Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Програмне забезпечення для ведення переліків бажань

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-4
Богдан РОГОЗЯНСЬКИЙ
(власне ім’я, прізвище)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: ас. Олег ЖУРИЛО
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Рогозянському Богдану Юрійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Програмне забезпечення для ведення переліків бажань

затверджена наказом по університету від “ 26 ” травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи

1) документація фреймворку Next.js;
2) документація TypeScript;
3) документація PostgreSQL;
4) документація Tailwind CSS;
5) документація React;
6) редактор програмного коду Visual Studio Code.

4. Перелік питань, що потрібно опрацювати у роботі

1) аналіз предметної області;
2) аналіз використовуваних технологій;
3) програмна реалізація;
4) інструкція користувача;
5) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд презентація – 11 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	27.05.25-30.05.25	
2	Вибір технологій розробки	31.05.25-02.06.25	
3	Розробка алгоритмічного забезпечення	03.06.25-05.06.25	
4	Розробка та відлагодження програмного	06.06.25-09.06.25	
	забезпечення		
5	Оформлення матеріалів кваліфікаційної роботи	10.06.25-11.06.25	
6	Подання кваліфікаційної роботи керівникові та	12.06.25-13.06.25	
	її попередній захист		
7	Подання кваліфікаційної роботи на рецензу-	14.06.25-16.06.25	
	вання		

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ас. Олег ЖУРИЛО
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 116 с., 27 рис., 1 табл., 2 дод., 30 джерел.

ВЕБЗАСТОСУНОК, СПИСКИ БАЖАНЬ, TYPESCRIPT, SSR, NEXT.JS, SUPABASE, PRISMA, POSTGRESQL, TAILWIND, JWT, BCRYPTJS, DOCKER, VERCEL.

Метою роботи є створення сучасного вебзастосунку для ведення переліків бажань. Застосунок дозволяє користувачам створювати власні списки бажань, управляти ними, взаємодіяти з іншими користувачами і ділитися списками з урахуванням рівня доступу.

У ході реалізації застосунку використано підхід з розділенням архітектури на фронтенд та бекенд з базою даних. Використано SSR (server-side rendering) у поєднанні з клієнтським рендером. Основними інструментами розробки стали Next.js, TypeScript, Tailwind CSS, Supabase та Prisma з PostgreSQL.

У застосунку реалізовано функціональність створення та редагування бажань, включно з додаванням зображень. Передбачена можливість встановлення пріоритету, ціни, бажаної дати отримання та статусу виконання бажання. Списки можуть бути приватними або публічними, з гнучким контролем доступу.

У результаті реалізовано функціональний, масштабований і зручний у користуванні вебзастосунок, який має потенціал розвитку завдяки використаній архітектурі.

ABSTRACT

Bachelor’s thesis: 116 pages, 27 figures, 1 tables, 2 appendices, 30 sources.

WEB APPLICATION, WISH LISTS, TYPESCRIPT, SSR, NEXT.JS, SUPABASE, PRISMA, POSTGRESQL, TAILWIND, JWT, BCRYPTJS, DOCKER, VERCEL.

This thesis aims to develop a modern web application for managing personal wish lists. The application allows users to create and manage their wish lists, interact with other users, and share them with customized access levels.

The development process involved architecture that separates the front and back end supported by a database. A hybrid rendering approach was employed, combining server-side rendering (SSR) with client-side rendering. The primary development tools were Next.js, TypeScript, Tailwind CSS, Supabase, and Prisma with PostgreSQL.

The application includes features such as functionality for creating and editing wishes and the ability to upload images. Users can set priorities, estimated prices, desired receive dates, and completion statuses for each wish. Lists can be private or public, with flexible access control.

As a result, a functional, scalable, and user-friendly web application has been developed, with significant potential for future growth thanks to the chosen architecture.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

