_path] = translation.en;

 });

 return result;
};

module.exports = transformTranslations;

const db = require("../db/dbConfig");

const get = async (trx = db) => {
 try {
  const result = await trx("hashtags");
  return result;
 } catch (error) {
  console.log("can not get hashtags");
  throw error;
 }
};

const getById = async (data, trx = db) => {
 try {
  const { id } = data;
  const result = await trx("hashtags").where({ id });
  return result[0];
 } catch (error) {
  console.log("can not getById hashtags");
  throw error;
 }
};

const create = async (data, trx = db) => {
 try {
  const result = await trx("hashtags").insert(data);
  return result[0];
 } catch (error) {
  console.log("can not create hashtags");
  throw error;
 }
};

const updateById = async (data, trx = db) => {
 try {
  const { id } = data;
  const updateData = { ...data };
  delete updateData.id;
  const result = await
trx("hashtags").update(updateData).where({ id });
  return result;
 } catch (error) {
  console.log("Can't update hashtags");
  throw error;
 }
};

const getByUserId = async (data, trx = db) => {
 try {
  const { user_id } = data;
  const result = await trx("hashtags").where({ user_id });
  return result[0];
 } catch (error) {
  console.log("can not getById hashtags");
  throw error;
 }
};

const hashtagsService = { get, create, getById, updateById,
getByUserId };

module.exports = hashtagsService;


import { createRoot } from"react-dom/client";
import { BrowserRouter } from"react-router-dom";
import { Provider } from"react-redux";
import*as serviceWorker from"serviceWorker";
import App from"App";
import store from"./redux/storage";
import config from"./config";


const container = document.getElementById("root");
const root = createRoot(container);
 <Providerstore={store}>
  <BrowserRouterbasename={config.basename}>
   <App/>
  </BrowserRouter>
 </Provider>
);


import Routes from"routes";
import NavigationScroll from"layout/NavigationScroll";
import MainProvider from"provider/MainProvider";
import { useEffect } from"react";
import api from"api";
import { useDispatch } from"react-redux";
import { appSettingsActions }
from"./redux/slices/appSettingsSlice";

const App = () => {
 const dispatch = useDispatch();
 useEffect(() => {
  const fetchTranslations = async () => {
   const result = await api.translations.get();
   dispatch(appSettingsActions.setTranslates(result));
  };

  fetchTranslations();
 });

 return (
  <NavigationScroll>
   <MainProvider>
    <Routes/>
```

```
    </MainProvider>
   </NavigationScroll>
  );
};

exportdefault App;
```

```
import LocaleProvider from"./LocalProvider";
import StyleProvider from"./StyleProvider";

const MainProvider = ({ children }) => {
 return (
   <StyleProvider>
     <LocaleProvider>{children}</LocaleProvider>
   </StyleProvider>
 );
};

exportdefault MainProvider;
```

```
import { IntlProvider } from"react-intl";
import { useSelector } from"react-redux";
import { selectLocale, selectTranslates }
from"../redux/slices/appSettingsSlice";

const LocaleProvider = ({ children }) => {
 const locale = useSelector(selectLocale);
 const translates = useSelector(selectTranslates);

 return (
   <IntlProviderlocale={locale === "ua" ? "uk" :
locale}messages={translates ? translates[locale] : {}}>
     {children}
   </IntlProvider>
 );
};

exportdefault LocaleProvider;
```

```
import { createTheme, CssBaseline, ThemeProvider }
from"@mui/material";
import { useMemo } from"react";
import { useSelector } from'react-redux';
import globalTheme from"../styles/globalTheme";
import { selectTheme } from'../redux/slices/appSettingsSlice';

const StyleProvider = ({ children }) => {
 const style = useSelector(selectTheme);
 const theme = useMemo(() =>
createTheme(globalTheme(style)), [style]);

 return (
   <ThemeProvidertheme={theme}>
     <CssBaseline/>
     {children}
   </ThemeProvider>
 );
};

exportdefault StyleProvider;
```

```
import { configureStore } from"@reduxjs/toolkit";
import { appSettingsReducer } from"./slices/appSettingsSlice";
import { dashboardReducer } from"./slices/dashboardSlice";
import { menuReducer } from"./slices/menuSlice/menuSlice";
import { userDataReducer }
from"./slices/userDataSlice/userDataSlice";

exportdefault configureStore({
 reducer: {
   appSettingsReducer,
   dashboardReducer,
   menuReducer,
   userDataReducer
 }
});
```

