КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

інформаційних технологій

(кафедра)

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО АТЕСТАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО РІВНЯ «БАКАЛАВР»

на тему: «Фронт-енд розробка автоматизованої системи оцінки психологічного стану»

ПАНАГОДА АРАЧЧІГЕ НІКІТА ДІКСОНОВИЧ

(прізвище, ім’я та по батькові студента повністю)

Київ 2024 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

інформаційних технологій

(кафедра)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТ
к.т.н., доцент Гончаренко Т. А.

_____
„___” _____ 2024 року

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО АТЕСТАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО РІВНЯ «БАКАЛАВР»

на тему: «Фронт-енд розробка автоматизованої системи оцінки психологічного стану»

Виконав: студент 4-го курсу, групи КНс-21

Спеціальності: 122 «Комп`ютерні науки»

Спеціалізація: «Інформаційні управляючі системи і технології»
(шифр і назва напряму підготовки, спеціальності)

Панагода Араччіге Н. Д.
(прізвище та ініціали)

Керівник к.т.н., доц. Горда О. В.
(прізвище та ініціали)

Рецензент к.т.н., доц. Шабала Є. Є.
(прізвище та ініціали)

Київ, 2024 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: автоматизації і інформаційних технологій
Кафедра: інформаційних технологій
Освітній рівень: «бакалавр» за ОП
Спеціальність: 122 «Комп`ютерні науки»
Спеціалізація: Інформаційні управляючі системи і технології

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТ
к.т.н., доцент Гончаренко Т. А.

_____
„12” лютого 2024 року

З А В Д А Н Н Я
ДО ВИКОНАННЯ АТЕСТАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО РІВНЯ «БАКАЛАВР»

Панагода Араччіге Нікіта Діксонович
Тема роботи: Фронт-енд розробка автоматизованої системи оцінки психологічного стану
затверджена наказом ректора КНУБА № 2650/2 від «12» лютого 2024 р.
2. Керівник роботи: Горда Олена Володимирівна, к.т.н, доцент кафедри інформаційних технологій проектування та прикладної математики
3. Строк подання студентом роботи до захисту:
4. Зміст пояснювальної записки за розділами:
Р.1. Аналіз предметної області та постановка задачі
Р.2. Проєктування програмного забезпечення
Р.3. Розробка програмного забезпечення
Р.4. Дизайн інтерфейсу користувача
5. Інформаційні слайди:
С.1. Аналіз предметної області
С.2. Вибір технології для реалізації проєкту
С.3. Проектування програмного забезпечення
С.4. Проєктування локалізації додатку
С.5. Розробка програмного забезпечення

С.6. Діаграма компонентів

С.7. Схема потоків даних «Sign Up » компоненту

С.8. Дизайн інтерфейсу користувача

С.9. Адаптивний дизайн та доступність

6. Календарний план виконання атестаційної випускної роботи

Види робіт та їх зміст	Дата виконання
Р.1. Аналіз предметної області та постановка задачі	Лютий 2024 р.
Р.2. Проєктування програмного забезпечення	Березень 2024 р.
Р.3. Розробка програмного забезпечення	Квітень 2024 р.
Р.4. Дизайн інтерфейсу користувача	Травень 2024 р.
Остаточне оформлення роботи	Травень 2024 р.
Направлення роботи на рецензування	Червень 2024 р.
Попередній захист роботи на кафедрі	Червень 2024 р.

7. Консультанти розділів атестаційної випускної роботи

Розділ	Прізвище, ініціали та посада консультанта, представника комісії	дата	підпис
Впровадження системи в експлуатацію	к.т.н., доц. Рябчун Ю.В.		
Прийом програмного продукту	к.т.н., доц. Шабала Є.Є.		

8. Дата видачі завдання: 12 лютого 2024 р.

Керівник _____ Горда О. В.
(підпис) (прізвище та ініціали)

Бакалавр _____ Панагода Н. Д.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Панагода Араччіге Н. Д. Фронт-енд розробка автоматизованої системи оцінки психологічного стану

Атестаційна випускна робота бакалавра за спеціальністю: 122 «Комп’ютерні науки», спеціалізація: «Інформаційні управляючі системи і технології». – Київський національний університет будівництва та архітектури. – Київ, 2024.

Робота присвячена розробці клієнтської частини автоматизованої системи оцінки психологічного стану людей, які постраждали внаслідок бойових дій. Реалізація системи включає проєктування, розробку програмного забезпечення та дизайн інтерфейсу користувача з використанням сучасних технологій.

Ключові слова: психологічний стан, діагностика, автоматизована система, фронт-енд, React.js, GraphQL, Vite, PNPM, i18next, JavaScript, TypeScript.

SUMMARY

Panahoda Arachchihe N. D. Front-end development of the automated psychological assessment system.

Bachelor's thesis in the specialty: 122 "Computer Science", specialization: "Information managing systems and technologies". – Kyiv National University of Construction and Architecture. – Kyiv, 2024.

The thesis is dedicated to the development of the client part of the automated system for assessing the mental state of people affected by hostilities. The implementation of the system includes design, software development, and user interface design using modern technologies.

Keywords: mental state, diagnostics, automated system, front-end, React.js, GraphQL, Vite, PNPM, i18next, react-hook-form, JavaScript, TypeScript.

ЗМІСТ

Перелік умовних позначень

UI – User Interface

UX – User Experience

API – Application Programming Interface

AES – Advanced Encryption Standard

RSA – Rivest-Shamir-Adleman

ECC – Elliptic Curve Cryptography

DOM – Document Object Model

JS – JavaScript

TS – TypeScript

CSS – Cascading Style Sheets

NPM – Node Package Manager

PNPM – Performant Node Package Manager

CRA – Create React App

HTML – Hypertext Markup Language

HTTP – Hypertext Transfer Protocol

IDE – Integrated Development Environment

JSON – JavaScript Object Notation

VSCode – Visual Studio Code

I18n – Internationalization

JWT – JSON Web Token

REST – Representational State Transfer

UML – Unified Modeling Language

URL – Uniform Resource Locator

Вступ

Актуальність дослідження. В сучасному світі зростає поширеність психічних захворювань. Після повномасштабного вторгнення РФ в Україну чисельність людей, страждаючих від психічних хвороб, стрімко зросла через пережиті ними травматичні події. Запорукою вдалого лікування психічних захворювань є рання діагностика і виявлення цих самих захворювань. Щоб полегшити процес діагностики і зробити його більш доступним, доцільно розробити автоматизовану систему оцінки психічного стану людини. Дана система складається з наступних компонентів: клієнтська частина, серверна частина та база даних. Кожен з цих компонентів являється невід'ємною частиною інформаційної системи та не може бути спростований. Саме тому розробка фронт-енд частини даної системи є актуальною темою дослідження, оскільки вона може сприяти полегшенню процесу діагностики і зробить його більш доступним. Це в свою чергу допоможе вчасно виявляти проблеми з психічним станом людини, що збільшить ймовірність одужання пацієнта.

Мета дослідження. Розробка фронт-енд системи оцінки психологічного стану людей, які постраждали від бойових дій або від інших психологічно-травмуючих подій. Після тестування користувач побачить вірогідність різноманітних психічних порушень та рекомендації, які потрібно вжити для їх усунення. Система буде складатися з тесту, який буде включати питання про симптоми і пережиті події. На основі отриманих даних буде формуватися оцінка поточного стану особи та надаватись відповідні рекомендації. Якщо система виявить, що людина має високий ризик розвитку психічного захворювання, їй дадуть рекомендації подальшого обстеження у психіатра або психолога.

Об'єкт дослідження. Клієнтське програмне забезпечення автоматизованої системи оцінки психічного стану, яке буде відповідати за весь функціонал системи, включаючи реєстрацію, проходження тесту та надання відповідних результатів та рекомендацій користувачу.

Предмет дослідження. Розробка інтерфейсу користувача, включаючи створення дизайну та впровадження програмного забезпечення для клієнтів. Цей предмет також визначає дослідження взаємодії з серверною частиною проєкту.

Методи дослідження: методи системного аналізу, моделювання, об’єктно орієнтованого програмування, розробки та впровадження веб-систем.

Практична значимість. Результати дослідження можуть бути використані для розробки та реалізації ефективних систем оцінки психічного стану людини. Така система буде сприяти швидшому одужанню постраждалих людей від військових дій та інших психічних травм, та буде сприяти їхній подальшій адаптації у суспільстві.

Короткий зміст розділів:

У першому розділі частково описана загальна та теоретична інформація щодо відомостей, концепцій та проблематики розробки фронт-енд частини автоматизованої системи оцінки психічного стану. Описується постановка та аналіз проблеми, дерево цілей, встановлюються вимоги та особливості проєктування системи, аналізуються існуючі розробки автоматизованих систем оцінки психічного стану, проводиться аналіз технологій розробки клієнтського програмного забезпечення, встановлюється задача дипломного проєкту.

Другий розділ присвячений проєктуванню програмного забезпечення. У ньому наведений перелік функціональних та не функціональних вимог до програмного забезпечення, розглянуто сучасні архітектурні рішення побудови складних веб-систем, вибір менеджера пакетів, інструментів для пакування та оптимізації додатку, бібліотек для роботи з формами, рішень для управління станом та маршрутизації, а також UI-бібліотек та фреймворків. Описується проєктування локалізації додатку.

Третій розділ описує реалізацію системи. Описується створення та налаштування проєкту, включаючи встановлення основних залежностей та налаштування середовища розробки, інтеграцію та налаштування GraphQL, а також файлову структуру проєкту. Розробляється основний функціонал

клієнтської частини додатку, включаючи реєстрацію та аутентифікацію користувача, проходження загального тестування, відображення результатів тестування та редагування профілю користувача. Описується впровадження локалізації, налаштування конфігурації проєкту та контейнеризація додатку.

У четвертому розділі розглянуто дизайн інтерфейсу користувача. В ньому викладено основи UI/UX дизайну, дослідження користувачів та вимоги до інтерфейсу, дизайн сторінок додатку, включаючи сторінки реєстрації та аутентифікації, головну сторінку, сторінку загального тесту, сторінку результату тесту, сторінку результатів пройдених тестів та профіль користувача. Також розглядається адаптивний дизайн та доступність.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Постановка та аналіз проблеми

Для визначення напрямку даної роботи, необхідно спочатку визначити об’єкт дослідження і предмет дослідження, а також надати опис, за допомогою яких методів планується проводити дослідження і вирішення проблеми.

Отже, для дослідження і побудови імітаційної моделі підсистеми виділяємо наступні сутності:

- Об’єктом дослідження даної роботи є клієнтське програмне забезпечення автоматизованої системи оцінки психічного стану.
- Предмет дослідження роботи є побудова архітектури, розробка інтерфейсу користувача та впровадження в експлуатацію клієнтського програмного забезпечення. В основі програмного забезпечення лежить додаток, який відповідає за комунікацію з сервером.
- Методи дослідження – на початку необхідно провести дослідження існуючих систем для оцінки психічного стану людини, таких як психологічні тестування; вивчити наукові статті та книги, що стосуються теорій психічного здоров'я, методів оцінки та інформаційних технологій в цій галузі; визначити параметри, які можуть бути використані для оцінки психічного стану; обрати стек технологій для розробки клієнтського програмного забезпечення та побудувати архітектуру всього додатку.

Розробка автоматизованої системи оцінки психічного стану людини є новою та перспективною сферою людської діяльності, оскільки має потенціал для суттєвого покращення охорони здоров’я людини. Автоматизовані системи оцінки психічного стану мають ряд переваг порівняно з традиційними методами. Вони можуть бути більш об’єктивними, точними та швидкими. Розробка інформаційного забезпечення автоматизованої системи оцінки психічного стану є важливим завданням, яке має потенціал для трансформації охорони здоров’я людини.

Поетапний процес розробки інформаційного забезпечення автоматизованої системи оцінки психічного стану можливо представити у вигляді життєвого циклу, серед яких виділяють наступні основні етапи:

1. Етап планування;
2. Етап збору вимог;
3. Етап проєктування;
4. Етап розробки програмного забезпечення;
5. Етап тестування програмного забезпечення;
6. Використання готового продукту на ринок;
7. Етап експлуатації та технічного обслуговування.

На першому етапі узгоджуються всі деталі майбутньої автоматизованої системи оцінки психічного стану людини, відповідаючи на важливі питання. Яка мета розробки? Які проблеми воно має вирішувати? Навіщо створюється система? Після відповіді на ці питання формується єдине бачення майбутньої системи. Що полегшує розробку та мінімізує ризики при створенні ПЗ.

Етап аналізу та збору вимог є одним із найважливіших етапів життєвого циклу розробки автоматизованої системи оцінки психічного стану людини. На цьому етапі збираються всі необхідні дані для успішної розробки та впровадження системи.

Після того, як стали зрозумілими цілі та завдання системи, а також технології, які будуть використовуватися для її розробки, можна переходити до проектування та дизайну. На цьому етапі проєктується майбутня архітектура проєкту у вибраній технології. Створюється адаптивний та зручний дизайн, продумується зв'язок фронт-енд частини програми з сервером, розробляються модулі та продумується система безпеки ресурсу.

Стадія розробки – це етап, на якому відбувається створення програмного продукту. На цьому етапі береться проєктна документація, прототипи, дизайн та архітектура, і на основі їх створюють код, який реалізує всі функціональні можливості системи. Завдання на цьому етапі розділені між членами команди відповідно до їхньої галузі спеціалізації. Розробники front-end частини

відповідають за створення інтерфейсу користувача, який буде відображатися на веб-браузері. Розробники back-end частини відповідають за створення серверної частини системи, яка обробляє дані та відповідає за функціональність системи. Адміністратори бази даних відповідають за створення та наповнення бази даних системи. Результатом етапу розробки є готовий програмний продукт, який відповідає вимогам, визначеним на етапі аналізу та збору вимог.

На етапі тестування команда перевіряє, чи відповідає розроблена система вимогам, визначеним на етапі аналізу та збору вимог. Тестування включає в себе такі основні завдання:

- Перевірка функціональності системи;
- Перевірка ефективності системи;
- Перевірка безпеки системи.

Після того, як система протестована і готова до використання, її запускають в експлуатацію. Система стає доступною для користувачів, які можуть її використовувати для оцінки свого психічного стану. Замовник збирає відгуки від користувачів, щоб зрозуміти, чи відповідає система їхнім потребам. Якщо в результаті збору відгуків виявляються помилки в системі, їх виправляють розробники.

Технічне обслуговування – це етап, на якому система підтримується в робочому стані і забезпечується її ефективне використання. На цьому етапі система постійно перевіряється на наявність помилок і проблем. Також на цьому етапі можуть вноситися зміни в систему для її поліпшення. Ці зміни можуть бути спрямовані на:

- усунення виявлених проблем;
- розширення функціональності системи;
- вдосконалення інтерфейсу користувача;
- підвищення продуктивності системи;
- забезпечення безпеки системи.

1.2 Дерево цілей

Основною метою роботи є отримання функціональної клієнтської частини автоматизованої системи оцінки психологічного стану. Дерево цілей опису даної мети наведено на рис. 1.1.

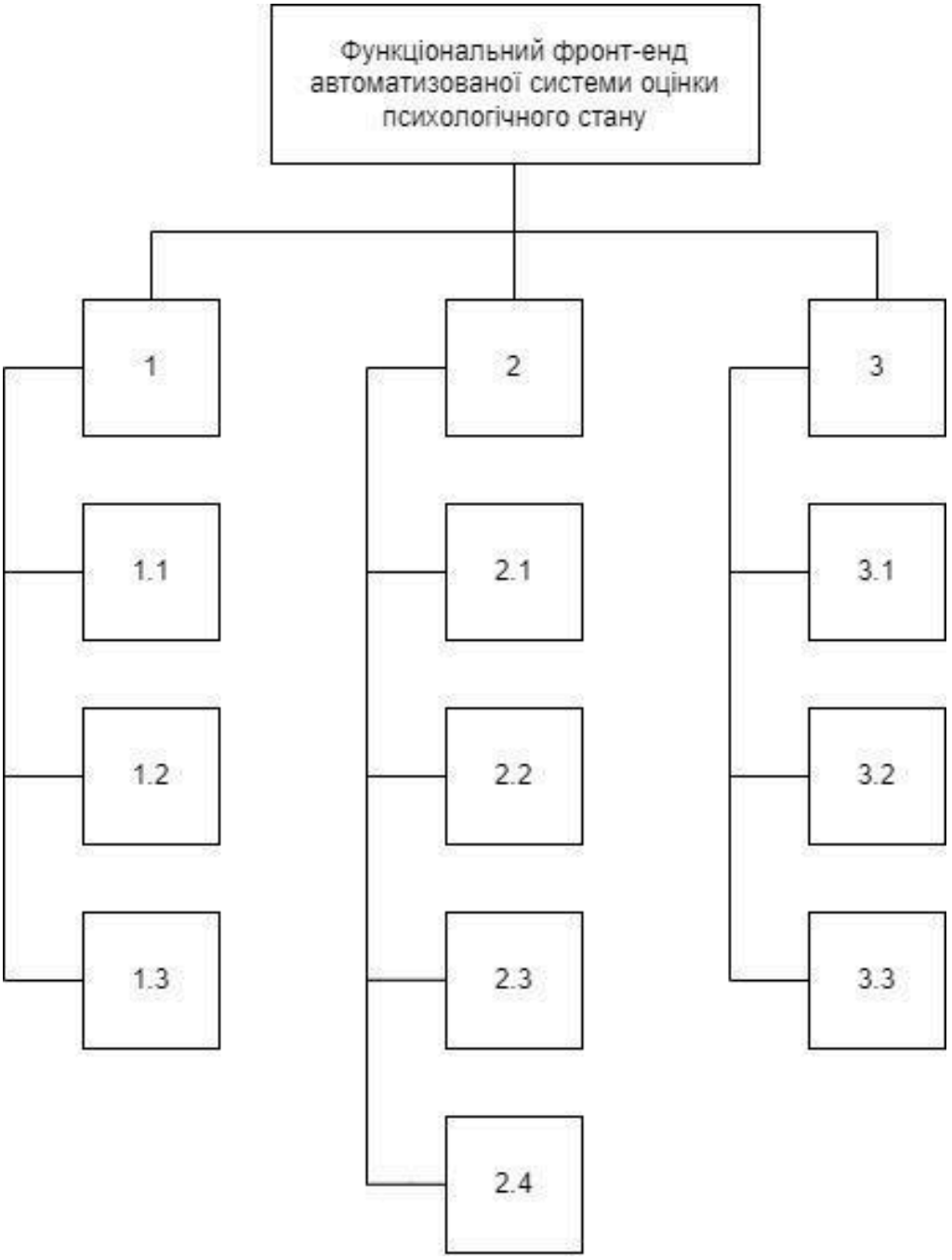


Рисунок 1.1. Дерево цілей

1. Проєктування програмного забезпечення
    1.1. Аналіз функціональних та не функціональних вимог.
    1.2. Огляд та вибір залежностей.
    1.3. Проєктування локалізації додатку.
2. Розробка програмного забезпечення
    2.1. Створення проєкту.
    2.2. Встановлення та налаштування залежностей.
    2.3. Побудова архітектури додатку.
    2.4. Імплементація функціоналу.
3. Дизайн інтерфейсу користувача.
    3.1. Дослідження користувачів та вимоги до інтерфейсу.
    3.2. Дизайн сторінок додатку.
    3.3. Адаптивний дизайн та доступність.

1.3 Вимоги та особливості проєктування системи

Вимоги та особливості фронт-енд частини системи автоматизованої оцінки психологічного стану людини відповідає за взаємодію з користувачем і відображення результатів. Вона повинна відповідати ряду вимог, які забезпечать її зручність і інтуїтивність використання.

Однією з основних вимог є приватність. Інформація, яку користувач надає, може бути конфіденційною, тому її відображення та передача повинні бути захищені від несанкціонованого доступу. Для цього необхідно використовувати сучасні методи шифрування та аутентифікації.

Іншою важливою вимогою є стабільність. Фронт-енд система повинна працювати безперебійно, навіть при великому навантаженні. Для цього необхідно використовувати надійні клієнтські технології.

Крім того, фронт-енд система повинна бути ефективною. Вона повинна відображати дані швидко і точно. Для цього необхідно використовувати сучасні алгоритми відображення даних і ефективні методи обробки запитів.

Для забезпечення приватності фронт-енд системи необхідно використовувати такі заходи:

- Шифрування даних. Вся інформація, яку користувач надає, повинна бути зашифрована перед її відправкою на сервер. Для цього можна використовувати такі алгоритми шифрування, як AES (Advanced Encryption Standard), RSA (Rivest-Shamir-Adleman) або ECC (Elliptic Curve Cryptography).
- Аутентифікація користувачів. Кожен користувач повинен бути ідентифікований перед тим, як він зможе отримати доступ до системи. Для цього можна використовувати такі методи аутентифікації, як паролі, біометрія або двофакторна аутентифікація.
- Контроль доступу. Доступ до різних ресурсів системи повинен бути обмежений відповідно до ролі користувача. Для цього можна використовувати такі механізми контролю доступу, як ролі, дозволи та аудит.

Для забезпечення стабільності фронт-енд системи необхідно використовувати такі заходи:

- Надійні клієнтські технології. Фронт-енд система повинна працювати на надійних клієнтських технологіях, які мають достатньо ресурсів для відображення даних.
- Мережеві технології. Фронт-енд система повинна використовувати надійні мережеві технології, які забезпечують безперебійну передачу даних.
- Автоматичне відновлення. Система повинна мати механізм автоматичного відновлення в разі непередбачених збоїв.