СУБД – система управління базами даних

API – Application Programming Interface

CI/CD – Continuous Integration / Continuous Delivery

CLI – Command Line Interface

CRM – Customer Relationship Management

CSV – Comma-Separated Values

CSS – Cascading Style Sheets

CTE – Common Table Expression

ER – Entity-Relationship

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

JSON – JavaScript Object Notation

JWT – JSON Web Token

OAuth – Open Authorization

ORM – Object-Relational Mapping

SEO – Search Engine Optimization

SQL – Structured Query Language

SSG – Static Site Generation

SSR – Server-Side Rendering

UI/UX – User Interface / User Experience

URL – Uniform Resource Locator

YAML – Yet Another Markup Language

ВСТУП

Вміння ефективно організовувати свої цілі, бажання та плани в сучасному темпі життя, який постійно прискорюється і тільки збільшує кількість інформації, з якою людство щодня стикається, є важливим компонентом успішного управління часом та ресурсами. Особливого значення в цьому контексті набувають персональні списки бажань, які відображають мрії, потреби та прагнення. Однак, часто ці списки розподілені між різними застосунками, нотатками на папері, або навіть просто зберігаються в пам’яті, що значно ускладнює їхнє відстеження та систематизацію. Такі методи зберігання перешкоджають процесу планування особистих покупок, а також вибору подарунків для друзів та рідних, створюють незручності та призводять до зайвих витрат часу.

У зв’язку з цим, створення програмного забезпечення для організації переліків бажань є актуальною та важливою задачею, яка має на меті запровадити користувачам зручний, централізований та інтуїтивно зрозумілий інструмент. Він допоможе систематизувати ведення та створювати списки бажань, безпечно зберігаючи їх на віддаленому сервері, що запобігає можливим ризикам втрати локальних даних.

Даний проєкт присвячений розробці програмного забезпечення з використанням сучасних технологій, що дозволить користувачам легко створювати, редагувати та керувати своїми списками та бажаннями, а в розробці дозволить реалізувати надійні архітектурні рішення та забезпечить безпеку доступу до даних. Головна мета проєкту – запровадити зручний інструмент, який допоможе ефективніше організовувати свої плани, здійснювати покупки та вибирати подарунки. У процесі розробки особливу увагу приділено вибору технологій та архітектурних рішень, що забезпечують стабільність, швидкодію та масштабованість програмного забезпечення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис задачі

Зазвичай люди використовують різні методи збереження цифрових чи письмових даних, такі як нотатки, месенджери, записники, папки з листами формату А4 чи сторонні комп’ютерні програми з певною спеціалізацією. Кожні із них мають свої переваги: одні зручні для довготривалого зберігання, інші для обміну чи забезпечення конфіденційності.

Однак, наявні методи часто не повністю задовольняють потреби користувачів у контексті ведення переліків бажань. Традиційні записники можуть бути загублені або пошкоджені, електронні нотатки в смартфонах, як і месенджери, не завжди зручні для структурування інформації. Крім того, більшість існуючих рішень не мають спеціалізованого функціоналу саме для управління бажаннями, що робить процес їх записування довшим [1], бо необхідно витрачати час на ручне прописування всіх потрібних даних і шаблонів.

Зазначені недоліки неспеціалізованих рішень створюють низку проблем для користувачів, такі як відсутність категоризації, що знижує ефективність пошуку конкретних бажань, а також неможливість встановлення пріоритетів, що ускладнює планування. Брак функцій для відстеження прогресу [2] не дозволяє користувачу бачити шлях до досягнення власних цілей.

Суттєвою проблемою існуючих рішень є також рівень безпеки та контролю доступу до особистої інформації. Записники можуть бути прочитані сторонніми особами, так як і записи в комп’ютерних програмах, а повідомлення в деяких месенджерах можна випадково відправити не тому адресату [3]. Користувачі прагнуть мати повний контроль над своїми даними та самостійно визначати, хто й до якої інформації має доступ.

Повсякденне життя показує, що сучасна людина часто розподіляє свої дані між різними платформами та засобами: список книг для читання зберігається в одному застосунку, бажані подарунки на день народження – в нотатках телефону, плани подорожей – у таблицях, а професійні цілі – в спеціалізованому планері [4]. Ця фрагментація призводить до суттєвих незручностей та неефективності в управлінні власними бажаннями. Особливо актуальною ця проблема є у випадках, коли необхідна інформація зберігається на недоступному в моменті пристрої чи в окремому застосунку.

Основною задачею, яку вирішує проєкт, є неорганізоване зберігання переліків бажань користувачів. Наявність кількох платформ ускладнює пошук потрібної інформації, призводить до плутанини, дублювання інформації, створює ризик втрати важливих даних і незручностей при координації з іншими людьми, особливо під час планування подарунків.

Відсутність єдиної системи, яка б дозволяла зручно структурувати бажання, встановлювати пріоритети, відстежувати прогрес та таким чином оптимізувати процес їх реалізації, призводить до того, що багато людей можуть відчувати труднощі з плануванням та досягненням своїх цілей. Це може створювати відчуття розсіяності, призводити до втрати мотивації та зниження якості життя загалом.