```
import { createSlice } from"@reduxjs/toolkit";
import localStorageUtil from"../../utils/localStorageUtil";
import appConst from"constant/appConst";
import themeConst from"constant/themesConst";

const initialState = () => ({
 theme: appConst.DEFAULT_THEME,
 locale: appConst.DEFAULT_LOCALE,
 currentUser: { isAuth: false, id: "" },
 translates: { [appConst.DEFAULT_LOCALE]: {} }
});

const appSettingsSlice = createSlice({
 name: "appSettingsSlice",
 initialState: initialState(),
 reducers: {
   setAppSettings: (state, action) => {
     state = action.payload;
     return state;
   },
   setTheme: (state, action) => {
     state.theme = action.payload;
   },
   setLocale: (state, action) => {
     state.locale = action.payload;
   },
   setTranslates: (state, action) => {
     state.translates = action.payload;
   },
   setCurrentUser: (state, action) => {
     state.currentUser = action.payload;
     state.currentUser.isAuth = true;
   },
   toggleColorMode: (state) => {
     if (state.theme === themeConst.LIGHT_THEME) {
       state.theme = themeConst.DARK_THEME;
       localStorageUtil.setTheme(themeConst.DARK_THEME);
     } elseif (state.theme === themeConst.DARK_THEME) {
       state.theme = themeConst.LIGHT_THEME;
       localStorageUtil.setTheme(themeConst.LIGHT_THEME)
;
     }
   }
 }
});
```

```
exportconst selectTheme = (state) =>
state.appSettingsReducer.theme;
exportconst selectLocale = (state) =>
state.appSettingsReducer.locale;
exportconst selectCurrentUser = (state) =>
state.appSettingsReducer.currentUser;
exportconst selectPermissions = (state) =>
state.appSettingsReducer.currentUser.permissions;
exportconst selectTranslates = (state) =>
state.appSettingsReducer.translates;
```

```
exportconst { reducer: appSettingsReducer, actions:
appSettingsActions } = appSettingsSlice;


const initialState = {
  isOpen: [],
  defaultId: "default",
  opened: true
};

exportconst menuSlice = createSlice({
  name: "menuSlice",
  initialState,
  reducers: {
    menuOpen: (state, action) => {
      state.isOpen = [action.payload];
    },
    setMenu: (state, action) => {
      state.opened = action.payload;
    }
  }
});

exportconst isOpenSelect = (state) =>
state.menuReducer.isOpen;
exportconst defaultIdSelect = (state) =>
state.menuReducer.defaultId;
exportconst openedSelect = (state) =>
state.menuReducer.opened;

exportconst { reducer: menuReducer, actions: menuActions } =
menuSlice;


import { createSlice } from"@reduxjs/toolkit";

exportconst userDataSlice = createSlice({
  name: "userData",
  initialState: { userId: null, moodLevelId: null, note: "",
emotionIds: [], entryId: null, hashtags: [] },
  reducers: {
    updateAllData: (state, action) => {
      return { ...action.payload };
    },
    setCurrentUser: (state, action) => {
      state.userId = action.payload;
    },
    setEntryId: (state, action) => {
      state.entryId = action.payload;
    },
    setHashtags: (state, action) => {
      state.hashtags = action.payload;
    }
  }
});

exportconst { reducer: userDataReducer, actions:
userDataActions } = userDataSlice;


const colorsPalette = (mode) => ({
  ...(mode === "dark"
    ? {
        powderBlue: {
          DEFAULT: "#afc2d5",
          100: "#1b2733",
          200: "#354e66",
          300: "#507498",
          400: "#7c9bba",
          500: "#afc2d5",
          600: "#bfcedd",
          700: "#cfdae6",
          800: "#dfe7ee",
          900: "#eff3f7"
        },
        honeydew: {
          DEFAULT: "#ccddd3",
          100: "#223329",
          200: "#446652",
          300: "#66987b",
          400: "#98baa6",
          500: "#ccddd3",
          600: "#d5e3db",
          700: "#e0eae4",
          800: "#eaf1ed",
          900: "#f5f8f6"
        },
        nyanza: {
          DEFAULT: "#dfefca",
          100: "#2f4414",
          200: "#5e8728",
          300: "#8cc641",
          400: "#b5da84",
          500: "#dfefca",
          600: "#e4f2d3",
          700: "#ebf5de",
          800: "#f2f8e9",
          900: "#f8fcf4"
        },
        vanilla: {
          DEFAULT: "#fff9a5",
          100: "#544e00",
          200: "#a79c00",
          300: "#fbea00",
          400: "#fff350",
          500: "#fff9a5",
          600: "#fffab6",
          700: "#fffbc8",
          800: "#fffdda",
          900: "#fffeed"
        },
        champagne_pink: {
          DEFAULT: "#ffe5d9",
          100: "#5f1e00",
          200: "#be3c00",
          300: "#ff651e",
          400: "#ffa67c",
          500: "#ffe5d9",
          600: "#ffebe2",
          700: "#fff0ea",
          800: "#fff5f1",
          900: "#fffaf8"
        },
        background: "#121212"
      }
    : {
        powderBlue: {
          DEFAULT: "#afc2d5",
          100: "#1b2733",
          200: "#354e66",
          300: "#507498",
          400: "#7c9bba",
          500: "#afc2d5",
          600: "#bfcedd",
          700: "#cfdae6",
          800: "#dfe7ee",
          900: "#eff3f7"
```