Для забезпечення ефективності фронт-енд системи необхідно використовувати наступні заходи:

- Сучасні алгоритми відображення даних. Фронт-енд система повинна використовувати сучасні алгоритми відображення даних, які дозволяють швидко і точно відображати великі обсяги даних.

- Ефективні методи обробки запитів. Фронт-енд система повинна використовувати ефективні методи обробки запитів, які дозволяють швидко виконувати запити користувачів.

Крім того, фронт-енд система повинна бути розроблена з урахуванням вимог користувачів. Вона повинна бути легкою у використанні і надавати користувачам доступ до всіх необхідних функцій.

1.4 Аналіз існуючих розробок систем оцінки психологічного стану

У сучасному світі автоматизовані системи оцінки психічного стану людини через тестування стають все більш актуальними та важливими для забезпечення ментального здоров'я населення. Такі системи дозволяють здійснювати швидку та ефективну оцінку психічного стану, забезпечуючи можливість раннього виявлення та лікування різних психічних розладів. У даному аналізі розглянуто найбільш відомі системи автоматизованої оцінки психічного стану, які вже існують на ринку.

Система «Я-ПСИХОЛОГ» – інформаційна платформа, що дозволяє проводити психологічні тестування та діагностику в режимі онлайн. Основна перевага системи полягає не лише у визначенні психологічного стану, але й у створенні індивідуальних психологічних характеристик для кожного користувача. Платформа використовує діагностичні методики та психологічні тести, які отримали рекомендації від Міністерства освіти і науки, молоді та спорту, Міністерства охорони здоров'я, Міністерства оборони України та інших важливих установ для використання в Україні.

MARTA – проєкт, анонсований міністерством ще у 2021 році під час зустрічі з представниками компанії Apple, представляє інноваційну систему для психологічної підтримки та оцінки психічного здоров’я учасників бойових дій. Проект був презентований Apple у формі демонстраційної версії, яка мала дозволити бійцям отримувати психологічну підтримку та оцінювати своє психічне здоров'я через смартфон або комп'ютер. Згодом MARTA стала

доступною широкому загалу користувачів. Система включає онлайн-тести, призначені для широкого кола користувачів, та допомагає визначити попередню характеристику психічного стану.

Beck Depression Inventory (BDI) Online – онлайн-версія відомого тесту Аарона Бека для визначення рівня депресії. Тест включає в себе питання, спрямовані на виявлення різних аспектів депресивних симптомів. Його перевагами є науково обґрунтована методика та широке застосування в клінічній практиці. Однак необхідно враховувати, що він вимагає від користувача високого рівня самосвідомості.

Anima – онлайн-платформа, яка дозволяє оцінити психологічне здоров'я за допомогою веб-камери. Весь процес передбачає проведення трьоххвилинного тесту, під час якого показуються різноманітні зображення, такі як їжа, пейзажі, портрети тощо. Програма аналізує рухи очей та їх реакцію на зображення, роблячи конкретні висновки. Наприклад, вона допомагає визначити рівень тривожності чи депресії у людини. Аналіз здійснюється за допомогою технології відстеження очей, використовуючи вбудовані математичні моделі для фіксації рухів очей.

Самопоміч – сайт, розроблений Національним інститутом психічного здоров’я Чеської Республіки у співпраці з іншими національними та міжнародними організаціями, які беруть участь у співпраці. Вміст сайту ґрунтується на наукових даних, що означає, що представлена на веб-сайті інформація є достовірною та науково перевіреною. На сайті є набір корисних тестів, які широко використовуються для виявлення проблем зі психічним здоров'ям. Присутній базовий тест, який визначає стан поточного психічного стану. Інші тести зосереджені на конкретних аспектах психічного здоров'я, таких як депресія та тривога. Друга частина містить корисний огляд симптомів, які можуть супроводжувати різні проблеми з психічним здоров'ям.

Порівняння існуючих розробок систем оцінки психічного стану наведено у табл. 1.1.

Таблиця 1.1. Опис існуючих розробок систем оцінки психічного стану

Назва	Метод оцінювання	Країна	Аудиторія	Особливості
«Я-ПСИХОЛОГ»	Тести з питаннями	Україна	Цивільні	Створення індивідуальних психологічних характеристик для кожного користувача
MARTA	Тести з питаннями	Україна	Військові	Надання психологічної підтримки та оцінки психічного здоров’я учасників бойових дій
Beck Depression Inventory (BDI) Online	Тести з питаннями	Чехія	Цивільні	Науково обґрунтована методика та широке застосування в клінічній практиці

Продовження таблиці 1.1. Опис існуючих розробок систем оцінки психічного стану

Назва	Метод оцінювання	Країна	Аудиторія	Особливості
Anima	Тест за допомогою веб-камери та тести з питаннями	Україна	Цивільні	Програма аналізує рухи очей та їх реакцію на зображення, роблячи конкретні висновки
Самопоміч	Тести з питаннями	Чехія	Цивільні	Представлена на веб-сайті інформація є достовірною та науково перевіреною. Містить корисний огляд симптомів, які можуть супроводжувати різні проблеми з психічним здоров'ям

Порівнюючи різноманітні системи тестування психологічного стану, можна визначити декілька ключових аспектів, що варто враховувати при розробці власної автоматизованої системи з оцінки психічного стану. Усі розглянуті системи використовують тести з питаннями як основний метод оцінювання. Проте Anima вирізняється використанням веб-камери для аналізу

рухів очей, що може надати додаткову інформацію про реакцію користувача. MARTA спеціалізується на військових, що робить її специфічною для цього сегменту користувачів. У той час як "Я-ПСИХОЛОГ", Beck Depression Inventory, Anima та Самопоміч орієнтовані на цивільну аудиторію. Розробка систем велась в різних країнах (Україна, Чехія), що може вплинути на особливості їх функціоналу та адаптацію до конкретного культурного та соціального середовища. Beck Depression Inventory та Самопоміч вирізняються науковою обґрунтованістю своїх методик, що підсилює достовірність інформації, яку вони надають. Усі ці фактори свідчать про те, що вибір можливостей та особливостей для системи тестування психологічного стану повинен бути здійснений з урахуванням конкретних потреб користувачів, їх контексту та специфіки ситуації, для якої вони шукають рішення.

1.5 Постановка задачі.

Метою дипломного проєкту є розробка клієнтської частини інформаційної системи для автоматизованої оцінки психічного стану користувача, яка дозволить користувачам проходити тестування, отримувати результати та рекомендації щодо свого стану. Реалізація проєкту передбачає вибір стеку технологій для розробки, побудову архітектури фронт-енд частини інформаційної системи, її розробку та розгортання для подальшого використання. Система повинна бути зручною в користуванні, надійною та захищеною. Очікуваним результатом є система, яка відмінно працює, відповідає всім вимогам користувачів та готова для подальшої експлуатації.

Основні завдання дипломного проекту:

1. Аналіз вимог користувачів та фахівців у галузі психічного здоров'я для визначення потреб і функціональності системи.
2. Вибір стеку технології, вивчення особливостей та можливостей обраного стеку технологій.

3. Проєктування архітектури серверної частини системи, організація структури проєкту.
4. Налаштування циклу розробки програмного забезпечення.
5. Розробка фронт-енд функціоналу системи.
6. Тестування системи.
7. Впровадження системи в експлуатацію.

1.6 Аналіз та вибір технології для реалізації проєкту

Для реалізації клієнтської частини проєкту автоматизованої системи оцінки психологічного стану людини необхідно вибрати технологію, яка відповідатиме таким вимогам:

- Швидкість роботи. Клієнтська частина повинна швидко відповідати на дії користувача та надавати йому необхідну інформацію;
- Безпека. Клієнтська частина повинна бути захищена від несанкціонованого доступу та взлому;
- Легкість у використанні. Клієнтська частина повинна бути легкою у використанні для користувачів;
- Можливість масштабування. Клієнтська частина повинна мати можливість масштабування для підтримки великої кількості користувачів.

Фронт-енд розробка має свої популярні технології. Це зокрема JavaScript, а також його фреймворки та бібліотеки, такі як React, Vue та Angular. Вибір між ними залежить від конкретних потреб проекту, особистих вподобань команди розробників, а також від екосистеми та підтримки, яку пропонує кожний з них.
React – це JavaScript бібліотека для побудови інтерфейсів, розроблена Facebook. React надає велику гнучкість та ефективність завдяки своєму віртуальному DOM (Document Object Model). Він також дозволяє використовувати JSX, синтаксис, який дозволяє вбудовувати HTML (Hypertext Markup Language) в JavaScript.

Сильні сторони React:

- Ефективність. React використовує віртуальний DOM, що дозволяє зменшити кількість прямих викликів DOM, що в свою чергу підвищує продуктивність;
- Компонентний підхід. React дозволяє створювати повторно використовувані компоненти, що спрощує розробку та підтримку коду.

Слабкі сторони React:

- Високий поріг входу. React може бути важким для вивчення, особливо для новачків;
- Бібліотека, а не фреймворк. React є бібліотекою, а не повноцінним фреймворком, тому для реалізації деяких функцій може знадобитися використання додаткових бібліотек.

Vue – це прогресивний JavaScript фреймворк для створення інтерфейсів користувача. Vue є більш легким та гнучким, ніж React або Angular, і він легко інтегрується з існуючими проектами. Vue також надає можливість використовувати HTML, CSS (Cascading Style Sheets) та JavaScript безпосередньо в .vue файлах.

Сильні сторони Vue:

- Простота вивчення. Vue має простий та зрозумілий синтаксис, що полегшує вивчення фреймворка;
- Гнучкість. Vue дозволяє використовувати різні підходи до розробки, включаючи як класичний серверний рендеринг, так і односторінкові застосунки.

Слабкі сторони Vue:

- Менша спільнота. Vue має меншу спільноту порівняно з React або Angular, що може впливати на кількість доступних ресурсів для вивчення та підтримки.

Angular – це повноцінний JavaScript фреймворк, розроблений Google. Angular є більш важким, ніж React або Vue, але він надає багато функцій "з

коробки", включаючи двосторонню прив'язку даних, модульність, HTTP (Hypertext Transfer Protocol) сервіси та залежності.

Сильні сторони Angular:

- Повноцінний фреймворк. Angular надає все необхідне для розробки веб-застосунків, включаючи підтримку форм, маршрутизації, HTTP запитів і т.д.;
- Типізація. Angular використовує TypeScript, що забезпечує статичну типізацію та підвищує якість коду.

Слабкі сторони Angular:

- Складність. Angular може бути складним для вивчення, особливо для новачків;
- Продуктивність. Angular може бути менш продуктивним порівняно з React або Vue через свою вагу та складність;

Нативний JS (JavaScript) – це використання "чистого" JavaScript без будь-яких додаткових бібліотек або фреймворків. Це може бути корисним для дуже простих веб-сайтів або для навчання основ JavaScript, але для більш складних застосунків це може бути менш ефективним та займати більше часу на розробку, порівняно з використанням фреймворків.

Сильні сторони нативного JS:

- Немає залежностей. Використання нативного JS виключає необхідність використання додаткових бібліотек або фреймворків;
- Швидкість. Нативний JS може бути швидшим, ніж JS з використанням додаткових бібліотек або фреймворків.

Слабкі сторони нативного JS:

- Складність. Розробка великих застосунків з використанням нативного JS може бути складнішою і потребувати більше часу;
- Підтримка: Нативний JS може мати проблеми з підтримкою в різних браузерах.

З урахуванням цих вимог, для реалізації клієнтської частини проєкту найбільш доцільно використовувати бібліотеку React. Вона відповідає всім

вимогам проєкту, є популярною бібліотекою з великою спільнотою розробників, а також забезпечує високу продуктивність та швидкість роботи.

Переваги використання React для реалізації клієнтської частини проєкту:

- Швидкість роботи. React використовує віртуальний DOM, що дозволяє зменшити кількість прямих викликів DOM, що в свою чергу підвищує продуктивність;
- Безпека. React має вбудовані механізми захисту від таких атак, як cross-site scripting;
- Легкість у використанні. React має простий синтаксис, що спрощує розробку та підтримку коду;
- Можливість масштабування. React має вбудовану підтримку для розробки великих застосунків з великою кількістю компонентів.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Опис функціональних та нефункціональних вимог програмного забезпечення

Функціональні та нефункціональні вимоги до програмного забезпечення є ключовим етапом у процесі розробки будь-якої інформаційної системи. Функціональні вимоги визначають, які функції має виконувати програмне забезпечення, які операції воно повинно забезпечувати та які результати мають бути отримані після виконання цих операцій. Нефункціональні вимоги визначають якість програмного забезпечення, такі як продуктивність, надійність, безпека, сумісність та інші аспекти, які не стосуються конкретних функцій програми.

Система повинна реалізовувати наступні функціональні вимоги:

- Реєстрація користувачів. Система повинна мати можливість реєстрації нових користувачів з подальшою можливістю збереження результатів тестування.
- Авторизація користувачів. Доступ до системи повинен бути обмежений та контрольований за допомогою авторизації. Кожен користувач повинен мати можливість увійти до свого облікового запису.
- Проведення тестів. Система повинна мати можливість проведення тестів для оцінки психологічного стану користувача.
- Аналіз результатів. Система автоматично оброблятиме результати тестування та надаватиме список виявлених потенціальних психологічних проблем користувачу.
- Надання рекомендацій. Користувач повинен мати можливість не лише дізнатись свої потенційні розлади, але й отримати повний список рекомендацій, завдяки яким можна покращити свій психологічний стан.

- Особистий профіль користувача. Користувач повинен мати можливість переглядати та редагувати особисту інформацію та за бажанням переглядати результати своїх попередніх тестувань.

До нефункціональних вимог належать наступні вимоги:

- Ефективність. Система повинна забезпечувати швидку та точну оцінку психологічного стану людини без зайвого затримання. Це означає, що час, необхідний для проведення тестування та отримання результатів, повинен бути мінімальним, щоб користувачі могли швидко отримати інформацію про свій психологічний стан.
- Надійність. Система повинна працювати безперебійно та надійно, забезпечуючи коректну інтерпретацію результатів тестування. Надійність системи важлива для того, щоб користувачі могли довіряти отриманим результатам та використовувати їх для подальшого аналізу свого психологічного стану.
- Безпека. Система повинна гарантувати конфіденційність даних користувачів, забезпечуючи захист від несанкціонованого доступу. Це означає, що всі особисті дані користувачів повинні бути зашифровані та зберігатися в безпечному середовищі, щоб уникнути витоку конфіденційної інформації.
- Масштабованість. Система повинна бути готовою до масштабування, здатною обробляти великий потік даних та користувачів без втрати продуктивності. Це дозволить системі ефективно працювати навіть при збільшенні обсягу користувачів та даних.
- Сумісність. Система повинна бути сумісною з різними операційними системами та пристроями, що дозволить користувачам отримати доступ до неї з будь-якого пристрою. Це забезпечить доступність системи для широкого кола користувачів та зручність її використання.
- Документація. Наявність докладної та зрозумілої документації з описом функціоналу системи, процесу взаємодії та вимог до користувачів. Це допоможе користувачам швидко ознайомитися з

системою, її можливостями та правилами використання, що сприятиме успішній інтеграції системи у їхню роботу.

2.2 Вибір менеджера пакетів

Менеджер пакетів – це інструмент, який автоматизує процес завантаження, оновлення, налаштування та видалення програмного забезпечення. Він дозволяє розробникам легко управляти залежностями проекту, що є особливо важливим у сучасному програмуванні, де проекти часто залежать від багатьох зовнішніх бібліотек та модулів.

Ми розглянемо три популярні менеджери пакетів для JavaScript: NPM, yarn та PNPM. Нижче наведено порівняння їхніх характеристик, переваг та недоліків.

NPM (Node Package Manager) – NPM є стандартним менеджером пакетів для Node.js. Він поставляється разом з Node.js і має найбільшу базу користувачів та пакетів [1].

Основні переваги NPM включають в себе:

- Широке використання. NPM є стандартним менеджером пакетів для Node.js, тому він має найбільшу базу користувачів та пакетів;
- Вбудована підтримка. NPM поставляється разом з Node.js, тому для початку роботи не потрібна додаткова установка;
- Гарна документація. Велика кількість документації та ресурсів для навчання.

Основні недоліки NPM включають в себе:

- Швидкість. NPM має повільніший час встановлення порівняно з Yarn і PNPM;
- Проблеми з управлінням залежностями. Раніше NPM мав проблеми з дублюванням пакетів та циклічними залежностями;
- Продуктивність. Високе споживання пам'яті при роботі з великими проектами.

Yarn – це швидкий, надійний та безпечний менеджер пакетів, розроблений Facebook. Він відомий своєю швидкістю та підтримкою монорепозиторіїв.

Основні переваги Yarn включають в себе:

- Швидкість. Yarn зазвичай швидший за NPM через агресивне кешування та паралельну установку пакетів;
- Робота з робочими просторами. Yarn добре підтримує монорепозиторії;
- Детерміновані установки. Yarn lock-файли забезпечують детермінованість встановлення пакетів.

Основні недоліки Yarn включають в себе:

- Додаткова установка. Потребує встановлення окремо від Node.js;
- Складність. Може бути складнішим у використанні для новачків через розширені можливості.

PNPM (Performant Node Package Manager) – це швидкий і ефективний менеджер пакетів, який використовує посилання на спільний кеш замість дублювання пакетів, що значно зменшує використання дискового простору.

Основні переваги PNPM включають в себе:

- Ефективність використання диску. PNPM використовує посилання на загальну кешовану копію замість дублювання пакетів, що значно зменшує споживання дискового простору;
- Швидкість. PNPM швидший за NPM і часто конкурує з Yarn у швидкості завдяки ефективному кешуванню та паралельній установці;
- Ізоляція залежностей. PNPM створює ізольовані дерева залежностей, що знижує ймовірність конфліктів версій;
- Підтримка робочих просторів: Як і Yarn, PNPM добре працює з монорепозиторіями.

Основні недоліки PNPM включають в себе:

- Менша популярність. PNPM не такий популярний як NPM чи Yarn, тому можна зустріти менше ресурсів для навчання;
- Додаткова установка. Потрібно встановлювати окремо від Node.js.

З графіку порівнянь часу завантаження залежностей (рис 2.1) видно що PNPM значно швидший за своїх конкурентів.

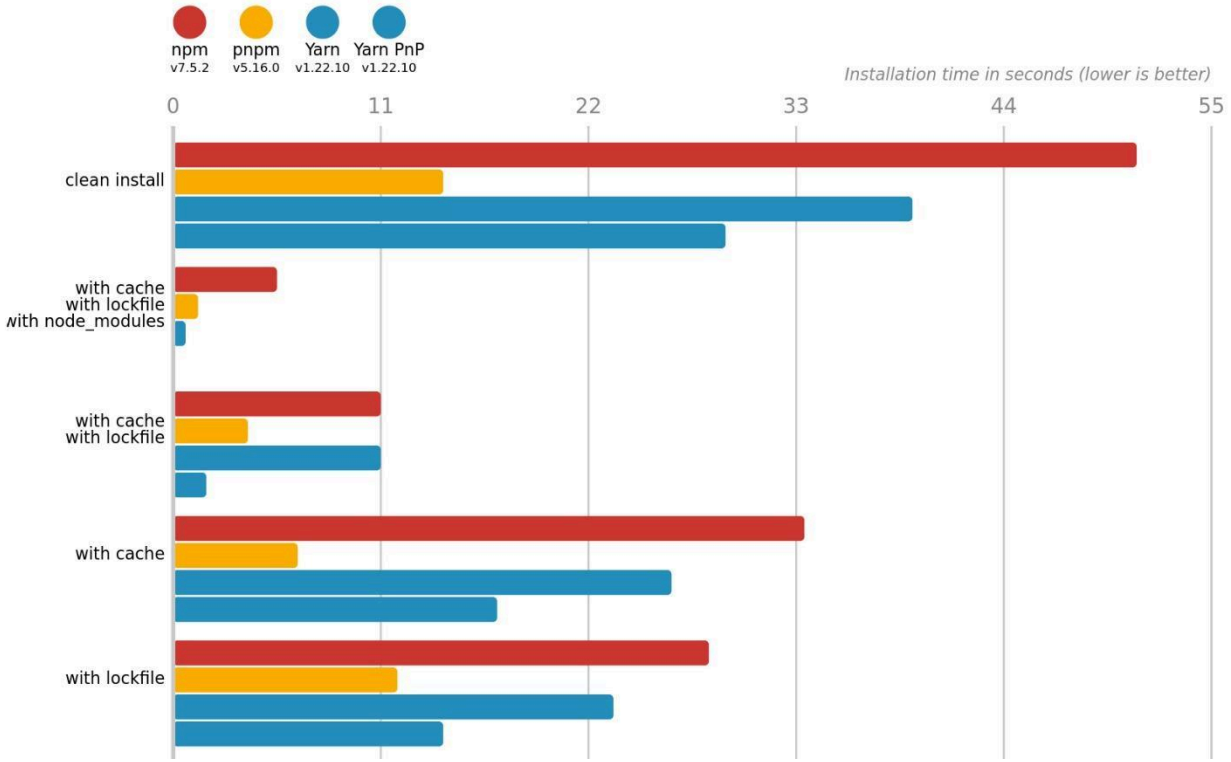


Рисунок 2.1. Графік порівнянь часу завантаження залежностей

Отже, проаналізувавши всі вищезазначені менеджери пакетів, можна зробити висновок, що найбільш доцільно використовувати PNPM, так як він більше підходить для проектів середнього та великого масштабів. З цим менеджером пакетів швидше можна встановлювати залежності, а також з ним простіше працювати та масштабувати проєкт у перспективі.