1.2 Аналіз існуючих рішень

Перед початком роботи, необхідно здійснити об’єктивний аналіз існуючих рішень у сфері цифрової організації переліків бажань. Таким чином можна буде врахувати поточний стан ринку, визначити переваги та недоліки популярних застосунків, а також зрозуміти потреби цільової аудиторії.

Сучасний ринок програмних засобів пропонує чимало мобільних і вебрішень для створення та обміну списками бажань. Проте більшість із них має суттєві вади, які знижують комфорт їхнього використання, наприклад, такі як орієнтованість тільки на одну платформу. Якщо користувач знайде

зручний застосунок для свого смартфона, немає гарантії, що розробники передбачили можливість використання цього ж функціоналу на персональному комп’ютері.

Ще одним суттєвим бар’єром є комерційна модель більшості таких застосунків: велика частина сервісів є частково або повністю платними, що обмежує доступ до всього функціоналу, а безкоштовні версії часто мають нав’язливу рекламу, а також схильні до перенасичення другорядними або нефункціональними можливостями, які не тільки не приносять користі, але й погіршують загальний користувацький досвід.

1.2.1 Мобільний застосунок «Listy»

Одним із поширених представників сучасних мобільних рішень для збереження персональних списків є застосунок «Listy» [5]. Його функціональні можливості засновані на концепції поділу інформації на категорії, зокрема на такі як «to-do», «books», «apps» або «songs» (рисунок 1.1), що на перший погляд здається універсальним і зручним.

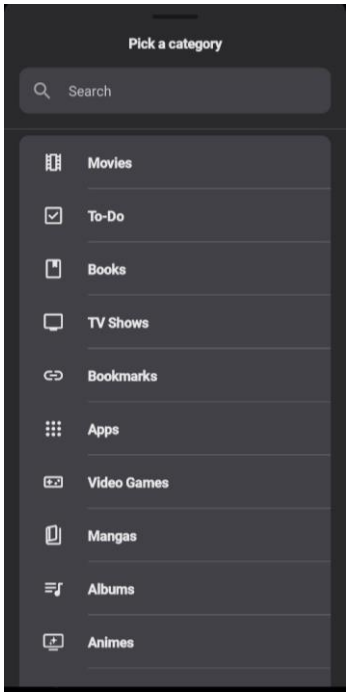

Рисунок 1.1 – Вибір категорій в застосунку «Listy»

Проте відсутність спеціалізації застосунка на формуванні саме переліків бажань обмежує його придатність у цьому конкретному сценарії використання. Застосунок не дозволяє пов’язувати бажання з конкретними особами чи адресатами подарунків.

Крім того, слід зазначити, що «Listy» існує виключно у форматі мобільного застосунку, без можливості доступу до даних або редагування списків через персональний комп’ютер. Це суттєво знижує зручність користування в умовах, коли користувач віддає перевагу багатоплатформеному середовищу або працює з великими обсягами даних, що передбачають введення з клавіатури, а не з екрана смартфона.

Іншою функціональною особливістю, яка може створювати труднощі, є принцип автоматичного завантаження інформації з відкритих джерел, таких як Goodreads – популярна онлайн-платформа, розташована в Сан-Франциско [6]. Наприклад, при додаванні книжок система звертається до серверів цієї платформи і отримає відповідні метадані (рисунок 1.2). Проте система не завжди може забезпечити наявність саме тієї книжки, яку бажає додати користувач.