```
    },
    honeydew: {
      DEFAULT: "#ccddd3",
      100: "#223329",
      200: "#446652",
      300: "#66987b",
      400: "#98baa6",
      500: "#ccddd3",
      600: "#d5e3db",
      700: "#e0eae4",
      800: "#eaf1ed",
      900: "#f5f8f6"
    },
    nyanza: {
      DEFAULT: "#dfefca",
      100: "#2f4414",
      200: "#5e8728",
      300: "#8cc641",
      400: "#b5da84",
      500: "#dfefca",
      600: "#e4f2d3",
      700: "#ebf5de",
      800: "#f2f8e9",
      900: "#f8fcf4"
    },
    vanilla: {
      DEFAULT: "#fff9a5",
      100: "#544e00",
      200: "#a79c00",
      300: "#fbea00",
      400: "#fff350",
      500: "#fff9a5",
      600: "#fffab6",
      700: "#fffbc8",
      800: "#fffdda",
      900: "#fffeed"
    },
    champagne_pink: {
      DEFAULT: "#ffe5d9",
      100: "#5f1e00",
      200: "#be3c00",
      300: "#ff651e",
      400: "#ffa67c",
      500: "#ffe5d9",
      600: "#ffebe2",
      700: "#fff0ea",
      800: "#fff5f1",
      900: "#fffaf8"
    },
    background: "rgba(255, 255, 255, 1)"
  })
});

const globalTheme = (mode) => {
  const colors = colorsPalette(mode);

  return {
    palette: {
      mode: mode,
      ...(mode === "dark"
        ? {
            background: {
              default: colors.background
            },
            primary: {
              main: colors.powderBlue.DEFAULT,
              light: colors.powderBlue[600],
              dark: colors.powderBlue[200],
              contrast: colors.powderBlue[300],
              extraLight: colors.powderBlue[800],
              extraDark: colors.powderBlue[100]
            },
            secondary: {
              main: colors.honeydew[300]
            },
            success: {
              main: colors.nyanza[300]
            },
            text: {
              primary: colors.powderBlue[900]
            },
            champagnePink: {
              main: colors.champagne_pink[300],
              light: colors.champagne_pink[700],
              dark: colors.champagne_pink[400],
              contrastText: colors.powderBlue[100]
            },
            vanilla: {
              main: colors.vanilla[200],
              light: colors.champagne_pink[700],
              dark: colors.champagne_pink[400],
              contrastText: colors.powderBlue[100]
            }
          }
        : {
            background: {
              default: colors.background
            },
            primary: {
              main: colors.powderBlue.DEFAULT
            },
            secondary: {
              main: colors.honeydew.DEFAULT
            },
            success: {
              main: colors.nyanza.DEFAULT
            },
            text: {
              primary: colors.powderBlue[100]
            },
            champagnePink: {
              main: colors.champagne_pink.DEFAULT,
              light: colors.champagne_pink[700],
              dark: colors.champagne_pink[400],
              contrastText: colors.powderBlue[100]
            },
            vanilla: {
              main: colors.vanilla.DEFAULT,
              light: colors.champagne_pink[700],
              dark: colors.champagne_pink[400],
              contrastText: colors.powderBlue[100]
            }
          })
    },
    components: {
      MuiButton: {
        styleOverrides: {
          text: {
            color: colors.powderBlue[100]
          }
        }
      },
      MuiLink: {
        styleOverrides: {
          root: {
            color: colors.powderBlue[100],
            textDecorationColor: colors.powderBlue[100]
          }
```