2.3 Інструменти для пакування та оптимізації додатку

Під час розробки веб-додатків на React важливо використовувати інструменти для пакування та оптимізації коду, щоб забезпечити ефективність, швидкість та легкість у підтримці проєкту. Серед популярних інструментів для цієї мети можна виділити CRA, Vite, Webpack, Parcel та esbuild.

CRA (Create React App) – це офіційний інструмент, розроблений командою React для швидкого створення нових додатків на React з нульовою конфігурацією. Він налаштовує основний робочий процес, включаючи

пакування, налаштування Babel, ESLint, і інші інструменти для розробки. CRA ідеально підходить для новачків та для швидкого створення прототипів, оскільки він приховує складнощі конфігурації та забезпечує готове середовище для розробки [2].

Vite – це сучасний інструмент для розробки, який орієнтований на швидкість та ефективність. Він використовує esbuild для попередньої компіляції залежностей та ES-модулі для забезпечення миттєвого завантаження під час розробки. Виробничі збірки обробляються за допомогою Rollup, що забезпечує ефективну оптимізацію, включаючи tree-shaking та code-splitting. Vite пропонує мінімальну конфігурацію за замовчуванням, що може бути розширена за допомогою плагінів для специфічних потреб.

Webpack – це потужний і гнучкий бандлер, який дозволяє зібрати всі ресурси додатку в єдину збірку. Він підтримує широкий спектр плагінів і лоадерів, які можуть бути налаштовані для вирішення будь-яких завдань. Webpack відомий своєю крутою кривою навчання, але його можливості роблять його незамінним для великих та складних проектів, де потрібна висока гнучкість.

Parcel – це простий у використанні бандлер, який практично не потребує конфігурації. Він використовує багатопоточність для прискорення процесу зборки і підтримує інкрементальні збірки для швидкого перезавантаження під час розробки. Parcel автоматично налаштовує оптимізацію бандлів, але не настільки гнучкий як Webpack, що може бути обмеженням для складних проектів.

esbuild – це надзвичайно швидкий бандлер, написаний на Go. Він забезпечує миттєву компіляцію та збірку JavaScript-коду, значно перевершуючи інші інструменти за швидкістю. esbuild має просту конфігурацію та API (Application Programming Interface), але наразі не підтримує всі можливості та плагіни, доступні в Webpack або Parcel, що може обмежити його використання в деяких випадках.

Виходячи з наведеного порівняння, Vite виглядає найбільш привабливим вибором для проєкту на React через наступні причини:

- Швидкість. Vite забезпечує значно швидший старт та розробку, оскільки використовує esbuild для попередньої компіляції залежностей;
- Простота використання. Vite має простіше налаштування у порівнянні з Webpack, що дозволяє швидше налаштувати проєкт і менше часу витрачати на конфігурацію;
- Модульність та сучасність. Використання ES-модулів робить процес розробки більш інтерактивним та сучасним;
- Оптимізація. Використання Rollup для виробничих збірок забезпечує високий рівень оптимізації, що важливо для кінцевого продукту.

Таким чином, Vite є оптимальним інструментом для пакування та оптимізації додатку на React, забезпечуючи баланс між швидкістю, простотою використання та можливістю отримати оптимізовану виробничу збірку.

2.4 Вибір бібліотек для роботи з формами

Робота з формами є однією з найважливіших складових при розробці веб-додатків, особливо коли мова йде про інтерактивні та динамічні інтерфейси. Для розробників, що використовують React, існує безліч бібліотек, які полегшують роботу з формами, надаючи зручні інструменти для керування станом, валідацією та обробкою даних. Вибір правильної бібліотеки може суттєво вплинути на ефективність розробки та якість кінцевого продукту. Розглянемо декілька популярних бібліотек для роботи з формами в контексті React.

Formik є однією з найпопулярніших бібліотек для роботи з формами у React. Вона надає зручний API для керування станом форм, їх валідацією та обробкою відправлення даних.

Основні переваги Formik включають:

- Простота використання та зрозуміла документація;
- Підтримка валідації за допомогою бібліотеки Yup;
- Легка інтеграція з іншими інструментами та бібліотеками.

Основні недоліки Formik включають:

- Може бути дещо важковаговою для невеликих проєктів;
- Більший розмір бандла порівняно з іншими рішеннями.

React Hook Form використовує концепцію хуків для керування формами у React. Ця бібліотека фокусується на продуктивності та мінімізації кількості перерендерів, забезпечуючи при цьому простий і зрозумілий API [3].

Основні переваги React Hook Form включають:

- Менший розмір бандла завдяки використанню хуків;
- Висока продуктивність завдяки відсутності перерендерів при зміні стану форми;
- Вбудована підтримка валідації без необхідності використання додаткових бібліотек.

Основні недоліки React Hook Form включають:

- Вимагає деякого часу для ознайомлення з концепцією хуків;
- Може бути складним для новачків у React.

Redux Form інтегрується з Redux, надаючи інструменти для керування станом форм через глобальний стан додатка. Це рішення добре підходить для великих додатків з складною логікою стану.

Основні переваги Redux Form включають:

- Тісна інтеграція з Redux для керування станом додатка;
- Відмінно підходить для великих проєктів з складним станом.

Основні недоліки Redux Form включають:

- Високий рівень складності налаштування та використання;
- Значно збільшує розмір бандла;
- Знижена продуктивність через часті перерендери компонентів.

Після порівняння різних бібліотек для роботи з формами у React, можна зробити висновок, що React Hook Form є оптимальним вибором для більшості

проєктів. Він поєднує в собі високу продуктивність, невеликий розмір бандла та зручність у використанні. Завдяки концепції хуків, React Hook Form дозволяє ефективно керувати станом форми без зайвих перерендерів, що особливо важливо для продуктивності великих додатків.

Обираючи React Hook Form, ми отримуємо сучасне та продуктивне рішення, яке легко інтегрується з іншими інструментами та бібліотеками, а також забезпечує високу гнучкість у реалізації будь-яких вимог до форм у веб-додатках на базі React.

2.5 Огляд рішень для управління станом

Управління станом є важливою частиною розробки сучасних веб-застосунків. У додатках на React, ефективне управління станом дозволяє розробникам підтримувати організовану структуру коду, покращувати продуктивність та спрощувати процес розробки. Існує кілька популярних бібліотек для управління станом у React, кожна з яких має свої особливості, переваги та недоліки.

У цьому пункті ми розглянемо найпопулярніші рішення, зокрема Redux, MobX, Context API та Zustand.

Redux є однією з найпопулярніших бібліотек для управління станом у React. Вона забезпечує передбачуваність та централізацію стану через єдине дерево стану та дії [4].

Основні переваги Redux включають:

- Передбачуваність. Строгі правила оновлення стану полегшують дебагінг та тестування;
- Інструменти розробки. Redux DevTools значно спрощують процес розробки та відлагодження.

Основні недоліки Redux включають:

- Шаблонність. Часто вимагає багато шаблонного коду (boilerplate);
- Крута крива навчання. Для новачків може бути складним у засвоєнні.

MobX пропонує інший підхід до управління станом, заснований на реактивності. Стан описується як "спостережуваний" (observable), і зміни автоматично відображаються у компонентах.

Основні переваги MobX включають:

- Простота використання. Менше шаблонного коду, ніж у Redux;
- Реактивність. Автоматичне оновлення UI при зміні стану.

Основні недоліки MobX включають:

- Неявна логіка. Може бути важко відслідковувати, як і коли змінюється стан;
- Менша передбачуваність. Через автоматичне оновлення складніше зрозуміти потік даних.

Вбудована у React функція Context API дозволяє передавати стан через дерево компонентів без необхідності пропс-дріллінгу.

Основні переваги Context API включають:

- Простота інтеграції. Вже є частиною React, не потребує додаткових бібліотек;
- Гнучкість. Легко використовувати для невеликих проектів або окремих частин додатка.

Основні недоліки Context API включають:

- Продуктивність. Може призводити до проблем з продуктивністю при великому обсязі даних;
- Відсутність інструментів розробки. Не має таких потужних інструментів, як Redux.

Zustand є більш легковаговою альтернативою для управління станом, яка забезпечує простий API та високу продуктивність.

Основні переваги Zustand включають:

- Легкість та простота. Мінімальна кількість шаблонного коду;
- Висока продуктивність. Оновлення стану без необхідності повторного рендерингу всього компонента;
- Прямий доступ до стану. Використання хуків для доступу до стану.

Основні недоліки Zustand включають:

- Менша екосистема. Менше додаткових інструментів та розширень у порівнянні з Redux.

Після детального розгляду доступних рішень для управління станом у React, обираемо Zustand для нашого проекту. Основними причинами цього вибору є простота використання, висока продуктивність та легкість інтеграції. Zustand забезпечує необхідну гнучкість та ефективність, що робить його ідеальним вибором для сучасних React-застосунків, де важлива як продуктивність, так і зручність розробки.

Таким чином, використання Zustand дозволить створити ефективний та легко підтримуваний код, зосередившись на основних завданнях проекту без зайвих ускладнень.

2.6 Огляд рішень для маршрутизації

Маршрутизація є ключовим аспектом у розробці сучасних веб-додатків, особливо тих, які побудовані на основі React. Вона дозволяє розробникам створювати багатосторінкові додатки з плавною навігацією без перезавантаження сторінки. Це не лише покращує користувацький досвід, але й підвищує продуктивність додатка. У цьому розділі ми розглянемо основні рішення для маршрутизації у React-додатках, порівняємо їхні можливості, переваги та недоліки, а також обґрунтуємо вибір конкретного рішення для нашого проекту.

React Router є найпоширенішим рішенням для маршрутизації у додатках на основі React. Він надає повний набір інструментів для реалізації складних маршрутизаційних схем, включаючи динамічні маршрути, вкладені маршрути та маршрути з параметрами [5].

Основні переваги React Router включають:

- Гнучкість. Підтримка динамічних і вкладених маршрутів;

- Спільнота. Велика кількість прикладів та документації завдяки широкому використанню;
- Підтримка. Регулярні оновлення та підтримка від спільноти та розробників.

Основні недоліки React Router включають:

- Крива навчання. Для новачків може бути складним через велику кількість функцій та можливостей;
- Вага. Може збільшити розмір бандлу додатка.

Reach Router був розроблений для забезпечення простішого і більш зручного API для маршрутизації у React-додатках. Основний акцент зроблено на доступності та простоті використання.

Основні переваги Reach Router включають:

- Простота. Простий та інтуїтивно зрозумілий API;
- Доступність. Вбудована підтримка доступності (a11y).

Основні недоліки Reach Router включають:

- Менша гнучкість. Менше можливостей у порівнянні з React Router;
- Обмежена підтримка. Менша спільнота та рідше оновлення у порівнянні з React Router.

Next.js – це фреймворк для React, який пропонує вбудовану маршрутизацію на основі файлової системи. Кожен файл у директорії pages автоматично стає маршрутом.

Основні переваги Next.js включають:

- Простота. Автоматичне створення маршрутів на основі файлової системи;
- Інтеграція. Глибока інтеграція з іншими функціями Next.js, такими як серверний рендеринг.

Основні недоліки Next.js включають:

- Файлова структура. Обмеження на організацію файлів у проекті;
- Менше контролю. Менший контроль над конфігурацією маршрутів у порівнянні з React Router.

Вибір відповідного рішення для маршрутизації залежить від специфічних вимог проекту. React Router є найбільш гнучким і потужним інструментом для складних додатків з багатою маршрутизацією. Reach Router підходить для простіших проектів, де важлива швидкість розробки та доступність. Next.js Router є ідеальним для проектів, які використовують Next.js і вимагають серверного рендерингу.

Для нашого React-додатка, враховуючи потребу в складних маршрутизаційних схемах та активне використання спільноти, було обрано React Router. Він надає необхідну гнучкість та підтримку, що дозволить легко розширювати та підтримувати додаток у майбутньому.

2.7 Огляд UI-бібліотек та фреймворків

У сучасному веб-розробленні створення інтуїтивно зрозумілого та привабливого інтерфейсу користувача є критично важливим завданням. Для цього існує безліч UI-бібліотек та фреймворків, які допомагають розробникам створювати ефективні, адаптивні та естетично привабливі інтерфейси. У цьому пункті ми розглянемо кілька популярних UI-бібліотек та фреймворків, таких як Material-UI, Ant Design, Bootstrap та JoyUI, порівняємо їхні особливості, переваги та недоліки.

Material-UI – це популярна React бібліотека, яка реалізує концепції Material Design від Google. Вона пропонує широкий набір компонентів, які легко налаштовуються та інтегруються [6].

Основні переваги Material-UI включають:

- Відповідає стандартам Material Design;
- Велика кількість готових компонентів;
- Активна спільнота та регулярні оновлення.

Основні недоліки Material-UI включають:

- Може бути складним для налаштування під специфічні потреби;

- Висока залежність від стилів Material Design, що може обмежувати креативність.

Ant Design – це комплексна UI-бібліотека для React, розроблена компанією Alibaba. Вона відома своїм багатим набором компонентів та високою якістю дизайну.

Основні переваги Ant Design включають:

- Великий набір компонентів;
- Висока якість та увага до деталей;
- Хороша документація та підтримка.

Основні недоліки Ant Design включають:

- Досить важка для налаштування;
- Орієнтована на корпоративні додатки, що може не підходити для всіх проектів.

Bootstrap – це один з найпопулярніших CSS-фреймворків, який часто використовують для швидкого створення адаптивних веб-інтерфейсів. Існує також версія для React (React-Bootstrap).

Основні переваги Bootstrap включають:

- Широко використовується та добре задокументований;
- Легка інтеграція та швидке налаштування;
- Велика кількість тем та шаблонів.

Основні недоліки Bootstrap включають:

- Обмежена кількість компонентів порівняно з іншими бібліотеками;
- Складно налаштовувати для специфічних потреб без написання додаткового CSS.

JoyUI – це нова UI-бібліотека для React, яка пропонує сучасний підхід до створення користувацьких інтерфейсів. Вона орієнтована на простоту використання та високу адаптивність.

Основні переваги JoyUI включають:

- Сучасний та мінімалістичний дизайн;
- Легка інтеграція та налаштування;

- Висока адаптивність та гнучкість.

Основні недоліки JoyUI включають:

- Відносно нова, тому може мати меншу спільноту та документацію;
- Обмежена кількість компонентів у порівнянні з більш зрілими бібліотеками.

Після детального аналізу існуючих рішень, вибір UI-бібліотеки для нашого додатку на React зупинився на JoyUI. Основні причини цього вибору включають сучасний та мінімалістичний дизайн, високу адаптивність та легкість інтеграції. Незважаючи на те, що JoyUI є відносно новою бібліотекою, її переваги значно перевищують недоліки, особливо для проектів, які потребують сучасного та гнучкого підходу до створення інтерфейсу користувача. Таким чином, JoyUI є оптимальним вибором для реалізації нашого додатку.

2.8 Проєктування локалізації додатку

Локалізація додатків стає все більш важливою в умовах глобалізації, дозволяючи розробникам охоплювати ширшу аудиторію, надаючи користувачам можливість взаємодіяти з додатком на їх рідній мові. У цьому розділі ми розглянемо процес проєктування локалізації для нашого додатку, порівняємо існуючі рішення та обґрунтуємо вибір оптимального інструменту для цього завдання.

i18next – це потужна і гнучка бібліотека для інтернаціоналізації JavaScript-додатків. Вона підтримує різні платформи, включаючи React, і має багатий набір функцій, таких як динамічне завантаження перекладів, підтримка множинних мов та форматування дат, чисел і валют [7].

Основні переваги i18next включають:

- Легка інтеграція з React.js через react-i18next;
- Підтримка асинхронного завантаження перекладів;
- Можливість використання ключів у вигляді JSON-об'єктів;

- Підтримка множинних мов і регіональних налаштувань;
- Широка документація і активна спільнота.

Основні недоліки i18next включають:

- Потребує певного часу на налаштування та освоєння.

React-Intl – це бібліотека для інтернаціоналізації, створена спеціально для React-додатків. Вона базується на стандарті Unicode CLDR і надає інструменти для форматування дат, чисел і повідомлень.

Основні переваги React-Intl включають:

- Спеціалізована для React, що спрощує інтеграцію;
- Вбудована підтримка форматування дат, чисел і валют;
- Використання стандартів Unicode CLDR.

Основні недоліки React-Intl включають:

- Менш гнучка у порівнянні з i18next;
- Обмежена підтримка асинхронного завантаження перекладів.

LinguiJS – це бібліотека для інтернаціоналізації, яка фокусується на простоті використання та продуктивності. Вона надає інструменти для компіляції перекладів і оптимізації розміру бандлу.

Основні переваги LinguiJS включають:

- Простота використання та налаштування;
- Інструменти для компіляції та оптимізації перекладів;
- Підтримка TypeScript.

Основні недоліки LinguiJS включають:

- Менш гнучка у порівнянні з i18next;
- Обмежена підтримка асинхронного завантаження перекладів.

Після аналізу існуючих рішень для інтернаціоналізації додатків на React, ми обрали i18next як оптимальне рішення для нашого проєкту. Основні причини цього вибору включають:

1. Гнучкість та функціональність. i18next пропонує багатий набір функцій, включаючи підтримку асинхронного завантаження

перекладів і можливість використання JSON-об'єктів для зберігання ключів перекладів.

2. Широка підтримка: Бібліотека підтримує множинні мови та регіональні налаштування, що дозволяє легко масштабувати додаток для різних ринків.
3. Документація та спільнота: i18next має широке ком'юніті та детальну документацію, що спрощує процес інтеграції та налаштування.
4. Інтеграція з React: Завдяки react-i18next інтеграція з React стає простою та інтуїтивно зрозумілою.

Таким чином, вибір i18next для локалізації нашого додатку є обґрунтованим та доцільним, враховуючи всі переваги, які ця бібліотека надає.

Локалізація додатку реалізована за допомогою локальних файлів у форматі JSON (JavaScript Object Notation), які містять переклади текстових елементів інтерфейсу для різних мов. Кожен мовний файл структурований у вигляді ключ-значення, де ключі є унікальними ідентифікаторами текстових елементів, а значення - відповідними перекладами. Схема взаємодії React з i18next зображена на рис. 2.2.

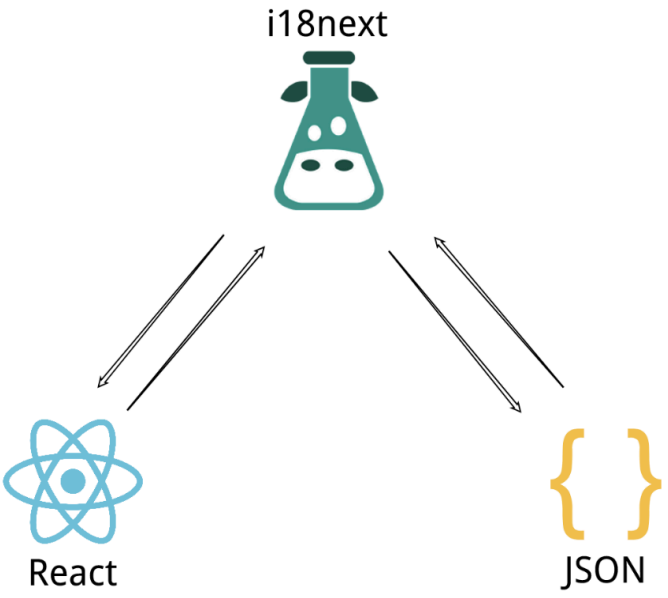


Рисунок 2.2. Схема взаємодії React з i18next

Для інтеграції i18next з React було використано додатковий модуль react-i18next, який забезпечує зручні інтеграційні компоненти та хуки для роботи з локалізацією у React-компонентах.

Після інтеграції локалізації необхідно провести тестування для перевірки коректного відображення текстів на різних мовах. Це включає перевірку всіх інтерфейсних елементів, повідомлень про помилки та інших текстових компонентів додатку.

Локалізація є важливим аспектом розробки сучасних веб-додатків, що дозволяє забезпечити доступність системи для широкого кола користувачів. Використання бібліотеки i18next у поєднанні з React дозволяє легко та ефективно реалізувати підтримку різних мов у додатку, що підвищує його зручність та привабливість для користувачів з різних країн.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Створення та налаштування проєкту

3.1.1 Створення проєкту

Розробка розпочинається з вибору середовища розробки. Я використовую VSCode (Visual Studio Code), яке відоме своєю зручністю, широкими можливостями налаштування та підтримкою численних розширень для ефективної роботи з веб-застосунками. Першим чином виберемо директорію для проєкту у VSCode (рис. 3.1).