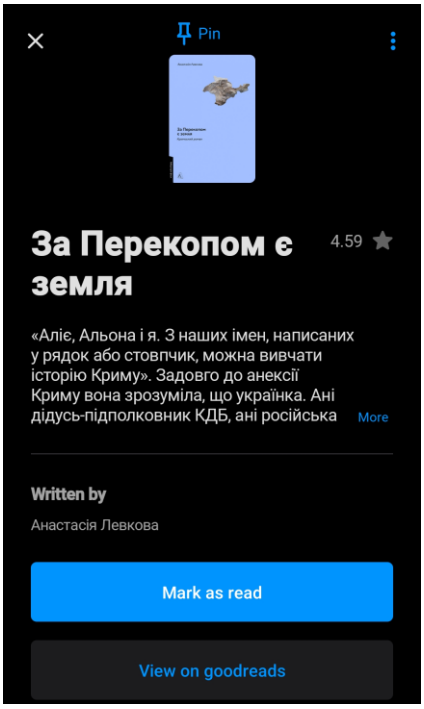

Рисунок 1.2 – Запис книги з підвантаженими даними

Варіант ручного додавання запису зі своєю назвою, не існуючою в Goodreads, частково вирішує проблему, але породжує іншу. Без платної підписки користувач не має можливості редагувати або доповнювати запис додатковими елементами – зображенням обкладинки, описом, примітками тощо. Таким чином, можуть існувати записи з однією лише назвою (рисунок 1.3). Навіть базова кастомізація списків стає недоступною, що суперечить головній меті подібного інструменту – створення персоналізованого простору для роботи з власними побажаннями.

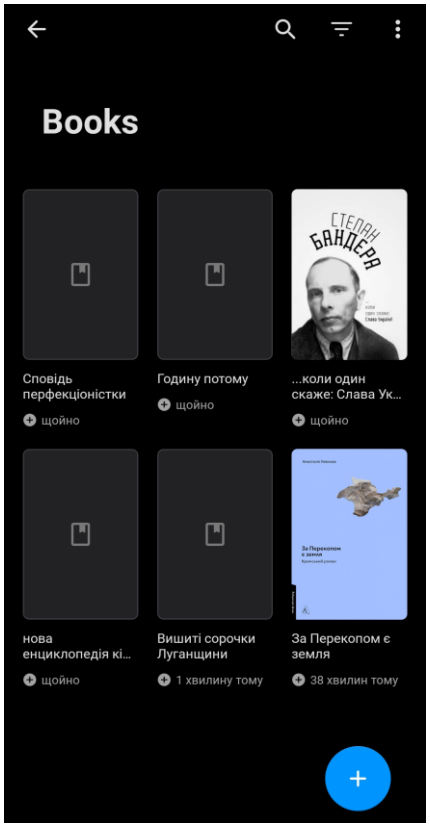

Рисунок 1.3 – Додані книжки

1.2.2 Мобільний застосунок «Wisher»

Основною функцією популярного мобільного рішення «Wisher» [7], в контексті організації переліків бажань, є збереження побажань користувачів (рисунок 1.4). Проте, відсутність зв’язку між записом і самим переліком погано впливає на організацію інформації.

Переліком у застосунку «Wisher» називаються «Колекції», до яких не обов’язково додавати бажання, що зробить пошук інформації важчим.

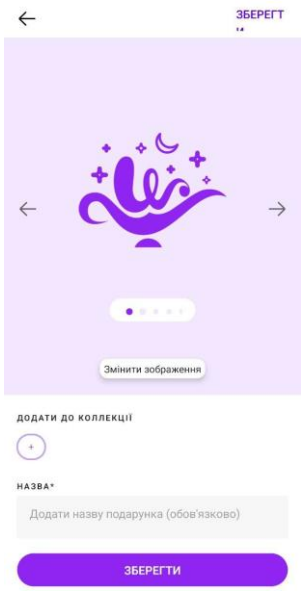

Рисунок 1.4 – Сторінка створення бажання в застосунку «Wisher»

Перше, що бачить людина, коли заходить в застосунок – це список улюблених магазинів, величезне вікно купонів і пропозицій, а також ідеї інших користувачів (рисунок 1.5). Таким чином, необхідно виконувати додаткові дії для перегляду власних бажань, що значно погіршує користувацький досвід особи.

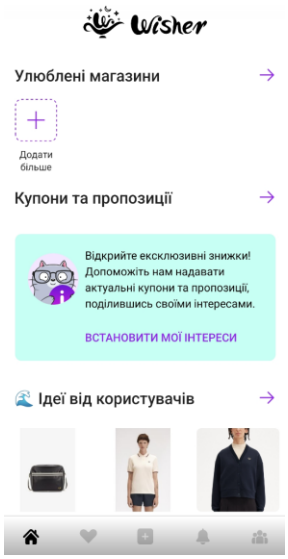

Рисунок 1.5 – Головна сторінка застосунку «Wisher»

Окрему увагу заслуговує додатковий функціонал, а саме можливість перегляду ідей від інших користувачів або збереження улюблених магазинів. Можливості покликані урізноманітнити користувацький досвід, але на практиці вони часто будуть відволікати від основної мети використання застосунку – створення персонального простору для збереження бажань.

У системі не передбачено чіткої ієрархії чи класифікації, що знижує ефективність роботи користувача з власними записами, особливо при повторному поверненні до застосунку через певний проміжок часу.