```
      }
    }
  }
 };
};

exportdefault globalTheme;


const SERVER_HOST =
process.env.REACT_APP_SERVER_HOST ||
"http://localhost";
const SERVER_PORT =
process.env.REACT_APP_SERVER_PORT || ":3001";
const SERVER_API_PATH =
process.env.REACT_APP_SERVER_API_PATH || "/api";

function wait(delay) {
 returnnew Promise((resolve) => {
  setTimeout(resolve, delay);
 });
}

function request(url, method = "GET", data = null) {
 const options = { method };

 if (data) {
  options.body = JSON.stringify(data);
  options.headers = {
   "Content-Type": "application/json; charset=UTF-8"
  };
 }

 return wait(300)
  .then(() => fetch(SERVER_HOST + SERVER_PORT +
SERVER_API_PATH + url, options))
  .then((response) => {
   if (!response.ok) {
    thrownew Error();
   }

   return response.json();
  });
}

exportconst client = {
 get: (url) => request(url),
 post: (url, data) => request(url, "POST", data),
 patch: (url, data) => request(url, "PATCH", data),
 delete: (url) => request(url, "DELETE")
};

import PropTypes from"prop-types";
import { motion } from"framer-motion";

const NavMotion = ({ children }) => {
 const motionVariants = {
  initial: {
   opacity: 0,
   scale: 0.99
  },
  in: {
   opacity: 1,
   scale: 1
  },
  out: {
   opacity: 0,
   scale: 1.01
  }
 };

 const motionTransition = {
  type: "tween",
  ease: "anticipate",
  duration: 0.4
 };

 return (
  <motion.divinitial="initial"animate="in"exit="out"variants=
{motionVariants}transition={motionTransition}>
   {children}
  </motion.div>
 );
};

NavMotion.propTypes = {
 children: PropTypes.node
};

exportdefault NavMotion;


import PropTypes from"prop-types";
import { useEffect } from"react";
import { useLocation } from"react-router-dom";

const NavigationScroll = ({ children }) => {
 const location = useLocation();
 const { pathname } = location;

 useEffect(() => {
  window.scrollTo({
   top: 0,
   left: 0,
   behavior: "smooth"
  });
 }, [pathname]);

 return children || null;
};

NavigationScroll.propTypes = {
 children: PropTypes.node
};

exportdefault NavigationScroll;


import { useDispatch, useSelector } from"react-redux";
import { Outlet } from"react-router-dom";
import { styled, useTheme } from"@mui/material/styles";
import { AppBar, Box, CssBaseline, Toolbar, useMediaQuery
} from"@mui/material";

import Breadcrumbs from"ui-
component/extended/Breadcrumbs";
import Header from"./Header";
import Sidebar from"./Sidebar";
import navigation from"menu-items";
import { drawerWidth } from"store/constant";

import { IconChevronRight } from"@tabler/icons-react";
import { menuActions, openedSelect }
from"../../redux/slices/menuSlice/menuSlice";

const Main = styled("main", { shouldForwardProp: (prop) =>
prop !== "open"&& prop !== "theme" })(({ theme, open }) =>
({
```

```
  ...theme.typography.mainContent,
  borderBottomLeftRadius: 0,
  borderBottomRightRadius: 0,
  transition: theme.transitions.create(
   "margin",
   open
    ? {
       easing: theme.transitions.easing.easeOut,
       duration: theme.transitions.duration.enteringScreen
      }
    : {
       easing: theme.transitions.easing.sharp,
       duration: theme.transitions.duration.leavingScreen
      }
  ),
  [theme.breakpoints.up("md")]: {
   marginLeft: open ? 0 : -(drawerWidth - 20),
   width: `calc(100% - ${drawerWidth}px)`
  },
  [theme.breakpoints.down("md")]: {
   marginLeft: "20px",
   width: `calc(100% - ${drawerWidth}px)`,
   padding: "16px"
  },
  [theme.breakpoints.down("sm")]: {
   marginLeft: "10px",
   width: `calc(100% - ${drawerWidth}px)`,
   padding: "16px",
   marginRight: "10px"
  }
}));


const MainLayout = () => {
  const theme = useTheme();
  const matchDownMd =
useMediaQuery(theme.breakpoints.down("md"));
  const leftDrawerOpened = useSelector(openedSelect);
  const dispatch = useDispatch();
  const handleLeftDrawerToggle = () => {
   dispatch(menuActions.setMenu(!leftDrawerOpened));
  };

  return (
   <Boxsx={{ display: "flex" }}>
    <CssBaseline/>
    <AppBar
     enableColorOnDark
     position="fixed"
     color="inherit"
     elevation={0}
     sx={{
      bgcolor: theme.palette.background.default,
      transition: leftDrawerOpened ?
theme.transitions.create("width") : "none"
     }}
    >
     <Toolbar>
      <HeaderhandleLeftDrawerToggle={handleLeftDrawerT
oggle}/>
     </Toolbar>
    </AppBar>