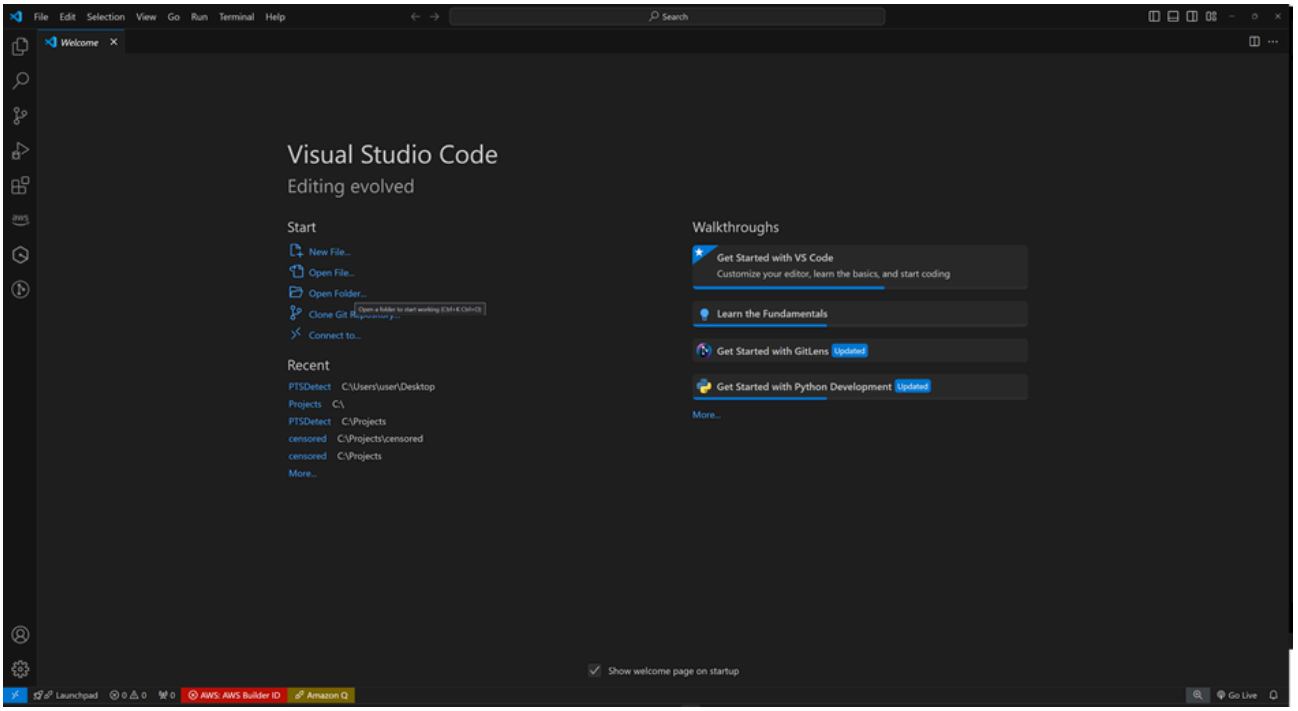


Рисунок 3.1. Вибір директорії проєкту

Для ініціалізації проєкту з використанням Vite, необхідно виконати команду (рис. 3.2.), де «PTSDetect» - це ім'я директорії проєкту, а «--template react-ts» - вказує на те, що ми хочемо створити проєкт на основі шаблону для React з TypeScript.

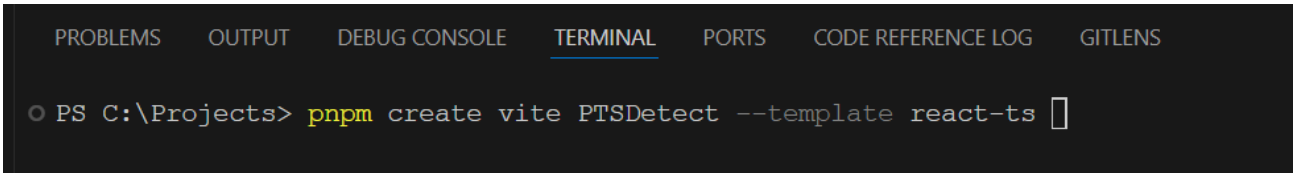


Рисунок 3.2. Команда для ініціалізації проєкту

В результаті виконанні команди ми отримали автоматично створений React проєкт з підтримкою TypeScript. Результат зображений на рис. 3.3.

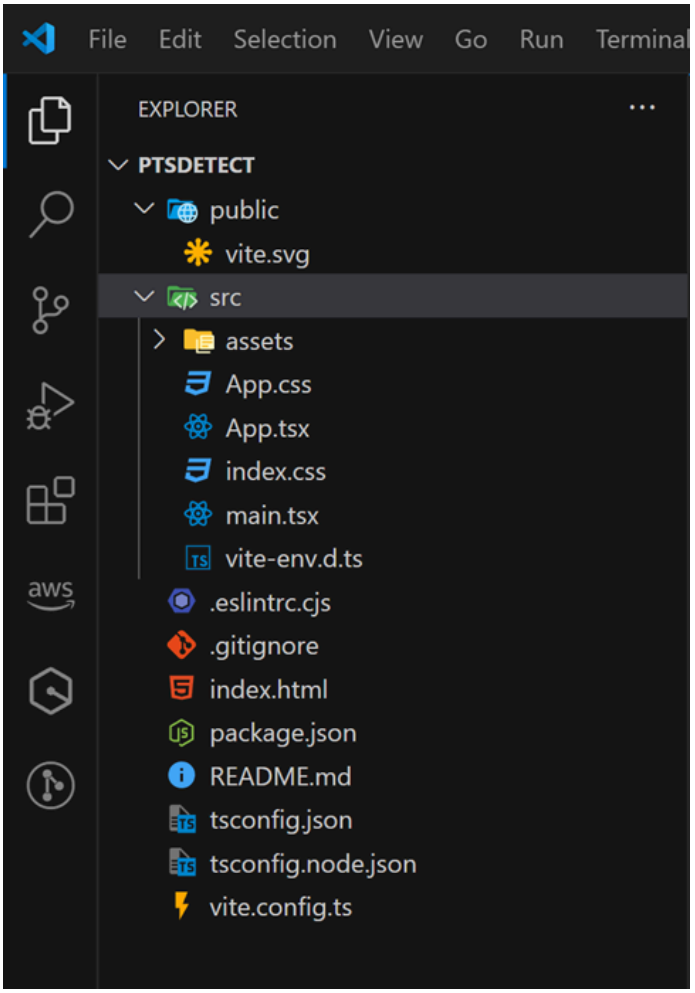


Рисунок 3.3. Результат ініціалізації проєкту

3.1.2 Встановлення основних залежностей

Наступним етапом необхідно встановити всі залежності проєкту, використовуючи PNMP – швидкий менеджер пакетів для JavaScript, який дозволяє ефективно працювати з залежностями, зменшуючи дублювання та покращуючи продуктивність. PNPM дозволяє розробникам швидко і зручно використовувати сторонні бібліотеки та інші ресурси, що сприяє покращенню продуктивності та якості розробки програмного забезпечення на платформі React.

Для початку необхідно завантажити та встановити PNMP. Це можна зробити за допомогою NPM (менеджера пакетів, який постачається разом з Node.js).

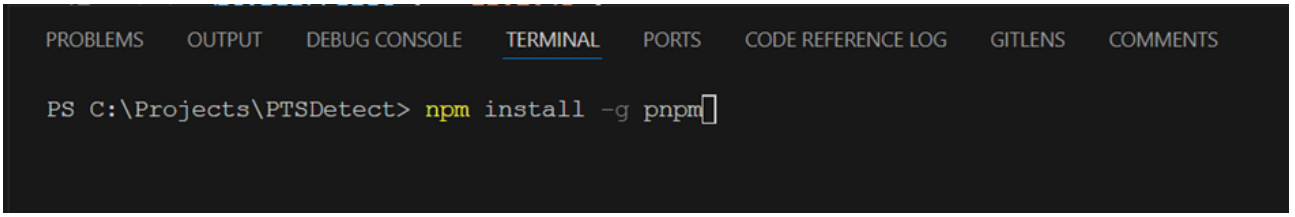


Рисунок 3.4. Команда для встановлення PNPM

Після встановлення PNPM, можемо приступити до встановлення основних залежностей проєкту. Команда для встановлення основних залежностей проєкту зображена на рис. 3.5, встановлені залежності описані в табл. 3.1.

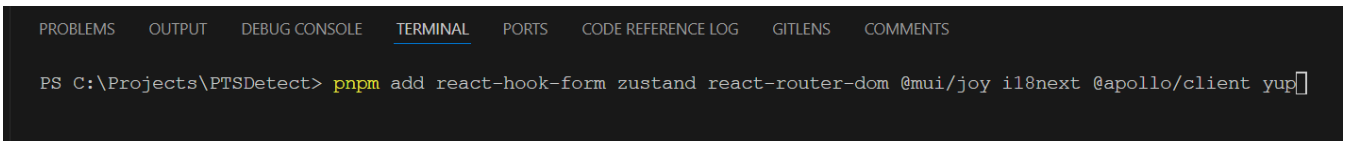


Рисунок 3.5. Команда для встановлення основних залежностей

Таблиця 3.1. Опис встановлених залежностей

Бібліотека	Опис
react-hook-form	Робота з формами
zustand	Робота зі станом
react-router-dom	Маршрутизація
@mui/joy	UI компоненти
i18next	Локалізація
@apollo/client	Робота з GraphQL
yup	Валідація схем

3.1.3 Підключення та налаштування залежностей для розробки

Для ефективної розробки сучасного веб-додатку на основі React необхідно використовувати різноманітні інструменти та бібліотеки, які допомагають оптимізувати процес розробки, забезпечити високу якість коду та полегшити підтримку проєкту. У цьому розділі ми розглянемо встановлення та налаштування основних залежностей для розробки, таких як Tailwind CSS, Prettier, ESLint та TypeScript, які використовуються в нашому проєкті.

Tailwind CSS – це утилітарний CSS фреймворк, який дозволяє швидко створювати сучасні інтерфейси користувача. Він надає набір готових класів, які можна комбінувати для створення стилів без написання кастомного CSS [8]. Команда для встановлення зображена на рис. 3.6, файл конфігурації зображений на рис. 3.7.

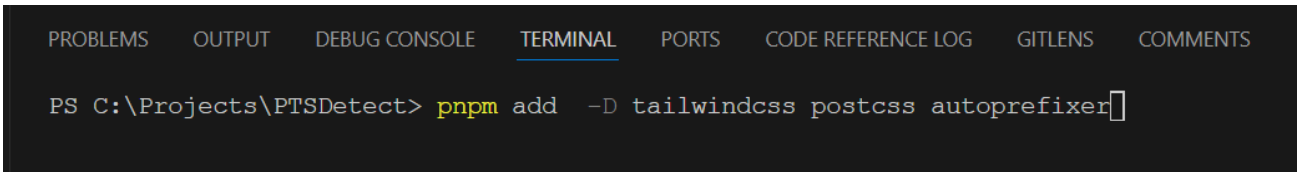


Рисунок 3.6. Команда для встановлення Tailwind CSS

Рисунок 3.7. Файл конфігурації Tailwind CSS

Prettier – це форматувач коду, який забезпечує єдиний стиль написання коду у всьому проєкті. Він автоматично форматує код відповідно до заданих правил, що дозволяє уникнути стилістичних помилок та покращує читабельність коду [9]. Команда для встановлення зображена на рис. 3.8, файл конфігурації зображений на рис. 3.9.

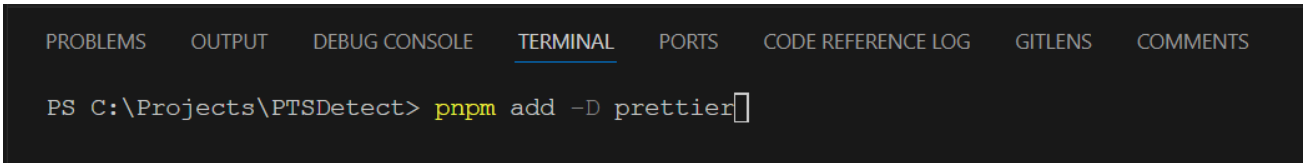


Рисунок 3.8. Команда для встановлення Prettier

Рисунок 3.9. Файл конфігурації Prettier

ESLint – це інструмент для аналізу коду, який допомагає знайти та виправити проблеми в JavaScript коді. Він забезпечує дотримання кращих практик написання коду, що підвищує його якість та знижує кількість помилок. Команда для встановлення зображена на рис. 3.10, файл конфігурації зображений на рис. 3.11.

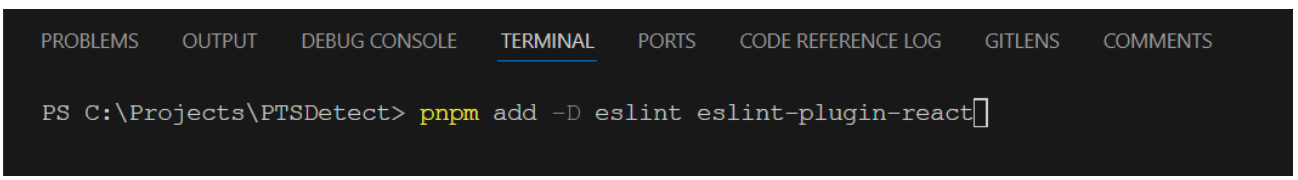
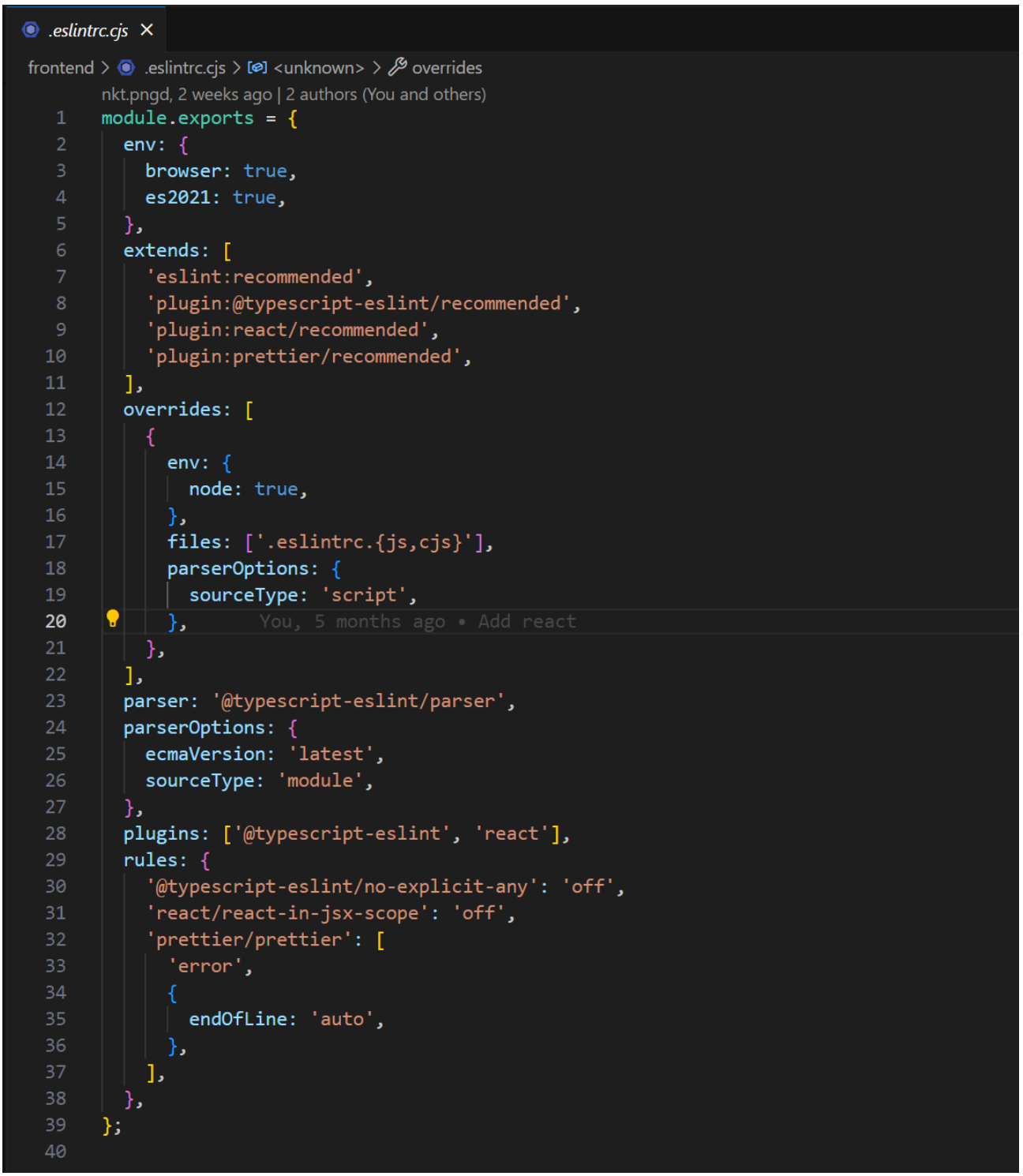


Рисунок 3.10. Команда для встановлення ESLint



```
.eslintrc.cjs
frontend > .eslintrc.cjs > <unknown> > overrides
nkt.pngd, 2 weeks ago | 2 authors (You and others)
module.exports = {
  env: {
    browser: true,
    es2021: true,
  },
  extends: [
    'eslint:recommended',
    'plugin:@typescript-eslint/recommended',
    'plugin:react/recommended',
    'plugin:prettier/recommended',
  ],
  overrides: [
    {
      env: {
        node: true,
      },
      files: ['.eslintrc.{js,cjs}'],
      parserOptions: {
        sourceType: 'script',
      },      You, 5 months ago • Add react
    },
  ],
  parser: '@typescript-eslint/parser',
  parserOptions: {
    ecmaVersion: 'latest',
    sourceType: 'module',
  },
  plugins: ['@typescript-eslint', 'react'],
  rules: {
    '@typescript-eslint/no-explicit-any': 'off',
    'react/react-in-jsx-scope': 'off',
    'prettier/prettier': [
      'error',
      {
        endOfLine: 'auto',
      },
    ],
  },
};
```

Рисунок 3.11. Файл конфігурації ESLint

TypeScript – це мова програмування, яка є надбудовою над JavaScript і додає статичну типізацію. Використання TypeScript допомагає виявляти помилки на етапі компіляції, що підвищує надійність та підтримуваність коду.

Рисунок 3.12. Файл конфігурації TypeScript

3.1.4 Інтеграція та налаштування GraphQL

GraphQL – це сучасна технологія для роботи з API, яка дозволяє клієнтам запитувати лише ті дані, які їм потрібні, замість отримання надмірних або недостатніх даних, як це часто буває з REST API. GraphQL надає більш гнучкий і ефективний спосіб взаємодії з даними, завдяки чому зростає популярність цієї

технології у розробників. У цьому розділі ми розглянемо інтеграцію GraphQL у наш веб-додаток, а також налаштування середовища для його використання.

Перед початком інтеграції GraphQL у ваш веб-додаток на React, необхідно встановити необхідні залежності. Це включає @apollo/client для Apollo Client та graphql для роботи з GraphQL [10].

Для інтеграції GraphQL з нашим додатком ми будемо використовувати Apollo Client. Apollo Client надає потужний і гнучкий інструментарій для роботи з GraphQL у фронтенді.

Створимо файл client.ts, який буде містити налаштування Apollo Client.

Для того щоб використовувати Apollo Client у вашому React-додатку, необхідно обгорнути кореневий компонент додатку в ApolloProvider.

GraphQL використовує запити (queries) для отримання даних і мутації (mutations) для їх зміни. Ми створимо окремі файли для запитів та мутацій.

Для покращення розробки і підтримки коду ми можемо використовувати GraphQL Code Generator для автоматичної генерації типів TypeScript.

Після налаштування, можна запустити генерацію типів за допомогою команди: «npx graphql-codegen».

Інтеграція та налаштування GraphQL у веб-додаток забезпечують більш ефективне та гнучке управління даними. Використовуючи Apollo Client, ми можете легко налаштувати запити та мутації, а GraphQL Code Generator допоможе підтримувати високий рівень типізації у нашому проекті. Це значно покращує зручність розробки та підтримки коду у довгостроковій перспективі.

3.1.5 Опис файлової структури

Файлова структура проєкту є важливою складовою успішного розроблення та підтримки програмного забезпечення. Вона визначає, як організовані файли та папки в проєкті, що полегшує навігацію, розуміння та управління кодовою базою. У цьому розділі ми опишемо файлову структуру проєкту, включаючи всі основні папки та файли, які використовуються для

розроблення, тестування та розгортання застосунку. Структура проєкту зображена на рис. 3.13 та описана в табл. 3.2.

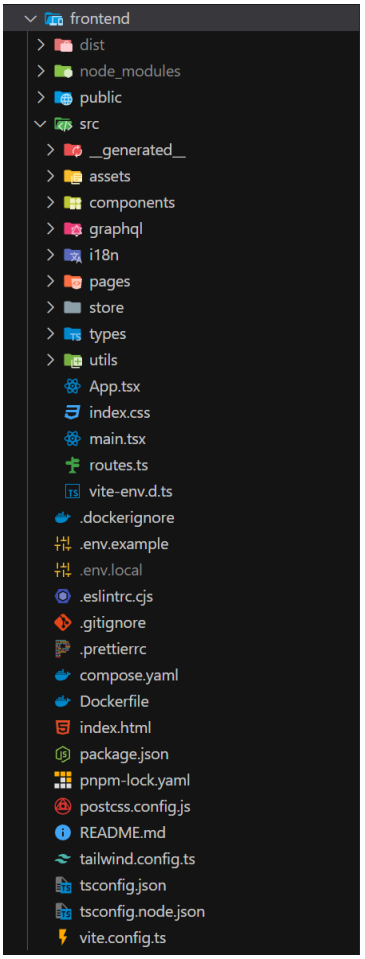


Рисунок 3.13. Файлова структура проєкту

Таблиця 3.2 Опис файлової структури проєкту

Структурна одиниця	Опис
dist	Директорія для зібраних (компільованих) файлів застосунку, готових для розгортання.
node_modules	Директорія, яка містить всі залежності проєкту, встановлені за допомогою npm або pnpm.