    <SidebardrawerOpen={!matchDownMd ?
leftDrawerOpened :
!leftDrawerOpened}drawerToggle={handleLeftDrawerToggle}
/>

    <Maintheme={theme}open={leftDrawerOpened}>
     <Breadcrumbsseparator={IconChevronRight}navigation=
{navigation}icontitlerightAlign/>
     <Outlet/>
    </Main>
   </Box>
  );
};

exportdefault MainLayout;


import PropTypes from"prop-types";

import { useTheme } from"@mui/material/styles";
import { Box, Drawer, Stack, useMediaQuery }
from"@mui/material";

import PerfectScrollbar from"react-perfect-scrollbar";
import { BrowserView, MobileView } from"react-device-
detect";

import MenuList from"./MenuList";
import LogoSection from"../LogoSection";
import MenuCard from"./MenuCard";
import { drawerWidth } from"store/constant";


const Sidebar = ({ drawerOpen, drawerToggle, window }) => {
  const theme = useTheme();
  const matchUpMd =
useMediaQuery(theme.breakpoints.up("md"));

  const drawer = (
   <>
    <Boxsx={{ display: { xs: "block", md: "none" } }}>
     <Boxsx={{ display: "flex", p: 2, mx: "auto" }}>
      <LogoSection/>
     </Box>
    </Box>
    <BrowserView>
     <PerfectScrollbar
      component="div"
      style={{
       height: !matchUpMd ? "calc(100vh - 56px)" :
"calc(100vh - 88px)",
       paddingLeft: "16px",
       paddingRight: "16px"
      }}
     >
      <MenuList/>
      <MenuCard/>
      <Stackdirection="row"justifyContent="center"sx={{ mb:
2 }}></Stack>
     </PerfectScrollbar>
    </BrowserView>
    <MobileView>
     <Boxsx={{ px: 2 }}>
      <MenuList/>
      <MenuCard/>
      <Stackdirection="row"justifyContent="center"sx={{ mb:
2 }}></Stack>
     </Box>
    </MobileView>
   </>
  );

  const container = window !== undefined ? () =>
window.document.body : undefined;
```

```
  return (
    <Boxcomponent="nav"sx={{ flexShrink: { md: 0 }, width:
matchUpMd ? drawerWidth : "auto" }}aria-label="mailbox
folders">
      <Drawer
        container={container}
        variant={matchUpMd ? "persistent" : "temporary"}
        anchor="left"
        open={drawerOpen}
        onClose={drawerToggle}
        sx={{
          "& .MuiDrawer-paper": {
            width: drawerWidth,
            background: theme.palette.background.default,
            color: theme.palette.text.primary,
            borderRight: "none",
            [theme.breakpoints.up("md")]: {
              top: "88px"
            }
          }
        }}
        ModalProps={{ keepMounted: true }}
        color="inherit"
      >
        {drawer}
      </Drawer>
    </Box>
  );
};

Sidebar.propTypes = {
  drawerOpen: PropTypes.bool,
  drawerToggle: PropTypes.func,
  window: PropTypes.object
};

exportdefault Sidebar;


import DashboardOutlinedIcon from"@mui/icons-
material/DashboardOutlined";
const icons = { DashboardOutlinedIcon };

const dashboard = {
  id: "dashboard",
  title: "Dashboard",
  type: "group",
  children: [
    {
      id: "default",
      title: "Панель приладів",
      type: "item",
      url: "/",
      icon: icons.DashboardOutlinedIcon,
      breadcrumbs: false
    }
  ]
};

exportdefault dashboard;


import EventAvailableIcon from"@mui/icons-
material/EventAvailable";
import AddTaskIcon from"@mui/icons-material/AddTask";
const icons = {
  EventAvailableIcon,
  AddTaskIcon
};

const pages = {
  id: "pages",
  title: "Журнал",
  type: "group",
  children: [
    {
      id: "today",
      title: "Сьогодні",
      type: "item",
      url: "/today",
      icon: icons.EventAvailableIcon,
      breadcrumbs: false
    },
    {
      id: "target",
      title: "Нова ціль",
      type: "item",
      url: "/target",
      icon: icons.AddTaskIcon,
      breadcrumbs: false
    }
  ]
};

exportdefault pages;


import DateRangeOutlinedIcon from"@mui/icons-
material/DateRangeOutlined";
import InsertChartOutlinedIcon from"@mui/icons-
material/InsertChartOutlined";
import EmojiEventsOutlinedIcon from"@mui/icons-
material/EmojiEventsOutlined";

const icons = {
  DateRangeOutlinedIcon,
  InsertChartOutlinedIcon,
  EmojiEventsOutlinedIcon
};

const statistic = {
  id: "statistic",
  title: "Статистика",
  type: "group",
  children: [
    {
      id: "daysInARow",
      title: "Днів поспіль",
      type: "item",
      url: "/daysInARow",
      icon: icons.DateRangeOutlinedIcon,
      breadcrumbs: false
    },
    {
      id: "moodsStatistic",
      title: "Графік настроїв",
      type: "item",
      url: "/moods-statistic",
      icon: icons.InsertChartOutlinedIcon,
      breadcrumbs: false
    }
  ]
};

exportdefault statistic;
```

```
import SettingsOutlinedIcon from"@mui/icons-
material/SettingsOutlined";

const icons = { SettingsOutlinedIcon };

const other = {
 id: "sample-docs-roadmap",
 type: "group",
 title: "Налаштування",
 children: [
  {
   id: "sample-page",
   title: "Мої налаштування",
   type: "item",
   url: "/sample-page",
   icon: icons.SettingsOutlinedIcon,
   breadcrumbs: false
  }
 ]
};

exportdefault other;
```

```
import dashboard from"./dashboard";
import pages from"./pages";
import statistic from"./statistic";
import other from"./other";

const menuItems = {
 items: [dashboard, pages, statistic, other]
};

exportdefault menuItems;
```

```
import LogInForm from"components/AuthPage/LogInForm";
import SignUpForm
from"components/AuthPage/SignUpForm";
import InitPage from"components/InitPage";

const AuthenticationRoutes = [
 { id: 1, path: "/init", element: <InitPage/>, hasPermission: true
},
 { id: 2, path: "/logIn", element: <LogInForm/>,
hasPermission: true },
 { id: 3, path: "/signUp", element: <SignUpForm/>,
hasPermission: true }
];

exportdefault AuthenticationRoutes;
```

```
import { lazy } from"react";

import MainLayout from"layout/MainLayout";
import Loadable from"ui-component/Loadable";
import Today from"views/today";
import Target from"views/target";
import DaysInARow from"views/daysInARow";
import MoodsStatistic from"views/moodsStatistic";

const DashboardDefault = Loadable(lazy(()
=>import("views/dashboard/Default")));

const MainRoutes = {
 path: "/",
 element: <MainLayout/>,
 children: [
  {
   path: "/",
   element: <DashboardDefault/>
  },
  {
   path: "/today",
   element: <Today/>
  },
  {
   path: "target",
   element: <Target/>
  },
  {
   path: "daysInARow",
   element: <DaysInARow/>
  },
  {
   path: "/moods-statistic",
   element: <MoodsStatistic/>
  }
 ]
};

exportdefault MainRoutes;
```

```
import { useRoutes } from"react-router-dom";

import MainRoutes from"./MainRoutes";
import AuthenticationRoutes from"./AuthenticationRoutes";

exportdefaultfunction ThemeRoutes() {
 return useRoutes([MainRoutes, ...AuthenticationRoutes]);
}
```

```
import { useMemo } from"react";
import { useSelector } from"react-redux";
import { dashboardDataSelect }
from"../../../redux/slices/dashboardSlice";
import getIconByName from"utils/getIconByName";
import { useIntl } from"react-intl";

const useGetEmotionsData = () => {
 const { emotions } = useSelector(dashboardDataSelect);
 const intl = useIntl();
 return useMemo(
  () =>
   emotions
    ? emotions.map(({ id, name }) => ({
      id,
      label: intl.formatMessage({ id:
`emotion.${name.toLowerCase()}` }),
      icon: getIconByName(name)
     }))
    : [],
  [emotions, intl]
 );
};

exportdefault useGetEmotionsData;
```

```
import { useMemo } from"react";
import { useSelector } from"react-redux";
import { dashboardDataSelect }
from"../../../redux/slices/dashboardSlice";
import getIconByName from"utils/getIconByName";
import { useIntl } from"react-intl";
```