Продовження таблиці 3.2. Опис файлової структури проєкту

Структурна одиниця	Опис
public	Директорія для статичних файлів, таких як зображення, які будуть доступні безпосередньо через веб-сервер.
src	Основна директорія для вихідного коду застосунку.
generated	Директорія для автоматично згенерованого коду, наприклад, GraphQL типів.
assets	Директорія для зберігання ресурсів, таких як зображення, шрифти та інші статичні файли.
components	Директорія для React-компонентів.
i18n	Директорія для файлів локалізації та інтернаціоналізації.
graphql	Директорія для GraphQL-запитів та схем.
pages	Директорія для сторінок застосунку.
store	Директорія для стану (state) та конфігурації Zustand.

Продовження таблиці 3.2. Опис файлової структури проєкту

Структурна одиниця	Опис
types	Директорія для TypeScript типів та інтерфейсів.
utils	Директорія для утиліт та допоміжних функцій.
vite-env.d.ts	Файл для специфікації типів середовища Vite.
routes.ts	Файл, що містить маршрути застосунку.
main.tsx	Вхідний файл для React-застосунку.
index.css	Основний файл стилів.
App.tsx	Головний компонент застосунку.
.dockerignore	Файл, який визначає, які файли та папки ігнорувати при створенні Docker-образу.
.env.example	Приклад конфігураційного файлу середовища.
.env.local	Локальний конфігураційний файл середовища.
.eslintrc.cjs	Конфігураційний файл для ESLint.

Продовження таблиці 3.2. Опис файлової структури проєкту

Структурна одиниця	Опис
.gitignore	Файл, який визначає, які файли та папки ігнорувати у системі контролю версій Git.
.prettierrc	Конфігураційний файл для Prettier.
compose.yaml	Файл конфігурації для Docker Compose.
Dockerfile	Файл для створення Docker-образу.
index.html	Основний HTML-файл застосунку.
package.json	Файл, який містить метадані проєкту та список залежностей.
pnpm-lock.yaml	Файл блокування залежностей для pnpm.
postcss.config.js	Конфігураційний файл для PostCSS.
README.md	Файл з описом проєкту та інструкціями.
tailwind.config.ts	Конфігураційний файл для Tailwind CSS.
tsconfig.json	Конфігураційний файл для TypeScript.

Продовження таблиці 3.2. Опис файлової структури проєкту

Структурна одиниця	Опис
tsconfig.node.json	Додатковий конфігураційний файл для TypeScript, специфічний для Node.js.
vite.config.ts	Конфігураційний файл для Vite.

3.2 Розробка основного функціоналу клієнтської частини додатку

3.2.1 Реєстрація та аутентифікація користувача

Однією з ключових функцій будь-якого веб-додатку є забезпечення безпеки та конфіденційності даних користувачів. Для досягнення цієї мети необхідно реалізувати надійну систему реєстрації та аутентифікації користувачів.

У цьому розділі буде розглянуто процес розробки реєстрації та аутентифікації користувачів. Ми розглянемо реалізацію таких функцій, як реєстрація нового користувача, підтвердження електронної пошти, відновлення паролю та вхід до системи.

Реєстрація нового користувача є першим кроком у взаємодії з додатком. Вона включає в себе введення користувачем своїх облікових даних, таких як електронна пошта та пароль, а також їх валідацію. Для цього ми використовуємо бібліотеку Yup для валідації форм та React Hook Form для управління станом форми. Функціональність реєстрації нового користувача реалізована за допомогою компонента «SignUp» (Додаток А).

Основні етапи роботи компонента:

1. Оголошення схеми форми. Схема форми створюється за допомогою бібліотеки yup, яка визначає вимоги до валідації даних. Зокрема, електронна пошта повинна бути валідною та обов'язковою, пароль

повинен відповідати кільком критеріям складності, а повторений пароль має збігатися з основним паролем.

2. Ініціалізація форми. Використання хука useForm з бібліотеки react-hook-form для керування станом форми та валідації.
3. Обробка події відправки форми. Після валідації даних форма відправляється на сервер за допомогою функції registerUser з useMutation для реєстрації нового користувача через GraphQL запит.
4. Відображення форми. Використання компонентів з бібліотеки @mui/joy для створення зручного та інтуїтивно зрозумілого інтерфейсу для введення даних.

Схема потоків даних компоненту «SignUp» зображена на рис. 3.14.

Після реєстрації користувач повинен підтвердити свою електронну пошту. Це забезпечує додатковий рівень безпеки та гарантує, що користувач ввів правильну адресу електронної пошти.

Аутентифікації дозволяє користувачам увійти до системи, використовуючи свої облікові дані. Це включає в себе введення електронної пошти та паролю, а також їх валідацію. Функціональність аутентифікації реалізована за допомогою компонента «SignIn» (Додаток Б). Основні етапи роботи компонента схожі з реєстрацією.

Функція відновлення паролю дозволяє користувачам відновити доступ до свого облікового запису у випадку, якщо вони забули свій пароль. У нашому випадку, ми реалізуємо функціонал відновлення паролю за допомогою двох основних компонентів: «ForgotPassword» (Додаток В) та «ResetPassword» (Додаток Г).

Компонент «ForgotPassword» відповідає за відправку запиту на скидання паролю на електронну пошту користувача. Він включає форму, де користувач може ввести свою електронну адресу, а також логіку для обробки цього запиту.

Компонент «ResetPassword» відповідає за встановлення нового паролю користувачем після того, як він пройшов верифікацію через отриманий на

електронну пошту посилання. Він включає форму для введення нового паролю та повторного введення паролю для підтвердження.

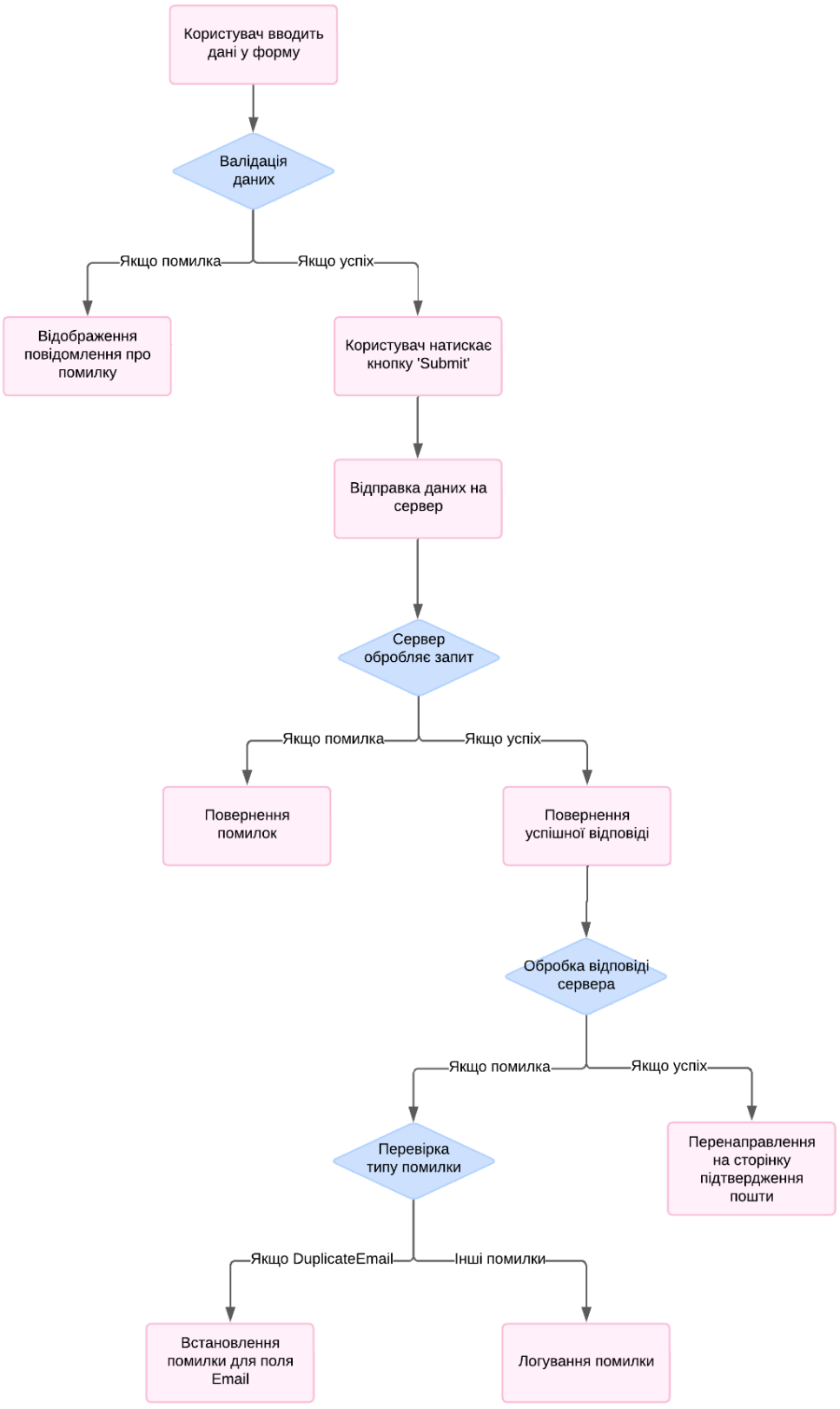


Рисунок 3.14. Схема потоків даних компоненту «SignUp»

3.2.2 Проходження загального тестування

У цьому розділі ми розглянемо реалізацію функціоналу проходження загального тестування. Основна мета цього функціоналу – забезпечити користувачам можливість проходити тестування, та отримувати результати після завершення тесту. Цей функціонал реалізований за допомогою компонента «GeneralTest» (Додаток Д).

Основні етапи роботи компонента:

1. Ініціалізація та завантаження запитань:
   - Ініціалізація стану. Використовується хук useState для створення стану компоненту, включаючи поточне запитання, список запитань, індикатор завантаження тощо;
   - Завантаження запитань. Використовується хук useLazyQuery з бібліотеки Apollo Client для виконання GraphQL-запиту, який отримує перше запитання з сервера.
2. Відображення запитань:
   - Рендеринг запитань. Після отримання запитань від сервера, вони зберігаються у стані компоненту та відображаються на екрані;
   - Форма для відповідей. Використовується бібліотека react-hook-form для керування формою та збору відповідей користувача.
3. Навігація між запитаннями:
   - Перехід до наступного запитання. Користувач натискає кнопку "Далі" для завантаження наступного запитання. Якщо це не останнє запитання, виконується новий GraphQL-запит для отримання наступного запитання;
   - Завершення тестування. Якщо це останнє запитання, кнопка змінюється на "Завершити", що дозволяє завершити тестування.

4. Обробка події відправки форми:
    - Збір відповідей. Після завершення тестування всі відповіді користувача збираються та формуються у відповідний формат;
    - Надсилання відповідей. Використовується хук useMutation з бібліотеки Apollo Client для надсилання відповідей на сервер через GraphQL-мутацію.
5. Отримання та відображення результатів:
    - Перенаправлення на сторінку результатів. Після успішного надсилання відповідей користувач перенаправляється на сторінку результатів тестування.

Ці етапи забезпечують повний цикл проходження загального тестування, починаючи від завантаження запитань і закінчуючи відображенням результатів.

3.2.3 Відображення результату тестування

У цьому розділі буде розглянуто розробку функціоналу відображенню результатів тестування. Веб-додаток надає користувачам можливість проходити загальне тестування, результат якого включає потенційні проблеми та рекомендації щодо їх вирішення. Відображення результатів тестування є важливою складовою інтерфейсу користувача, що дозволяє зручно й інформативно представити отримані дані.

Компонент «GeneralTestResult» (Додаток Е) відповідає за відображення результатів тестування користувача, включаючи потенційні проблеми та поради щодо їх вирішення. Основні етапи роботи цього компонента наведені нижче:

1. Ініціалізація стану та параметрів. Використання хуків useState для стану результату та індексу активної вкладки, а також useParams для отримання ідентифікатора результату тестування з URL.
2. Запит на отримання результату тестування. Виконання GraphQL-запиту з використанням useLazyQuery та збереження результату в стані компонента.

3. Відображення стану завантаження. Показ індикатора завантаження під час отримання результатів.
4. Відображення результатів тестування:
    - Якщо потенційні проблеми відсутні, відображається відповідне повідомлення;
    - Якщо є потенційні проблеми, вони відображаються разом з порадами щодо їх вирішення.

Цей компонент є важливою частиною інтерфейсу користувача, він забезпечує зручне та інформативне відображення результатів тестування.

3.2.4 Редагування профілю користувача

Редагування профілю користувача є важливою частиною будь-якого веб-додатку, що дозволяє користувачам оновлювати свою персональну інформацію та налаштування.

Компонент «Profile» (Додаток Є) рендерить форму, яка дозволяє користувачу редагувати свої персональні дані, такі як ім'я, прізвище, дата народження, стать та статус шлюбу. Також користувач може змінити свій аватар, натиснувши на кнопку редагування та вибравши новий файл. Після внесення змін користувач може зберегти їх або скасувати.

Основні етапи роботи компонента включають ініціалізацію станів та хуків, обробку змін аватара, валідацію та відправку форми, а також рендеринг інтерфейсу користувача.

3.3 Впровадження локалізації

Локалізація є важливим аспектом розробки веб-додатків, оскільки вона дозволяє користувачам взаємодіяти з додатком на їхній рідній мові. Це не лише підвищує зручність використання, але й сприяє залученню ширшої аудиторії. У цьому розділі буде описано процес впровадження локалізації у веб-додаток на React за допомогою бібліотеки i18next.

Для реалізації локалізації у веб-додатку було обрано бібліотеку i18next. Ця бібліотека є популярним рішенням для інтернаціоналізації (i18n) у JavaScript-додатках і має гарну інтеграцію з React через плагін react-i18next.

Налаштування i18next:

1. Імпорт необхідних модулів та перекладів. Необхідно імпортувати i18next, плагін initReactI18next, а також JSON-файли з перекладами для кожної мови.
2. Створення об'єкта ресурсів. Створюється об'єкт ресурсів, який містить переклади для кожної мови.
3. Ініціалізація i18next. Наступним кроком є ініціалізація i18next з використанням initReactI18next та налаштуванням початкової мови, яка зберігається у localStorage.

Для зручності користувачів у інтерфейсі додатка було додано компонент для вибору мови. Компонент «LanguageSelect» (Додаток Ж) відображає випадаючий список з мовами та дозволяє користувачу змінювати мову інтерфейсу

Впровадження локалізації у веб-додаток на React за допомогою бібліотеки i18next є простим та ефективним способом забезпечити багатомовність інтерфейсу. Використання JSON-файлів для зберігання перекладів та додавання компонента для вибору мови робить додаток зручним для користувачів з різних країн.

3.4 Налаштування конфігурації проєкту

Для того щоб забезпечити коректну роботу веб-додатку, важливо правильно налаштувати конфігурацію проєкту. У цьому розділі буде розглянуто налаштування конфігурації проєкту, створеного на основі React, з використанням Vite як інструменту для побудови. Vite пропонує швидке і сучасне рішення для розробки, що включає підтримку модулів ES і миттєве

завантаження. Основні моменти, які ми розглянемо, включають налаштування плагінів, шляхи до конфігураційних файлів та налаштування сервера.

Для конфігурації проєкту на основі Vite використовується файл vite.config.ts. Наведений нижче приклад коду (рис. 3.15) демонструє, як налаштувати Vite для роботи з React, обробки шляхів до конфігураційних файлів TypeScript та підтримки SVG.

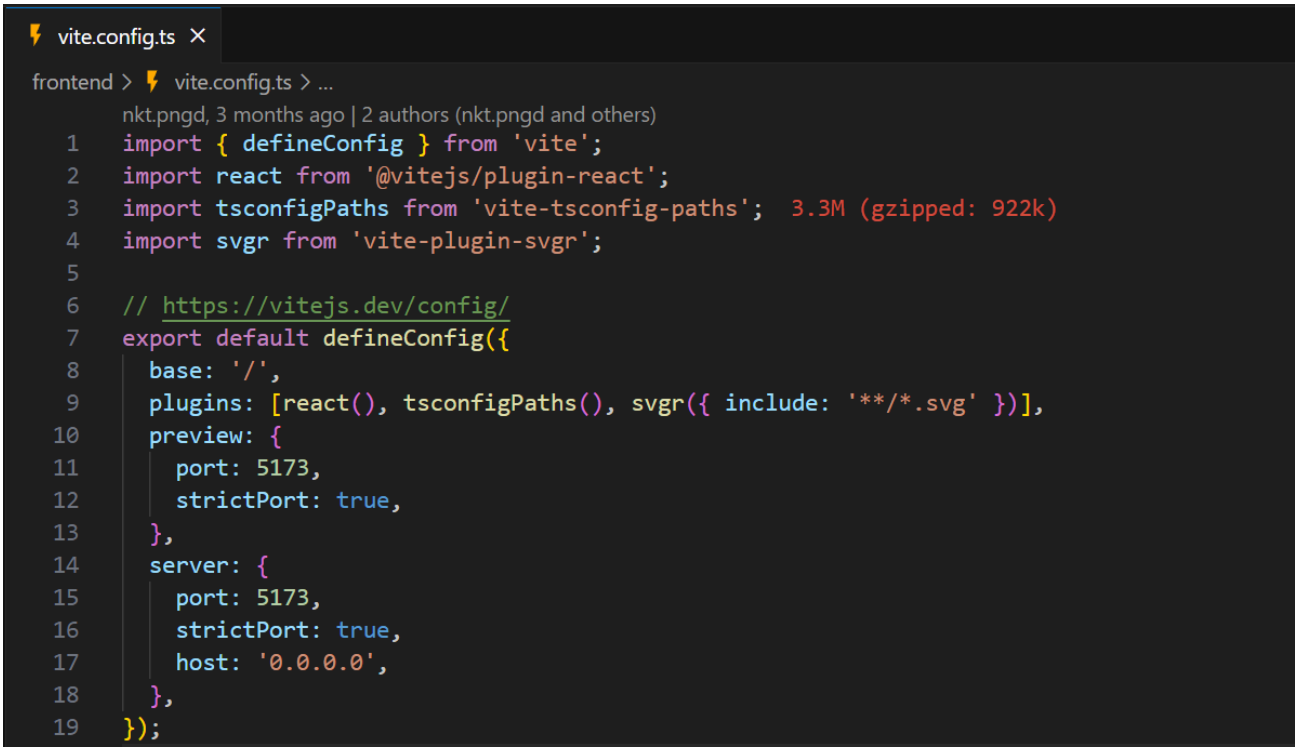


Рисунок 3.15. Файл конфігурації Vite

Опис конфігурації:

1. Базовий шлях (base: '/'):
   - Визначає базовий шлях для проєкту, який використовується для всіх абсолютних URL.
2. Плагіни (plugins: [react(), tsconfigPaths(), svgr({ include: '**/*.svg' })]):
   - react. Плагін для підтримки React у Vite, що забезпечує швидку компіляцію та підтримку сучасних можливостей React;
   - tsconfigPaths. Плагін, який дозволяє використовувати шляхи, визначені у tsconfig.json, для зручної навігації по проєкту;
   - svgr. Плагін для імпорту SVG файлів як React компоненти, що дозволяє легко використовувати SVG в інтерфейсі користувача.

3. Налаштування сервера для попереднього перегляду (preview):
    - port. Вказує порт, на якому буде працювати сервер попереднього перегляду;
    - strictPort. Забезпечує використання саме вказаного порту. Якщо порт зайнятий, сервер не запуститься.
4. Налаштування сервера розробки (server):
    - port. Вказує порт для сервера розробки;
    - strictPort. Забезпечує використання саме вказаного порту. Якщо порт зайнятий, сервер не запуститься;
    - host. Вказує серверу прослуховувати всі мережеві інтерфейси, що дозволяє доступ до нього з інших пристроїв у мережі.

Використання Vite для конфігурації додатку значно спрощує процес розробки завдяки швидкості збірки та простоті налаштувань. Вказана конфігурація охоплює всі основні аспекти налаштування проєкту, включаючи підтримку сучасних плагінів та гнучкість у налаштуванні серверів для різних етапів розробки та тестування.

3.5 Контейнеризація додатку

Контейнеризація додатків є сучасною практикою в розробці програмного забезпечення, яка дозволяє ізолювати додаток з усіма його залежностями в окремий контейнер. Це забезпечує консистентність, портативність та спрощує розгортання додатків у різних середовищах. Використання контейнерів допомагає уникнути проблем з сумісністю між різними середовищами розробки, тестування та продуктивного середовища. В цьому розділі розглянемо процес контейнеризації веб-додатку, використовуючи Docker [11].

Dockerfile містить інструкції для створення образу Docker (рис 3.16), який включає всі необхідні для роботи додатку залежності та конфігурації.