```
const useGetHealthData = () => {
  const { health } = useSelector(dashboardDataSelect);
  const intl = useIntl();
  return useMemo(
    () =>
      health
        ? health
            .map(({ id, name }) => ({
              id,
              label: intl.formatMessage({ id: name }),
              icon: getIconByName(name)
            }))
            .slice(0, 7)
        : [],
    [health, intl]
  );
};

exportdefault useGetHealthData;
```

```
import { useDispatch, useSelector } from"react-redux";
import { selectCurrentUser }
from"../../../../redux/slices/appSettingsSlice";
import { dashboardDataSelect }
from"../../../../redux/slices/dashboardSlice";
import { useEffect, useMemo, useState } from"react";
import { userDataActions }
from"../../../../redux/slices/userDataSlice/userDataSlice";

const useGetEntryData = (isLoading) => {
  const dispatch = useDispatch();
  const { todayEntry, todayEmotions } =
useSelector(dashboardDataSelect);
  const currentUser = useSelector(selectCurrentUser);
  const defaultEntry = useMemo(
    () => ({
      userId: currentUser?.id,
      moodLevelId: null,
      note: "",
      emotionIds: [],
      sleep: null,
      healthIds: [],
      canUpdate: false
    }),
    [currentUser.id]
  );
  const entryFromServer = useMemo(
    () =>
      todayEntry
        ? {
            userId: currentUser?.id,
            moodLevelId: todayEntry?.mood_level_id || null,
            note: todayEntry?.note || "",
            emotionIds: todayEmotions.map(({ emotion_id }) =>
emotion_id) || null,
            emotions: todayEmotions || null,
            sleep: todayEntry?.sleep || 0,
            healthIds: todayEntry?.healthIds || [],
            canUpdate: true
          }
        : null,
    [currentUser.id, todayEntry, todayEmotions]
  );
  const [newEntry, setNewEntry] = useState(null);

  useEffect(() => {
    if (!isLoading && entryFromServer) {
      dispatch(userDataActions.updateAllData(entryFromServer)
);
      setNewEntry(entryFromServer);
      dispatch(userDataActions.setEntryId(todayEntry?.id));
    } else {
      setNewEntry(defaultEntry);
    }
  }, [entryFromServer, defaultEntry, dispatch, isLoading,
todayEntry?.id]);

  return [newEntry, setNewEntry];
};

exportdefault useGetEntryData;
```

```
import { Chip } from"@mui/material";
import { useSelector } from"react-redux";
import { dashboardDataSelect }
from"../../../../redux/slices/dashboardSlice";
import { useMemo } from"react";
import getIconByName from"utils/getIconByName";
import { useIntl } from"react-intl";

const useGetGoalsData = () => {
  const { auxiliaryGoal } = useSelector(dashboardDataSelect);
  const intl = useIntl();

  return useMemo(
    () =>
      auxiliaryGoal
        ? auxiliaryGoal.map(({ id, name }) => (
            <Chiplabel={intl.formatMessage({ id: name
})}key={id}icon={getIconByName(name)}clickable/>
          ))
        : [],
    [auxiliaryGoal, intl]
  );
};

exportdefault useGetGoalsData;
```

```
import { useSelector } from"react-redux";
import { dashboardDataSelect }
from"../../../../redux/slices/dashboardSlice";
import { useMemo } from"react";
import getIconByName from"utils/getIconByName";

const useGetMoodSliderData = () => {
  const { moodLevels } = useSelector(dashboardDataSelect);

  return useMemo(() => {
    const res = {};

    moodLevels?.forEach((item) => {
      res[item?.id] = {
        value: `${item.id}`,
        label: item.name,
        icon: getIconByName(item.name, { sx: { width: "60px",
height: "60px" } }),
      };
    });

    return res;
  }, [moodLevels]);
};

exportdefault useGetMoodSliderData;
```

```
import { useEffect, useMemo, useState } from"react";

import { Grid, SpeedDial, SpeedDialAction, SpeedDialIcon }
from"@mui/material";