Опис конфігурації Dockerfile:

- «FROM node:18-alpine». Вказує на використання базового образу Node.js на базі Alpine Linux, який є легким та оптимізованим для роботи в контейнерах;
- «ARG VITE_GRAPHQL_URI». Оголошує аргумент для передачі URI GraphQL сервера, що буде використовуватись під час побудови образу;
- «WORKDIR /app». Встановлює робочу директорію всередині контейнера;
- «COPY package.json ./». Копіює файл package.json до робочої директорії контейнера;
- «RUN npm install». Виконує встановлення залежностей, зазначених у package.json;
- «COPY . .». Копіює всі файли з поточної директорії на хост машині до робочої директорії в контейнері;
- «RUN npm run build». Збирає додаток для продуктивного середовища;
- «EXPOSE 5173». Відкриває порт 5173 для доступу до додатку;
- «CMD ["npm", "run", "preview"]». Вказує команду, яка буде виконана при запуску контейнера - запуск додатку у режимі попереднього перегляду.

Рисунок 3.16. Конфігурація Dockerfile

Файл docker-compose.yml (рис 3.17) дозволяє нам визначити та запускати багатоконтейнерні Docker додатки. У нашому випадку він містить конфігурацію для сервісу frontend.

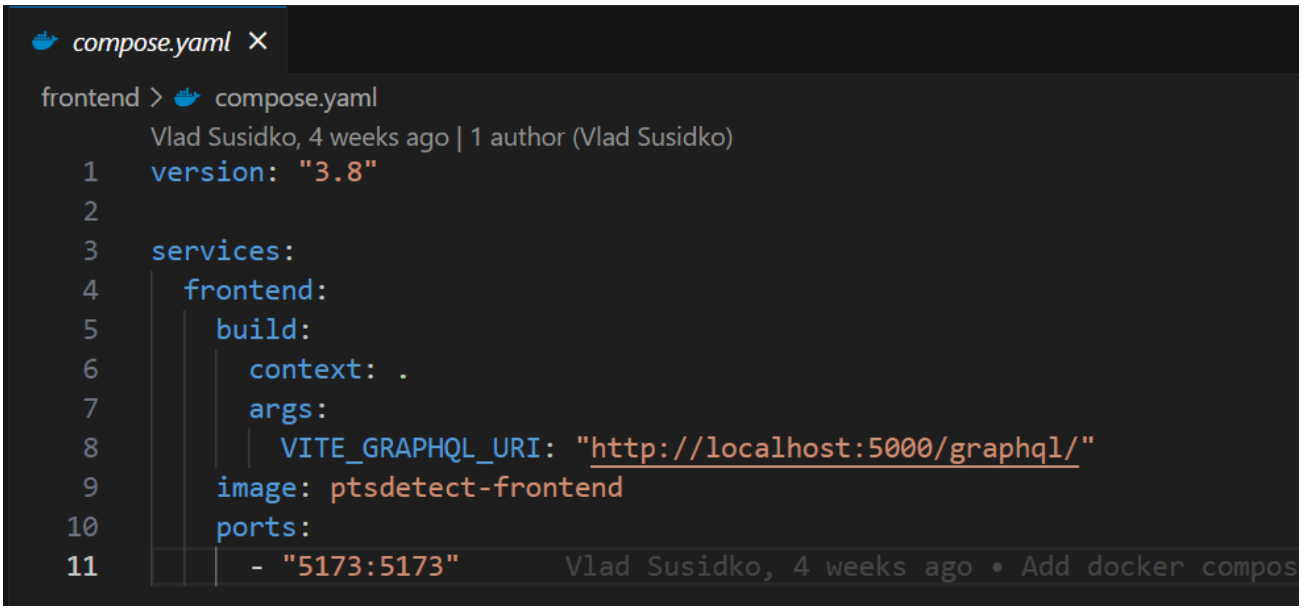


Рисунок 3.17. Конфігурація Docker Compose

Опис конфігурації Docker Compose:

1. version. Вказує версію Docker Compose, яку ми використовуємо.
2. services. Визначає сервіси, які будуть запущені.
3. frontend. Конфігурація сервісу для нашого React додатку.
    - build. Визначає параметри для побудови образу;
    - context. Вказує на поточну директорію як контекст побудови;
    - args. Передає аргумент VITE_GRAPHQL_URI для побудови образу;
    - image. Вказує ім'я образу, який буде створено;
    - ports. Вказує порти для зв'язку між контейнером та хостом.

Таким чином, за допомогою Dockerfile та docker-compose.yml ми можемо легко контейнеризувати наш веб-додаток на React, що забезпечує простоту його розгортання та підтримки у різних середовищах.

4 ДИЗАЙН ІНТЕРФЕЙСУ КОРИСТУВАЧА

4.1 Основи UI/UX дизайну

UI (User Interface) та UX (User Experience) дизайн є двома взаємопов'язаними, але різними аспектами створення цифрових продуктів. UI дизайн зосереджений на візуальному оформленні та інтерактивних елементах, тоді як UX дизайн спрямований на загальний досвід користувача, включаючи зручність, ефективність та задоволення від використання продукту [12].

Основні принципи UX дизайну:

1. Дослідження користувачів:
    - Персонажі (Personas). Персонажі є вигаданими представниками цільової аудиторії, створеними на основі реальних даних про користувачів. Вони допомагають дизайнерам зрозуміти потреби, поведінку та цілі різних користувачів;
    - Користувацькі сценарії (User Stories). Користувацькі сценарії описують, як різні типи користувачів будуть взаємодіяти з продуктом для досягнення своїх цілей. Це допомагає визначити функціональні вимоги та пріоритети;
    - Аналітика та опитування. Збір даних про поведінку користувачів за допомогою аналітичних інструментів та опитувань дозволяє краще розуміти, як користувачі взаємодіють з продуктом і які проблеми вони стикаються.
2. Інформаційна архітектура:
    - Картографія сайту (Sitemap). Картографія сайту є візуальним представленням структури сайту або додатку, що показує, як різні сторінки та розділи взаємопов'язані. Це допомагає дизайнерам створювати логічну та інтуїтивно зрозумілу навігацію;
    - Карти користувацьких шляхів (User Journey Maps). Карти користувацьких шляхів описують маршрути, які користувачі

проходять для досягнення своїх цілей. Вони допомагають виявити можливі перешкоди та покращити загальний досвід користувача.

3. Прототипування та тестування:
   - Wireframes. Низькодеталізовані макети, що показують основну структуру сторінок без детального дизайну. Вони допомагають визначити розташування елементів та їх взаємозв'язки;
   - Прототипи. Інтерактивні моделі, що імітують функціональність продукту. Прототипи дозволяють тестувати взаємодію користувачів з продуктом на ранніх етапах розробки;
   - Юзабіліті-тестування (Usability Testing). Перевірка прототипів з реальними користувачами для виявлення проблем у зручності використання та ефективності дизайну.

Основні принципи UI дизайну:

1. Візуальна ієрархія. Використання розміру, кольору та розташування для привернення уваги до важливих елементів. Візуальна ієрархія допомагає користувачам швидко орієнтуватися на сторінці та знаходити необхідну інформацію.
2. Консистентність. Використання однакових стилів, шрифтів та кольорів для створення єдиного вигляду та відчуття. Консистентність забезпечує зручність використання та сприяє впізнаваності бренду.
3. Колірна палітра та типографіка. Вибір кольорів та шрифтів, що відповідають бренду та забезпечують зручність читання. Правильна колірна палітра та типографіка можуть значно покращити сприйняття інформації та загальний досвід користувача.
4. Інтерактивні елементи. Дизайн кнопок, посилань, форм та інших елементів, з якими взаємодіють користувачі. Інтерактивні елементи повинні бути інтуїтивно зрозумілими та легко доступними.
5. Адаптивний дизайн. Забезпечення зручності використання продукту на різних пристроях та екранах. Адаптивний дизайн дозволяє створювати інтерфейси, що автоматично підлаштовуються під розмір

екрану, забезпечуючи оптимальний досвід на мобільних телефонах, планшетах та комп'ютерах.

UI/UX дизайн є критично важливим для створення успішних цифрових продуктів. Від розуміння потреб користувачів до створення інтуїтивно зрозумілих інтерфейсів, кожен етап процесу вимагає ретельного планування та тестування. Використання відповідних інструментів та методів допоможе забезпечити високу якість кінцевого продукту, що задовольнить потреби користувачів та досягне бізнес-цілей.

4.2 Дослідження користувачів та вимоги до інтерфейсу

Успіх будь-якого веб-додатку значною мірою залежить від його здатності задовольняти потреби та очікування своїх користувачів. Особливо це важливо у випадку систем, які мають справу з чутливими даними, такими як психологічний стан людей. Наш веб-додаток має на меті забезпечити зручний та безпечний інструмент для проведення тестування, аналізу результатів та моніторингу динаміки психологічного стану користувачів.

Цей розділ присвячений дослідженню потреб цільової аудиторії, аналізу конкурентних рішень та визначенню вимог до інтерфейсу веб-додатку. На основі проведених досліджень будуть сформульовані рекомендації щодо дизайну інтерфейсу та функціональних можливостей системи, що сприятимуть підвищенню її зручності, надійності та ефективності.

Дослідження користувачів:

1. Цільова аудиторія:
    - Пацієнти. Люди, які хочуть оцінити свій психологічний стан;
    - Психологи та психотерапевти. Професіонали, які використовують систему для діагностики та моніторингу стану своїх пацієнтів;
    - Адміністратори. Технічні спеціалісти, які підтримують роботу системи.
2. Потреби користувачів:

- Пацієнти:
  - Простота використання. Інтерфейс має бути інтуїтивно зрозумілим;
  - Конфіденційність. Захист особистих даних та результатів тестування;
  - Надійність. Тести повинні бути науково обґрунтованими та точними.
- Психологи та психотерапевти:
  - Доступ до детальної інформації. Можливість перегляду детальних результатів тестів;
  - Інструменти для аналізу. Функції для порівняння результатів та відстеження динаміки стану пацієнтів.
- Адміністратори :
  - Легкість у підтримці. Зручність у налаштуванні та оновленні системи;
  - Безпека. Захист даних від несанкціонованого доступу.

3. Методи дослідження:
   - Опитування. Проведення анкетування серед потенційних користувачів для виявлення їх потреб та очікувань;
   - Інтерв'ю. Глибинні інтерв'ю з психологами та пацієнтами для розуміння специфічних вимог;
   - Аналіз конкурентів. Вивчення існуючих рішень на ринку для виявлення їх сильних та слабких сторін.

Вимоги до інтерфейсу:

1. Загальні вимоги:
   - Інтуїтивність. Інтерфейс має бути зрозумілим для користувачів з різним рівнем технічної підготовки;
   - Мінімалізм. Використання простих та зрозумілих елементів дизайну;

- Доступність. Забезпечення доступу для людей з обмеженими можливостями.

2. Функціональні вимоги:
   - Реєстрація та авторизація. Простий та безпечний процес реєстрації та входу в систему;
   - Проходження тестів. Зручний інтерфейс для проходження тестів;
   - Перегляд результатів. Інформативний та зрозумілий відображення результатів тестування.
3. Нефункціональні вимоги:
   - Безпека. Захист даних користувачів від несанкціонованого доступу та витоку інформації;
   - Швидкодія. Оптимізація швидкості завантаження сторінок та обробки даних;
   - Сумісність. Підтримка різних браузерів та пристроїв.

У результаті проведеного дослідження користувачів та аналізу їх потреб, було визначено основні вимоги до інтерфейсу веб-додатку. Ці вимоги включають інтуїтивність, мінімалізм, доступність, безпеку та швидкодію системи. Особливу увагу було приділено забезпеченню конфіденційності та надійності обробки даних, що є критично важливими для користувачів, які проходять психологічне тестування.

Проведені опитування та інтерв'ю з представниками цільової аудиторії дозволили виявити ключові функціональні та нефункціональні вимоги до системи, що включають зручний процес реєстрації та авторизації, можливість збереження проміжних результатів тестування, інформативний перегляд результатів.

Узагальнюючи результати дослідження, можна зробити висновок, що створений інтерфейс веб-додатку відповідає потребам різних категорій користувачів та забезпечує зручність, надійність та ефективність його використання. Подальші покращення можуть бути спрямовані на розширення

функціональних можливостей системи та підвищення її адаптивності до змінних потреб користувачів.

4.3 Дизайн сторінок додатку

4.3.1 Сторінки реєстрації та аутентифікації

Одним із ключових аспектів успішного функціонування додатків є забезпечення зручного та безпечного процесу реєстрації та аутентифікації користувачів. У даному розділі розглядається дизайн сторінок реєстрації та аутентифікації.

Сторінка аутентифікації повинна бути безпечною та зручною. Вона повинна включати поля для введення електронної пошти та пароля, а також можливість відновлення пароля у разі його втрати. Також важливо забезпечити швидкий та зручний доступ до функції відновлення пароля, щоб користувачі могли легко відновити доступ до свого акаунту у разі потреби. Дизайн сторінки аутентифікації зображений на рис. 4.1.

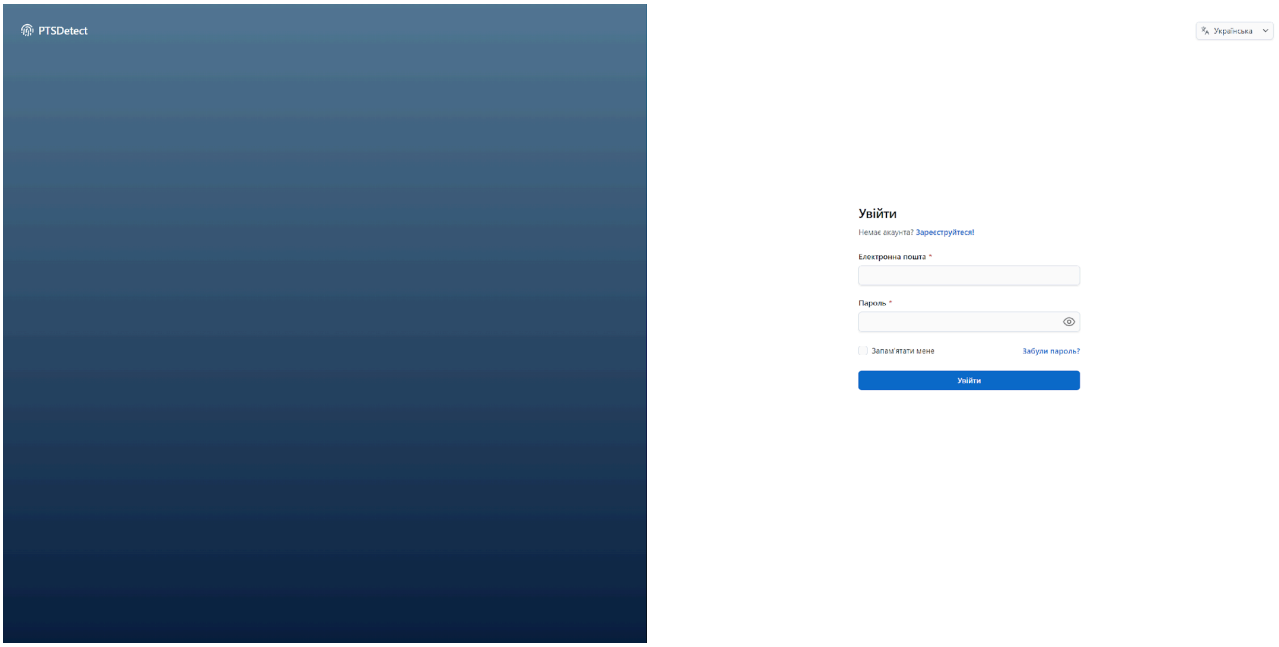


Рисунок 4.1. Дизайн сторінки аутентифікації

Сторінка реєстрації повинна бути простою та зрозумілою. Вона повинна включати поля для введення необхідної інформації, такої як електронна пошта та пароль. Важливо забезпечити зворотний зв'язок у разі помилок введення,

наприклад, підсвічування полів з помилками та відповідні повідомлення. Це дозволяє користувачам швидко і легко виправити помилки та завершити процес реєстрації. Дизайн сторінки реєстрації зображений на рис. 4.2.

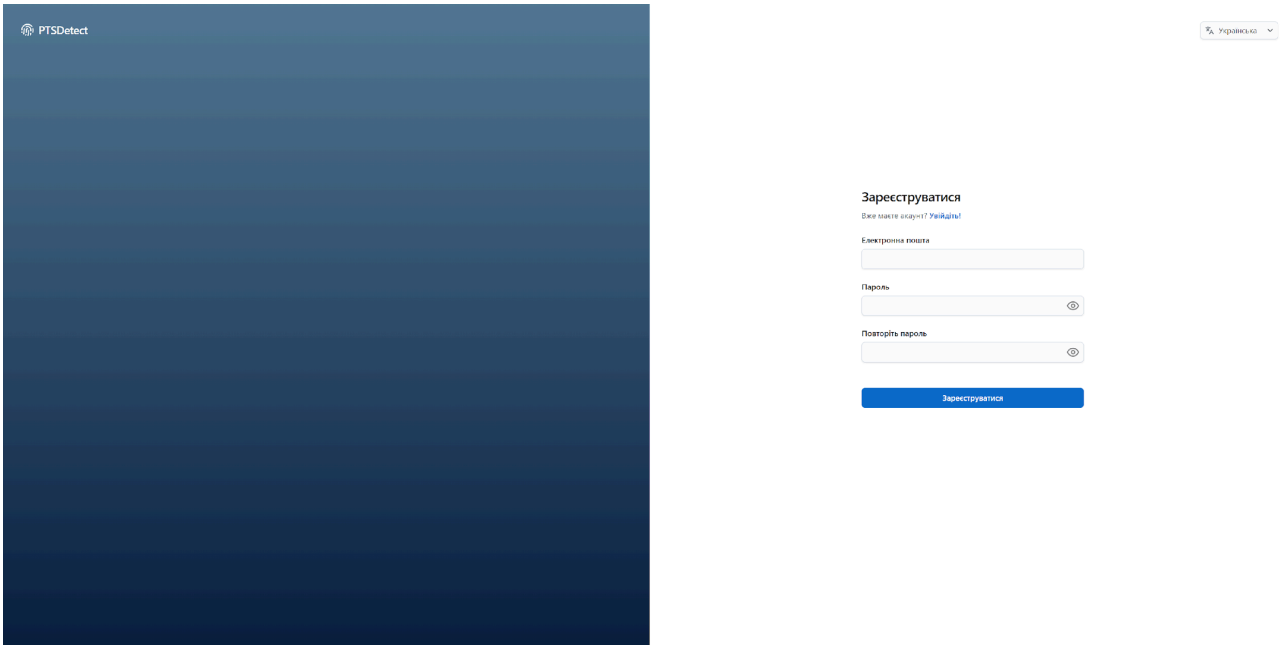


Рисунок 4.2. Дизайн сторінки реєстрації

4.3.2 Головна сторінка

Головна сторінка веб-додатку є першою точкою взаємодії користувача з системою. Вона повинна створювати позитивне перше враження, бути зрозумілою та привабливою, а також надавати користувачеві необхідну інформацію та можливості для подальшої взаємодії з додатком. Ця сторінка відіграє ключову роль у залученні користувачів та забезпеченні їхньої зручності при початку тестування.

Головна сторінка повинна бути мінімалістичною та функціональною, щоб користувачі могли швидко зрозуміти її призначення та легко знайти необхідні елементи. Основними компонентами головної сторінки є текст та кнопка для початку тестування. Дизайн головної сторінки зображений на рис. 4.3.

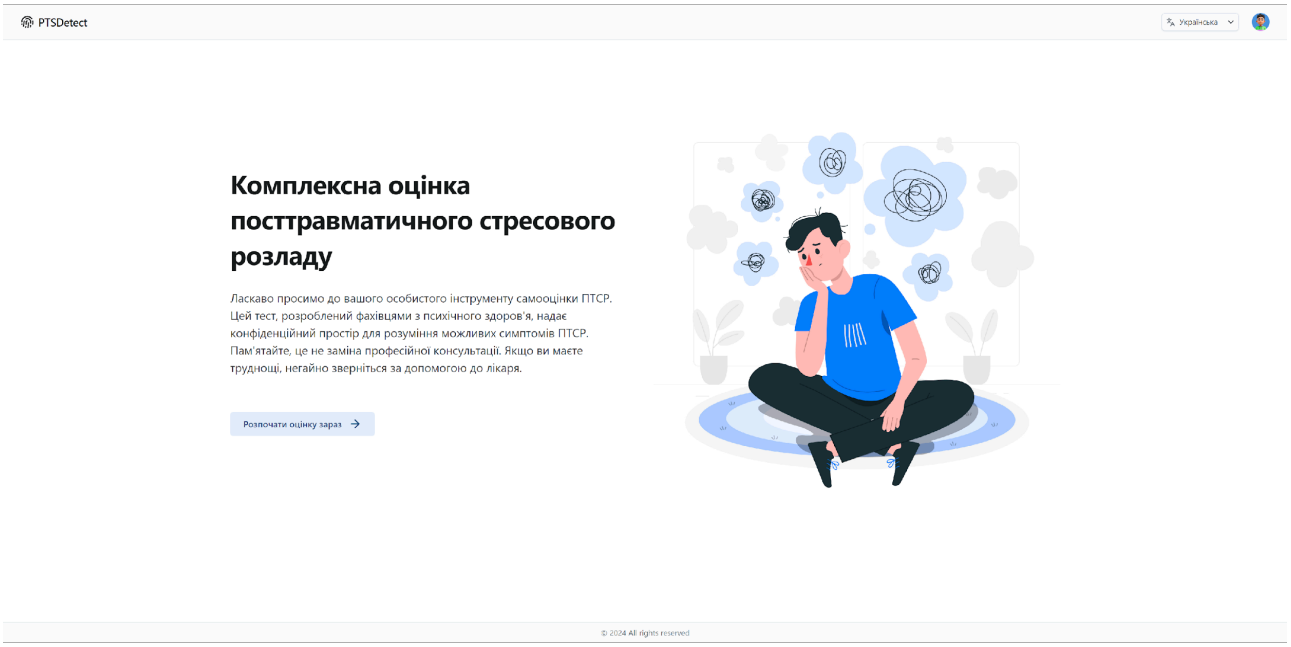


Рисунок 4.3. Дизайн головної сторінки

Дизайн головної сторінки веб-додатку є важливим елементом, що впливає на перше враження користувачів та їхню подальшу взаємодію з системою. Мінімалістичний та функціональний дизайн, що включає інформативний текст та чітку кнопку для початку тестування, сприяє залученню користувачів та забезпечує їхню зручність. Ретельно продуманий дизайн головної сторінки допомагає створити позитивний користувацький досвід, що є важливим для успішного функціонування додатку.

4.3.3 Сторінка загального тесту

Сторінка тестування є ключовим елементом веб-додатку. Вона повинна бути зручною та інтуїтивно зрозумілою, щоб користувачі могли легко проходити тестування без зайвих труднощів. У цьому розділі розглядається дизайн сторінки тестування, яка включає текст запитання, номер питання зі загальної кількості, варіанти відповідей та кнопку для переходу до наступного питання.

Сторінка тестування повинна бути структурованою та чіткою, щоб користувачі могли зосередитися на запитаннях та відповідях. Основні компоненти сторінки тестування включають:

1. Текст запитання: Текст запитання повинен бути розміщений у верхній частині сторінки та мати чіткий і зрозумілий шрифт. Він повинен бути достатньо великим, щоб його легко можна було прочитати. Важливо забезпечити, щоб текст запитання був добре видимим і не зливався з іншими елементами сторінки.
2. Номер питання: Номер питання зі загальної кількості повинен бути розміщений поруч із текстом запитання або у верхньому куті сторінки. Це допомагає користувачам орієнтуватися у процесі тестування та розуміти, скільки питань залишилося до завершення.
3. Варіанти відповідей: Варіанти відповідей повинні бути представлені у вигляді радіо-кнопок або чекбоксів, залежно від типу запитання (один або кілька варіантів відповіді). Важливо забезпечити достатній інтервал між варіантами відповідей, щоб уникнути випадкових натискань. Також варіанти відповідей повинні бути чітко виділені та легко доступні для вибору.
4. Кнопка "Далі": Кнопка для переходу до наступного питання повинна бути розташована у нижній частині сторінки. Вона повинна мати привабливий дизайн та чіткий заклик до дії, наприклад, "Далі" або "Наступне питання". Кнопка повинна бути інтерактивною, змінювати колір або анімацію при наведенні курсора, щоб привернути увагу користувача.

Дизайн сторінки тестування є важливим елементом, що впливає на зручність та ефективність процесу проходження тестування. Ретельно продуманий дизайн, що включає чіткий текст запитань, номер питання, варіанти відповідей та кнопку для переходу до наступного питання, забезпечує позитивний користувацький досвід. Це сприяє точності та надійності отриманих результатів, що є ключовим для оцінки психологічного стану користувачів. Дизайн сторінки тестування зображений на рис. 4.4.

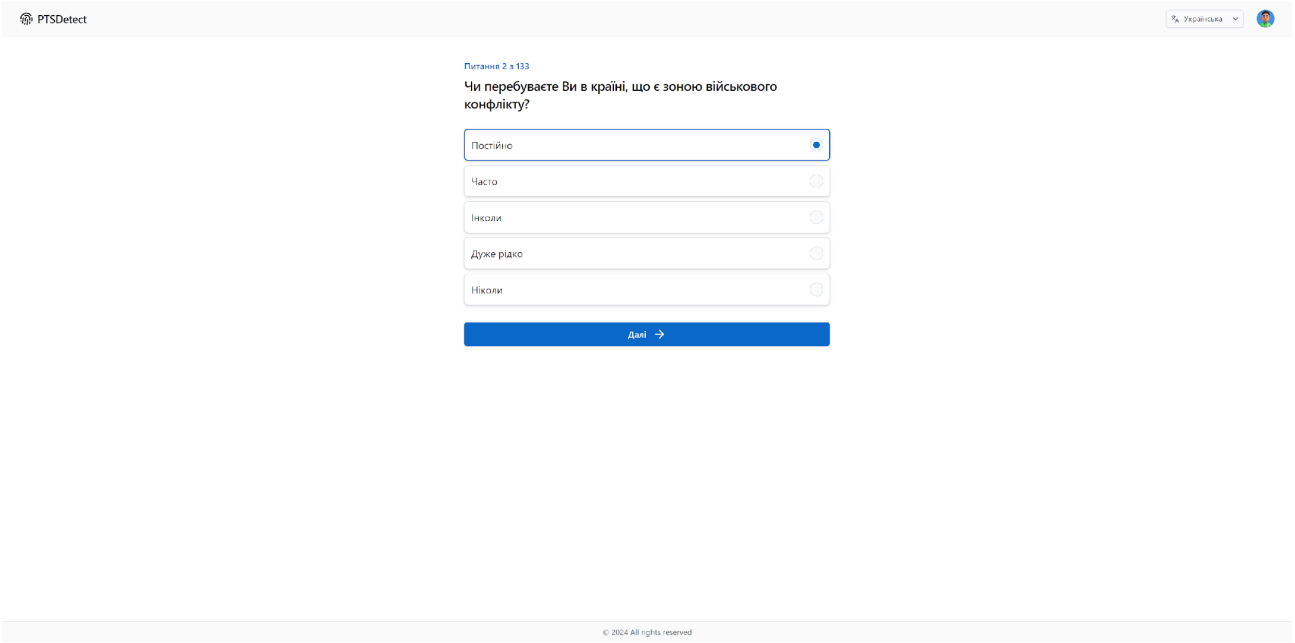


Рисунок 4.4. Дизайн сторінки тестування

4.3.4 Сторінка результату тесту

Сторінка результату тесту є кінцевою точкою взаємодії користувача з веб-додатком. Вона повинна надавати користувачам зрозумілу та структуровану інформацію про їхній психологічний стан, можливі проблеми та рекомендації щодо їх виправлення. У цьому розділі розглядається дизайн сторінки результату тесту, яка включає інформацію про можливі проблеми, такі як ПТСР (посттравматичний стресовий розлад) та інші, а також поради щодо їх виправлення.

Дизайн сторінки результату тесту у веб-додатку є важливим елементом, що впливає на розуміння користувачами своїх результатів та подальших дій. Ретельно продуманий дизайн, що включає інформацію про можливі проблеми та поради щодо їх виправлення, забезпечує користувачам зрозумілу та корисну інформацію. Це сприяє підвищенню ефективності додатку у наданні послуг з оцінки та поліпшення психологічного стану користувачів. Дизайн сторінки результату тесту зображений на рис. 4.5.

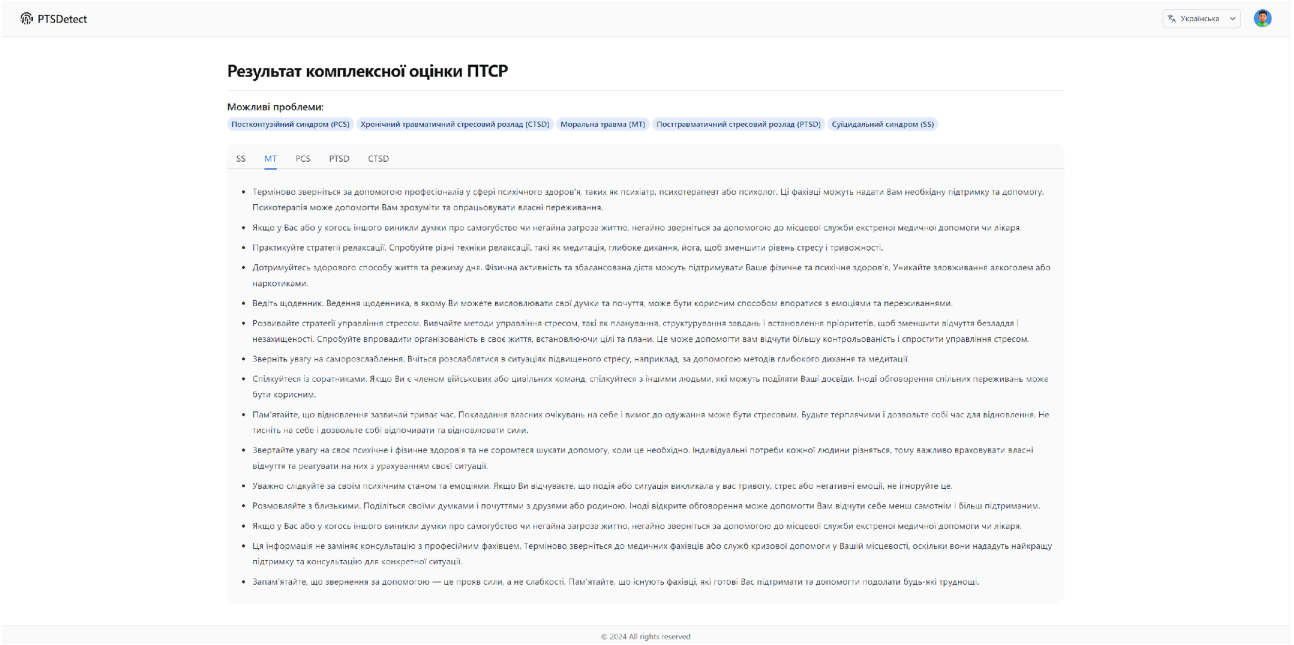


Рисунок 4.5. Дизайн сторінки результату тесту

4.3.5 Сторінка результатів пройдених тестів

Сторінка результатів пройдених тестів повинна дозволяти користувачам переглядати історію своїх тестувань. Вона повинна включати дати тестів та можливість переглянути результат. Дизайн сторінки результатів пройдених тестів зображений на рис. 4.6.

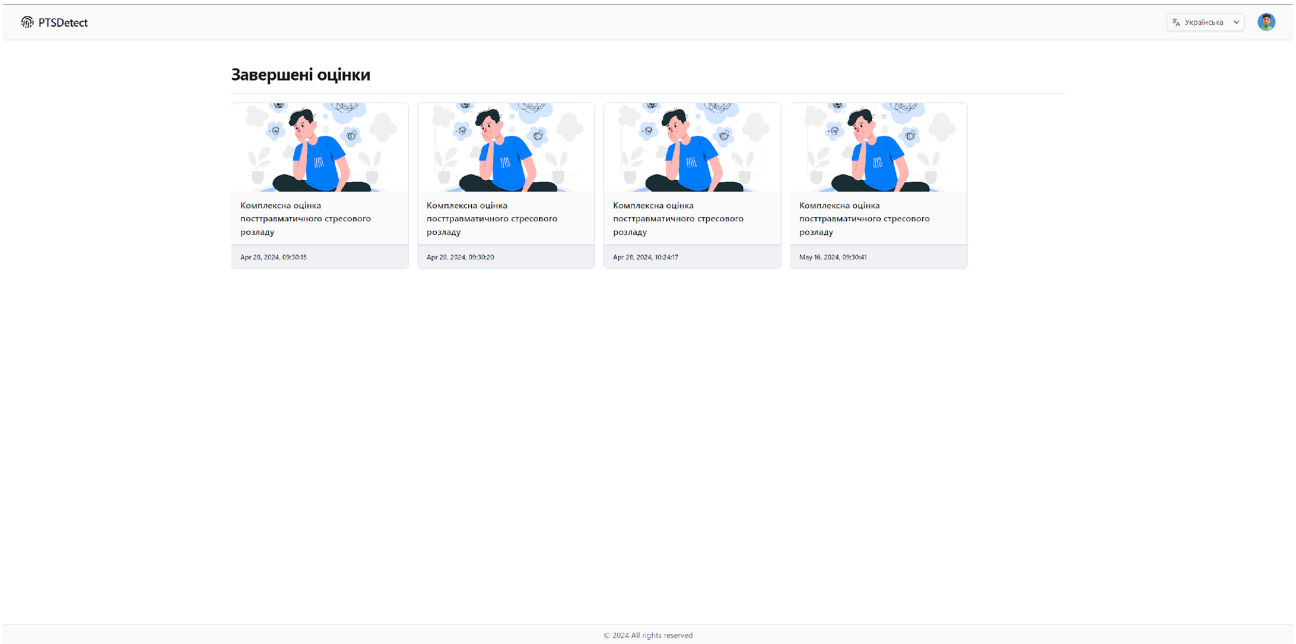


Рисунок 4.6. Дизайн сторінки результатів пройдених тестів

4.3.6 Профіль користувача

Сторінка профілю користувача надає користувачам можливість переглядати та редагувати свою особисту інформацію. Вона повинна бути зручною та інтуїтивно зрозумілою, щоб користувачі могли легко оновлювати свої дані. У цьому розділі розглядається дизайн сторінки профілю, яка включає форму для редагування аватару, ім'я та прізвища, статі, дня народження та сімейного стану.

Сторінка профілю повинна бути структурованою та зручною для користувача. Основні компоненти сторінки профілю включають:

1. Аватар. У верхній частині сторінки має бути розміщений аватар користувача. Поруч з аватаром повинна бути кнопка або посилання для його редагування. Користувачі повинні мати можливість завантажувати нові зображення або вибирати з наявних.
2. Форма для редагування особистих даних. Форма повинна включати наступні поля:
    - Ім'я та прізвище. Поля для введення імені та прізвища користувача. Важливо забезпечити, щоб ці поля були обов'язковими для заповнення;
    - Стать. Поле для вибору статі користувача. Це може бути випадаючий список або радіо-кнопки з варіантами "Чоловік", "Жінка" та "Інше";
    - День народження. Поле для вибору дати народження користувача. Це може бути текстове поле з календарем для зручного вибору дати;
    - Сімейний стан. Поле для вибору сімейного стану користувача. Це може бути випадаючий список з варіантами, такими як "Неодружений / незаміжня", "Одружений / заміжня", "Розлучений / розлучена" та інші.

3. Кнопка збереження змін. У нижній частині форми повинна бути розміщена кнопка для збереження змін. Вона повинна мати чіткий заклик до дії, наприклад, "Зберегти зміни". Кнопка повинна бути інтерактивною, змінювати колір або анімацію при наведенні курсора, щоб привернути увагу користувача.

Дизайн сторінки профілю користувача є важливим елементом, що впливає на зручність та ефективність взаємодії користувачів з додатком. Ретельно продуманий дизайн, що включає форму для редагування аватару, ім'я та прізвища, статі, дня народження та сімейного стану, забезпечує користувачам можливість легко оновлювати свої особисті дані. Дизайн сторінки профілю користувача зображений на рис. 4.7.

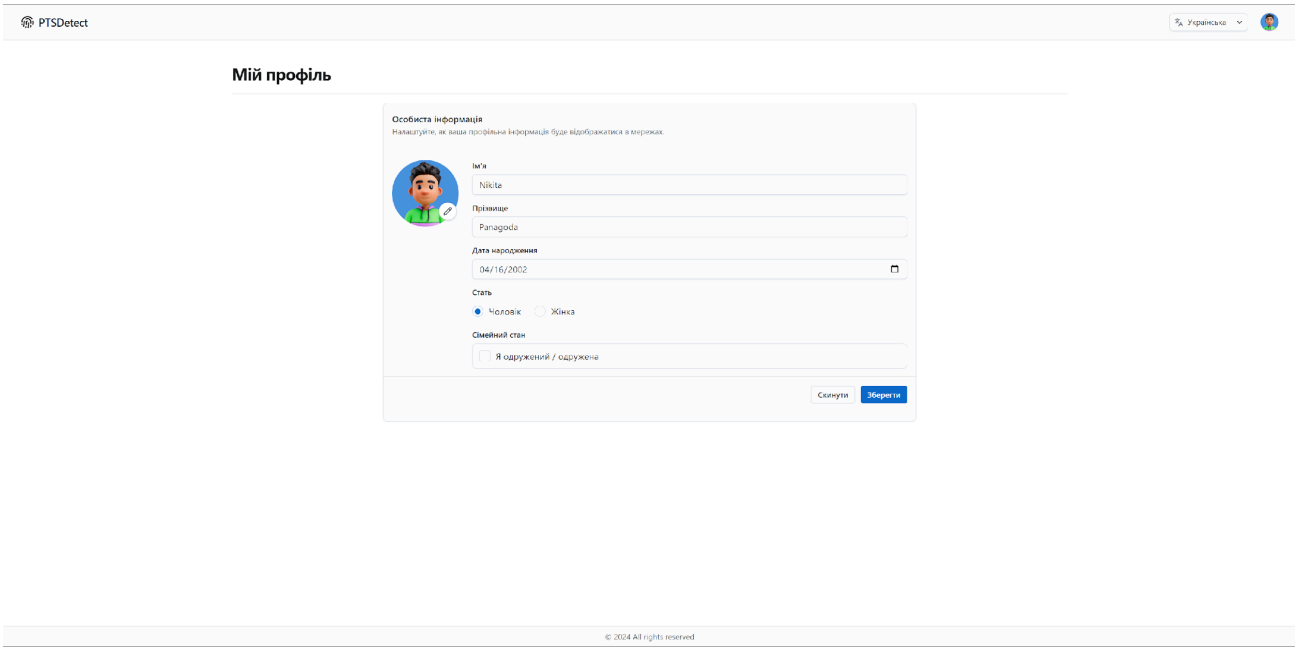


Рисунок 4.7. Дизайн сторінки профілю користувача

4.4 Адаптивний дизайн та доступність

У сучасному світі, де користувачі можуть взаємодіяти з веб-додатками на різних пристроях, від смартфонів до настільних комп'ютерів, адаптивний дизайн та доступність стають критично важливими аспектами розробки. Адаптивний дизайн забезпечує оптимальний користувацький досвід на різних екранах, тоді як доступність гарантує, що веб-додаток може бути використаний

людьми з різними можливостями [13]. У цьому розділі розглядаються принципи адаптивного дизайну та доступності, які були застосовані у нашому веб-додатку.

Адаптивний дизайн передбачає створення веб-сторінок, які автоматично підлаштовуються під розмір екрану та роздільну здатність пристрою. Основні принципи адаптивного дизайну, застосовані у веб-додатку, включають:

1. Гнучка сітка. Використання гнучкої сітки дозволяє елементам сторінки автоматично змінювати свої розміри та положення залежно від розміру екрану. Це забезпечує зручний перегляд та взаємодію на будь-якому пристрої.
2. Медіа-запити. Медіа-запити CSS використовуються для застосування різних стилів залежно від характеристик пристрою, таких як ширина екрану, орієнтація (портретна або ландшафтна) та роздільна здатність. Це дозволяє оптимізувати відображення контенту для різних пристроїв.
3. Гнучкі зображення та медіа. Зображення та відео автоматично підлаштовуються під розмір контейнера, що запобігає їхньому виходу за межі екрану та забезпечує коректне відображення на різних пристроях.

Приклад адаптивного дизайну сторінок зображений на рис. 4.8.

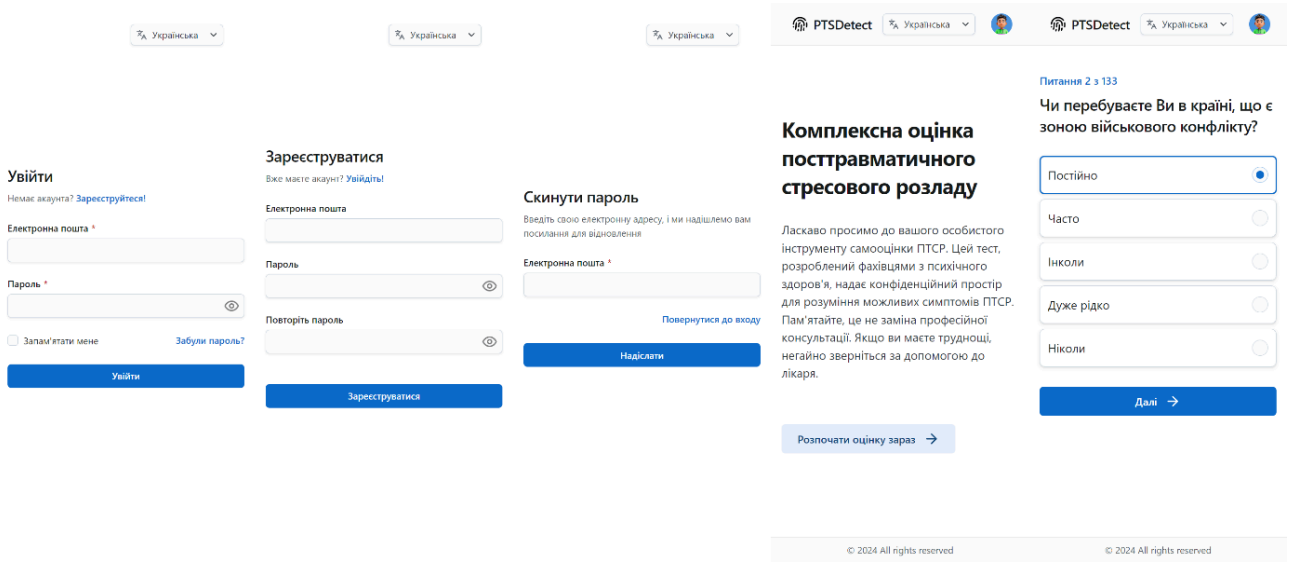


Рисунок 4.8. Приклад адаптивного дизайну сторінок

Доступність веб-додатків означає, що вони можуть бути використані людьми з різними можливостями, включаючи осіб з обмеженими можливостями. Основні принципи доступності, застосовані у веб-додатку, включають:

1. Альтернативний текст для зображень. Всі зображення мають альтернативний текст (alt-теги), що описує їхній зміст. Це забезпечує доступність для користувачів з вадами зору, які використовують екранні читалки;
2. Контрастність кольорів. Високий контраст між текстом та фоном забезпечує легке читання контенту для користувачів з вадами зору або кольоровою сліпотою;
3. Навігація з клавіатури. Веб-додаток підтримує повну навігацію з клавіатури, що дозволяє користувачам, які не можуть користуватися мишкою, легко взаємодіяти з додатком;
4. ARIA-атрибути. Використання ARIA (Accessible Rich Internet Applications) атрибутів допомагає екранним читалкам краще інтерпретувати елементи сторінки та забезпечує більш інклюзивний досвід;
5. Форми та поля вводу. Всі форми та поля вводу мають чіткі мітки та інструкції, що забезпечує легкість їх заповнення для всіх користувачів.