import TotalGrowthBarChart from"./TotalGrowthBarChart";
import { gridSpacing } from"store/constant";
import TodayMoodCard from"./TodayMoodCard";
import TodayNotesCard from"./TodayNotesCard";
import TodayTargetCard from"./TodayTargetCard";
import DaysInRow from"./DaysInRow";
import SaveOutlinedIcon from"@mui/icons-
material/SaveOutlined";
import { dashboardActions }
from"../../../redux/slices/dashboardSlice";
import { useDispatch, useSelector } from"react-redux";
import api from"api";
import { selectCurrentUser }
from"../../../redux/slices/appSettingsSlice";
import DeleteOutlineOutlinedIcon from"@mui/icons-
material/DeleteOutlineOutlined";
import { formatDataBaseDateTime }
from"../../../utils/formatDataBaseDateTime";
import useGetEntryData from"./hooks/useGetEntryData";
import CreateOutlinedIcon from"@mui/icons-
material/CreateOutlined";
import { entryIdSelect }
from"../../../redux/slices/userDataSlice/userDataSelect";

const Dashboard = () => {
  const dispatch = useDispatch();
  const [isLoading, setLoading] = useState(true);
  const currentUser = useSelector(selectCurrentUser);
  const entryId = useSelector(entryIdSelect);

  useEffect(() => {
    const fetchData = async () => {
      setLoading(true);
      dispatch(dashboardActions.fetchDataStart());
      try {
        const response = await
api.dashboard.getData(currentUser.id);
        dispatch(dashboardActions.fetchDataSuccess(response));
        setLoading(false);
      } catch (error) {
        dispatch(dashboardActions.fetchDataFailure(error.toStrin
g()));
        setLoading(false);
      }
    };

    fetchData();
  }, [currentUser.id, dispatch]);

  const [newEntry, setNewEntry] =
useGetEntryData(isLoading);
  const [hashtags, setHashtags] = useState([]);

  const actions = useMemo(
    () => [
      {
        icon: <SaveOutlinedIcon/>,
        name: "Save",
        visible: newEntry?.canUpdate ? false : true,
        onClick: async () => {
          try {
            const saveDate = { ...newEntry, date:
formatDataBaseDateTime(new Date()), hashtags };
            const result = await
api.dashboard.createEntry(saveDate);
            console.log(result);
          } catch (error) {
            console.error(error);
          }
        }
      },
      {
        icon: <CreateOutlinedIcon/>,
        name: "Update",
        visible: newEntry?.canUpdate ? true : false,
        onClick: async () => {
          try {
            const updateData = { ...newEntry, entryId };
            const saveDate = { ...updateData, date:
formatDataBaseDateTime(new Date()), hashtags };
            const result = await
api.dashboard.updateEntry(saveDate);
            console.log(result);
          } catch (error) {
            console.error(error);
          }
        }
      },
      { icon: <DeleteOutlineOutlinedIcon/>, name: "Clean",
visible: true, onClick: () => {} }
    ],
    [entryId, hashtags, newEntry]
  );

  return (
    <Gridcontainerspacing={gridSpacing}justifyContent="cente
r">
      <Griditemxs={12}>
        <Gridcontainerspacing={gridSpacing}justifyContent="ce
nter"sx={{ marginTop: "40px" }}>
          <Griditemlg={3}md={6}sm={6}xs={12}>
            <TodayMoodCardisLoading={isLoading}newEntry={n
ewEntry}setNewEntry={setNewEntry}/>
          </Grid>
          <Griditemlg={3}md={6}sm={6}xs={12}>
            <TodayTargetCardisLoading={isLoading}newEntry={
newEntry}setNewEntry={setNewEntry}/>
          </Grid>
          <Griditemlg={3}md={6}sm={6}xs={12}>
            <TodayNotesCard
              isLoading={isLoading}
              newEntry={newEntry}
              setNewEntry={setNewEntry}
              setHashtags={setHashtags}
              hashtags={hashtags}
            />
          </Grid>
        </Grid>
      </Grid>
      <Griditemxs={12}>
        <Gridcontainerspacing={gridSpacing}justifyContent="ce
nter">
          <Griditemxs={12}md={5}>
            <TotalGrowthBarChartisLoading={isLoading}/>
          </Grid>
          <Griditemxs={12}md={5}>
            <DaysInRowisLoading={isLoading}/>
          </Grid>
        </Grid>
      </Grid>
      {newEntry?.moodLevelId && (
```

```
      <SpeedDialariaLabel="SpeedDial basic example"sx={{
position: "absolute", top: 200, right: 80
}}icon={<SpeedDialIcon/>}>
        {actions.map(
          (action) =>
            action?.visible && (
              <SpeedDialActionkey={action.name}icon={action.ic
on}tooltipTitle={action.name}onClick={action.onClick}/>
            )
        )}
      </SpeedDial>
    )}
  </Grid>
 );
};

exportdefault Dashboard;
```