Адаптивний дизайн та доступність є ключовими аспектами розробки сучасних веб-додатків. Завдяки гнучкому дизайну та дотриманню принципів доступності, додаток забезпечує оптимальний користувацький досвід на різних пристроях та для користувачів з різними можливостями. Це сприяє підвищенню рівня задоволеності користувачів та робить додаток більш інклюзивним та зручним для всіх.

ВИСНОВОК

У даній дипломній роботі розглянуто процес розробки фронт-енд частини автоматизованої системи оцінки психологічного стану. Проведено глибокий аналіз існуючих методів оцінки психологічного стану, визначено основні вимоги до автоматизованої системи та виявлено ключові компоненти системи, такі як клієнтська частина, серверна частина та база даних.

Розроблено архітектуру системи з використанням сучасних технологій і підходів до розробки програмного забезпечення, зокрема, застосовано React.js для реалізації фронт-енд частини, що забезпечило високу продуктивність та зручність у розробці. Реалізовано основні функціональні можливості системи, включаючи реєстрацію та аутентифікацію користувачів, проведення тестувань та відображення результатів, а також впроваджено локалізацію для підтримки багатомовного інтерфейсу.

Створено зручний та інтуїтивно зрозумілий інтерфейс користувача, що забезпечує легкий доступ до всіх функцій системи, використано принципи адаптивного дизайну для забезпечення доступності на різних пристроях. Висновки з виконаної роботи свідчать про доцільність використання сучасних технологій для розробки автоматизованих систем, які можуть значно полегшити процес діагностики психологічного стану.

Система, створена в рамках даної роботи, має потенціал для подальшого розвитку та впровадження в реальних умовах, що може сприяти підвищенню ефективності раннього виявлення психічних захворювань.

Результати даної роботи можуть бути використані як основа для подальших досліджень та розробок в області автоматизованих систем оцінки психологічного стану, а також для розширення функціональності існуючої системи з метою покращення якості діагностики та надання психологічної допомоги.

Список використаних джерел

1. Package Managers Comparison: Yarn, NPM, PNPM, Tomas V., 2024 [Електронний ресурс] // Режим доступу: https://www.cookielab.io/blog/package-managers-comparison-yarn-npm-pnpm
2. 4 Reasons Why You Should Prefer Vite Over Create-React-App (CRA), Chetan G., 2023 [Електронний ресурс] // Режим доступу: https://semaphoreci.com/blog/vite
3. React Hook Form vs Formik: A Friendly Comparison, Louis Y., 2023 [Електронний ресурс] // Режим доступу: https://medium.com/@louis.young0420/react-hook-form-vs-formik-a-friendly-comparison-d2fc0650f1e3
4. Comparing React State Management Libraries: Redux, Zustand, Recoil, and MobX? Shraddha P., 2023 [Електронний ресурс] // Режим доступу: https://shraddha-paghdar.medium.com/comparing-react-state-management-libraries-redux-zustand-recoil-and-mobx-945402dc0cb
5. React Router VS Reach Router, Nathan S., 2020 [Електронний ресурс] // Режим доступу: https://blog.bitsrc.io/react-router-vs-reach-router-d26fe706d8db
6. Material UI vs Joy UI vs Base UI, Darsha K., 2023 [Електронний ресурс] // Режим доступу: https://medium.com/@darshakasrimal1234/material-ui-vs-joy-ui-vs-base-ui-729748b966b1
7. react-i18next vs react-intl [Електронний ресурс] // Режим доступу: https://i18nexus.com/posts/comparing-react-i18next-and-react-intl
8. Using Tailwind CSS with React.js: A Concise Guide, Kauna H., 2023 [Електронний ресурс] // Режим доступу: https://dev.to/haszankauna/using-tailwind-css-with-reactjs-a-concise-guide-33j
9. Setup eslint and prettier in React app, Ankit K., 2021 [Електронний ресурс] // Режим доступу: https://dev.to/knowankit/setup-eslint-and-prettier-in-react-app-357b

10. How to Fetch Data in React from a GraphQL API, Reed B., 2021 [Електронний ресурс] // Режим доступу: https://www.freecodecamp.org/news/5-ways-to-fetch-data-react-graphql/
11. How to Dockerize a React Application – A Step by Step Tutorial, Kunal N., 2023 [Електронний ресурс] // Режим доступу: https://www.freecodecamp.org/news/how-to-dockerize-a-react-application/
12. UI vs UX Design (an overview), Jesse S., 2018 [Електронний ресурс] // Режим доступу: https://medium.com/@iamjesseshow/ux-vs-us-7431dd859418
13. John F. The Design of Everyday Things: Revised and Expanded Edition, 1st Edition, 2013, pp. 110-145.

Додатки

Додаток А

Вихідний код компонента SignUp

SignUp.tsx

frontend > src > pages > Auth > SignUp > SignUp.tsx > ...

```tsx
      You, 6 months ago • Add sign in ui
export const SignUp = () => {
  const { t } = useTranslation();

  const [passwordInputType, setPasswordInputType] = useState('password');
  const [repeatPasswordInputType, setRepeatPasswordInputType] = useState('password');

  const [registerUser, { loading }] = useMutation(REGISTER_USER);
  const navigate = useNavigate();

  const {
    register,
    handleSubmit,
    setError,
    formState: { errors },
  } = useForm({
    resolver: yupResolver(formSchema),
    mode: 'onTouched',
    defaultValues: {
      email: '',
      password: '',
      repeatPassword: '',
    },
  });

  const onSubmit = handleSubmit(async (data) => {
    const formData = {
      email: data.email,
      password: data.password,
    };

    try {
      const { data: registerData } = await registerUser({ variables: formData });

      if (registerData?.registerUser.errors && registerData.registerUser.errors.length > 0) {
        registerData.registerUser.errors.forEach((error) => {
          switch (error.__typename) {
            case 'RegistrationFailedError':
              error.errors.forEach((error) => {
                switch (error.key) {
                  case 'DuplicateEmail':
                    setError('email', { message: t('validations.user-with-email-exists') });
                    break;
                }
              });
              break;
            default:
              console.error(`Unhandled error type: ${error.__typename}`);
          }
        });

        return;
      }

      navigate(routes.CONFIRM_EMAIL);
    } catch (error) {
      console.error('Sign up error:', error);
    }
  });

  return (
    <div className="flex flex-col w-[400px] gap-6">
      <div className="flex flex-col gap-2">
        <Typography level="h3">{t('sign-up.title')}</Typography>
```

{..} : 197 AWS: AWS Builder ID Amazon Q

Додаток Б

Вихідний код компонента SignIn

SignIn.tsx

frontend > src > pages > Auth > SignIn > SignIn.tsx > SignIn

```tsx
export const SignIn = () => {
  const { t } = useTranslation();

  const [passwordInputType, setPasswordInputType] = useState('password');
  const {
    register,
    handleSubmit,
    setError,
    formState: { errors },
  } = useForm({
    mode: 'onTouched',
    resolver: yupResolver(formSchema),
  });
  const [login, { loading }] = useMutation(LOGIN);

  const navigate = useNavigate();

  const onSubmit = handleSubmit(async (formData) => {
    try {
      const { data } = await login({
        variables: {
          input: {
            email: formData.email,
            password: formData.password,
          },
        },
      });

      if (data?.login.token?.value) {
        localStorage.setItem('token', data.login.token.value);
      }

      if (data?.login.errors && data.login.errors.length > 0) {
        data.login.errors.forEach((error) => {
          switch (error.__typename) {
            case 'EmailIsNotVerifiedError':
              setError('email', { message: t('validations.email-not-verified') });
              break;
            case 'InvalidCredentialsError':
              setError('password', { message: t('validations.wrong-password') });
              break;
            default:
              console.error(`Unhandled error type: ${error.__typename}`);
          }
        });

        return;
      }

      navigate(routes.HOME);
    } catch (error) {
      console.error('Login error:', error);
    }
  });

  return (
    <div className="flex flex-col w-[400px] gap-6">
      <div className="flex flex-col gap-2">
        <Typography level="h3">{t('sign-in.title')}</Typography>
        <Typography level="body-sm">
          {t('sign-in.sub-title-part-1')}{' '}      nkt.pngd, 2 weeks ago • Add internalization (#76) …
          <NavLink to={routes.SIGN_UP}>
            <Link level="title-sm">{t('sign-in.sub-title-part-2')}</Link>
```

{..} : 197 AWS: AWS Builder ID Amazon Q

Додаток В

Вихідний код компонента ForgotPassword

ForgotPassword.tsx

frontend > src > pages > Auth > ForgotPassword > ForgotPassword.tsx > ForgotPassword

```tsx
export const ForgotPassword = () => {
  const { t } = useTranslation();

  const {
    register,
    handleSubmit,
    setError,
    formState: { errors },
  } = useForm({
    resolver: yupResolver(formSchema),
  });

  const [requestPasswordReset, { loading }] = useMutation(REQUEST_PASSWORD_RESET);
  const [isEmailSent, setIsEmailSent] = useState(false);

  const onSubmit = handleSubmit(async (formData) => {
    try {
      const { data } = await requestPasswordReset({
        variables: {
          email: formData.email,
        },
      });

      if (data?.requestPasswordReset.errors && data.requestPasswordReset.errors.length > 0) {
        data.requestPasswordReset.errors.forEach((error) => {
          switch (error.__typename) {
            case 'UserNotFoundError':
              setError('email', { message: t('validations.user-with-email-not-found') });
              break;
            default:
              console.error(`Unhandled error type: ${error.__typename}`);
          }
        });

        return;
      }

      setIsEmailSent(true);
    } catch (error) {
      console.error('Request password reset error:', error);
    }
  });

  if (isEmailSent) {
    return (
      <div className="flex flex-col w-[400px] gap-6">
        <div className="flex flex-col gap-2 items-center">
          <Mail size={92} color="#185EA5" />
          <Typography level="h3" textAlign="center">
            {t('forgot-password.check-your-email')}
          </Typography>
          <Typography level="body-sm" textAlign="center">
            {t('forgot-password.we-sent-recovery-link')}
          </Typography>
        </div>
      </div>
    );
  }

  return (
    <div className="flex flex-col w-[400px] gap-6">
      <div className="flex flex-col gap-2">
        <Typography level="h3">{t('forgot-password.title')}</Typography>
        <Typography level="body-sm">{t('forgot-password.sub-title')}</Typography>
```

{..} : 197 AWS: AWS Builder ID Amazon Q

Додаток Г

Вихідний код компонента ResetPassword

ResetPassword.tsx

frontend > src > pages > Auth > ResetPassword > ResetPassword.tsx > formSchema > password

```tsx
export const ResetPassword = () => {
  const { t } = useTranslation();

  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm({
    mode: 'onTouched',
    resolver: yupResolver(formSchema),
  });

  const [isVerified, setIsVerified] = useState(false);
  const [isReseted, setIsReseted] = useState(false);
  const [passwordInputType, setPasswordInputType] = useState('password');
  const [repeatPasswordInputType, setRepeatPasswordInputType] = useState('password');

  const [resetPassword, { loading }] = useMutation(RESET_PASSWORD);
  const [verifyResetPasswordToken, { loading: isVerifyLoading }] = useLazyQuery(VERIFY_RESET_PASSWORD_TOKEN);
  const [searchParams] = useSearchParams();
  const navigate = useNavigate();

  useEffect(() => {
    const userId = searchParams.get('userId');
    const token = searchParams.get('token');

    const verify = async () => {
      try {
        if (userId && token) {
          const { data } = await verifyResetPasswordToken({
            variables: {
              input: {
                userId,
                token,
              },
            },
          });

          setIsVerified(!!data?.verifyResetPasswordToken.isVerified);
        }
      } catch (error) {
        console.error('Verify error:', error);
      }
    };

    verify();
  }, [searchParams]);

  const onSubmit = handleSubmit(async (formData) => {
    try {
      const { data } = await resetPassword({
        variables: {
          userId: searchParams.get('userId') ?? '',
          token: searchParams.get('token') ?? '',
          newPassword: formData.password,
        },
      });

      if (data?.resetPassword.errors && data.resetPassword.errors.length > 0) {
        data.resetPassword.errors.forEach((error) => {
          switch (error.__typename) {
            case 'UserNotFoundError':
              console.log('User not found');
              break;
```

{..} : 197 AWS: AWS Builder ID Amazon Q

Додаток Д

Вихідний код компонента GeneralTest

GeneralTest.tsx

frontend > src > pages > GeneralTest > GeneralTest.tsx > GeneralTest

```tsx
export const GeneralTest = () => {
  const { t, i18n } = useTranslation();

  const [questions, setQuestions] = useState<Question[] | undefined>([]);
  const [endCursor, setEndCursor] = useState<string | undefined>(undefined);
  const [hasNextPage, setHasNextPage] = useState<boolean | undefined>(undefined);
  const [currentQuestion, setCurrentQuestion] = useState(0);
  const [totalQuestions, setTotalQuestions] = useState(0);

  const { register, handleSubmit, watch, reset } = useForm({});
  const watchedValues = watch();

  const navigate = useNavigate();

  const [getQuestions, { loading: isQuestionsLoading, data: questionsData }] = useLazyQuery(
    GET_GENERAL_TEST_QUESTIONS,
    {
      fetchPolicy: 'no-cache',
    }
  );
  const [submitAnswers, { loading: isSubmitLoading }] = useMutation(SUBMIT_GENERAL_TEST_ANSWERS);

  useEffect(() => {
    setCurrentQuestion(0);
    setQuestions([]);
    setEndCursor(undefined);
    setHasNextPage(undefined);
    reset();

    getQuestions({
      variables: {
        input: {
          languageCode: i18n.language,
        },
        first: 1,
      },
    });
  }, [i18n.language]);
       You, 4 months ago • Add home page
  useEffect(() => {
    if (questionsData) {
      const newQuestions = questionsData.generalTestQuestions.questions?.nodes || [];
      const currentQuestions = questions || [];

      setQuestions([...currentQuestions, ...newQuestions]);
      setEndCursor(questionsData.generalTestQuestions.questions?.pageInfo.endCursor ?? '');
      setHasNextPage(questionsData.generalTestQuestions.questions?.pageInfo.hasNextPage ?? false);
      setTotalQuestions(questionsData.generalTestQuestions.questions?.totalCount || 0);

      if (questions && questions?.length >= 1) {
        setCurrentQuestion((currState) => currState + 1);
      }
    }
  }, [questionsData]);

  const loadNextQuestion = () => {
    getQuestions({
      variables: {
        input: {
          languageCode: i18n.language,
        },
        after: endCursor,
        first: 1,
      },
```

{..} : 197 AWS: AWS Builder ID Amazon Q

Додаток Е

Вихідний код компонента GeneralTestResult

GeneralTestResult.tsx

frontend > src > pages > GeneralTest > GeneralTestResult.tsx > GeneralTestResult

```tsx
export const GeneralTestResult = () => {
  const { t, i18n } = useTranslation();

  const [result, setResult] = useState<null | undefined | Result>(null);
  const [index, setIndex] = useState(0);

  const { id } = useParams();
  const [getResult, { loading: isResultLoading }] = useLazyQuery(GET_GENERAL_TEST_RESULT, {
    fetchPolicy: 'no-cache',
  });

  useEffect(() => {
    getResult({
      variables: {
        input: {
          languageCode: i18n.language,
          resultId: id,
        },
      },
    }).then((result) => {
      setResult(result.data?.generalTestResult?.result);
    });
  }, [id, i18n.language]);

  if (isResultLoading) {
    return (
      <div className="flex flex-col items-center mx-auto gap-2">
        <CircularProgress />
      </div>
    );
  }

  if (result?.potentialProblems.length === 0) {
    return (
      <div className="flex flex-col mx-auto gap-4">
        <Typography level="h2">{t('general-test-result.title')}</Typography>

        <Divider />

        <Typography level="title-lg">{t('general-test-result.no-problems-found')}</Typography>
      </div>
    );
  }

  return (
    <>
      {result && (
        <div className="flex flex-col mx-auto gap-4">
          <Typography level="h2">{t('general-test-result.title')}</Typography>

          <Divider />       nkt.pngd, 2 weeks ago • Bug fixes (#79) …

          <Typography level="title-lg">
            {t('general-test-result.sub-title')}
            <div className="flex flex-wrap gap-1 mt-2">
              {result.potentialProblems.map((problem: any) => (
                <Chip key={problem} variant="soft" color="primary">
                  {t(`problems.${problem}`)} ({problem})
                </Chip>
              ))}
            </div>
          </Typography>

          <Box
```

{..} : 197 AWS: AWS Builder ID Amazon Q

Додаток Є

Вихідний код компонента Profile

Profile.tsx

frontend > src > pages > Profile > Profile.tsx > Profile

```tsx

export const Profile = () => {
  const { t } = useTranslation();
  const { user, setUserInfo, setAvatarUrl, avatarUrl } = useStore((state) => state);

  const [avatarPreview, setAvatarPreview] = useState<any>(null);
  const [avatarFile, setAvatarFile] = useState<any>(null);
  const inputFile = useRef<HTMLInputElement>(null);

  const [updateUserInfo, { loading }] = useMutation(UPDATE_USER_INFO);
  const [updateUserAvatar] = useMutation(UPDATE_USER_AVATAR);
  const [getUploadAvatarUrl] = useLazyQuery(GET_UPLOAD_AVATAR_URL);

  const {
    register,
    handleSubmit,
    formState: { errors },
    setValue,
    watch,
  } = useForm({
    resolver: yupResolver(formSchema),
  });

  const setDefaultValues = () => {
    if (user) {
      setValue('firstName', user.personalInfo?.firstName || '');
      setValue('lastName', user.personalInfo?.lastName || '');
      setValue('birthdate', user.personalInfo?.birthdate || '');
      setValue('sex', user.personalInfo?.sex || Sex.Male);
      setValue('isMarried', user.personalInfo?.isMarried || false);
    }
  };

  useEffect(() => {
    setDefaultValues();
  }, [user, setValue]);

  const handleAvatarChange = (event: any) => {
    const file = event.target.files[0];
    const reader = new FileReader();

    reader.onloadend = () => {
      setAvatarPreview(reader.result);
      setAvatarFile(file);
    };

    if (file) {
      reader.readAsDataURL(file);
    }
  };

  const handleAvatarUpload = async () => {
    if (avatarFile) {
      const { data } = await getUploadAvatarUrl();

      if (data) {
        const { uploadUrl, avatarId } = data.uploadAvatarUrl;

        await fetch(uploadUrl, {
          method: 'PUT',
          body: avatarFile,
          headers: {
            'x-ms-blob-type': 'BlockBlob',
          },
```

{..} : 197 AWS: AWS Builder ID Amazon Q

Додаток Ж

Вихідний код компонента LanguageSelect

LanguageSelect.tsx

frontend > src > components > LanguageSelect > LanguageSelect.tsx > ...

nkt.pngd, 2 weeks ago | 1 author (nkt.pngd)

```tsx
import { Select, selectClasses, Option } from '@mui/joy';  105.3k (gzipped: 33.9k)
import { ChevronDown, Languages } from 'lucide-react';  1.3k (gzipped: 789)
import { useTranslation } from 'react-i18next';

export const LanguageSelect = () => {
  const { i18n } = useTranslation();

  const onLanguageChange = (_event: any, value: any) => {
    i18n.changeLanguage(value);
    localStorage.setItem('lang', value);
  };

  return (
    <Select
      onChange={onLanguageChange}
      disabled={false}
      defaultValue={i18n.language}
      size="sm"
      startDecorator={<Languages size={16} />}
      indicator={<ChevronDown size={16} />}
      sx={{
        width: 140,
        [`& .${selectClasses.indicator}`]: {
          transition: '0.2s',
          [`&.${selectClasses.expanded}`]: {
            transform: 'rotate(-180deg)',
          },
        },
      }}
    >
      <Option value="en">English</Option>
      <Option value="uk">Українська</Option>
      <Option value="ru">Русский</Option>
    </Select>
  );
};

```