Можливо в майбутньому розробники створять можливість поєднувати бажання з переліками, бо під час створення запису там наявна спеціальна кнопка, але наразі вона не має ніякого впливу – створити список неможливо.

1.2.3 Платформа «Rewish»

«Rewish» відрізняється серед інших тим, що має як мобільну версію, так вебінтерфейс (рисунок 1.6). Ця мультиплатформність є безперечною перевагою, бо дозволяє взаємодіяти з системою з будь-якого пристрою.

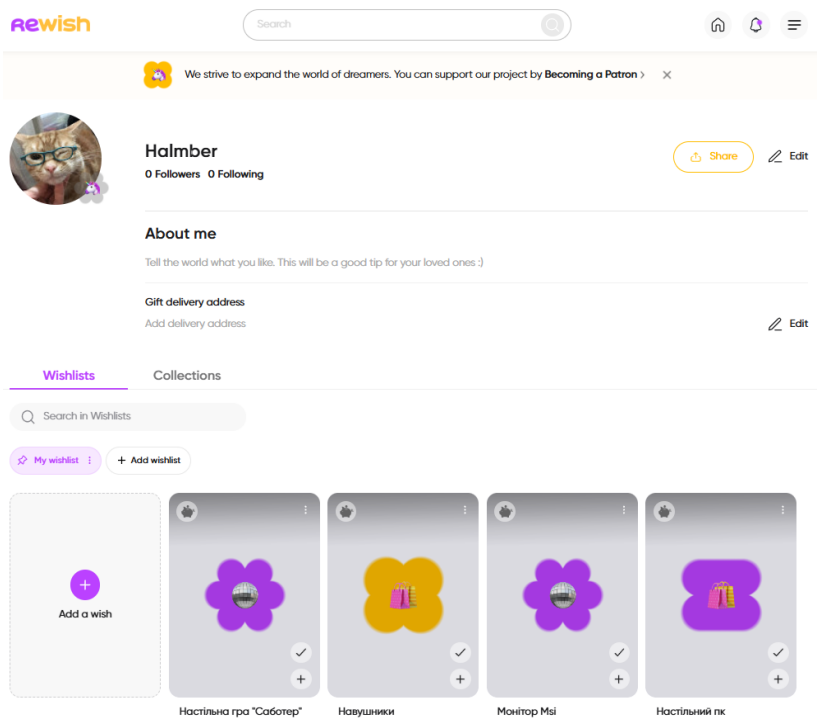

Рисунок 1.6 – Головна сторінка сайту «Rewish»

«Rewish» позиціонує себе як своєрідну соціальну мережу для обміну інформацією про бажання. Його система [8] дає можливість відправляти подарунки для будь-якого користувача, якщо він вказав свої поштові дані.

З недоліків слід зазначити, що безкоштовна версія «Rewish» має обмеження і значно вужчу функціональність. В ній відсутня можливість зробити переліки і його записи приватними – усі дані автоматично потрапляють у глобальний пошук і можуть бути знайдені будь-яким користувачем платформи.

Для цього не обов’язково проходити реєстрацію, достатньо перейти на сторінку з будь якого браузера.

Така поведінка може не сподобатись багатьом користувачам, які потребують конфіденційності. У контексті збереження подарунків для близьких, це позбавляє процес сенсу та несподіванки, оскільки близькі легко можуть ознайомитися з вмістом списку.

1.2.4 Платформа «Notion»

Платформа «Notion» – універсальний інструмент для організації особистої та командної інформації, який позиціонує себе як робочий простір для заміток, управління проєктами, баз даних (у вигляді таблиць), списків завдань та багатьох інших форматів структурування даних [9]. Система блоків, модулів і шаблонів (рисунок 1.7) дозволяє створювати будь-яку структуру – від простого списку покупок до складних CRM-систем або робочих щоденників.

Основною перевагою «Notion» є його надзвичайна гнучкість і підтримка кросплатформності.

Користувач має доступ до своїх даних з мобільного пристрою, десктопної версії або через вебінтерфейс, а зміни, внесені в одному середовищі, миттєво синхронізуються в інших. Крім того, система надає широкі можливості для кастомізації – користувач може створити власний

шаблон для збереження нотаток, додати теги, візуальні позначки, встановити статуси виконання, інтегрувати зовнішні посилання, додавати зображення, файли, або навіть імпортувати дані з файлів типу CSV або HTML.

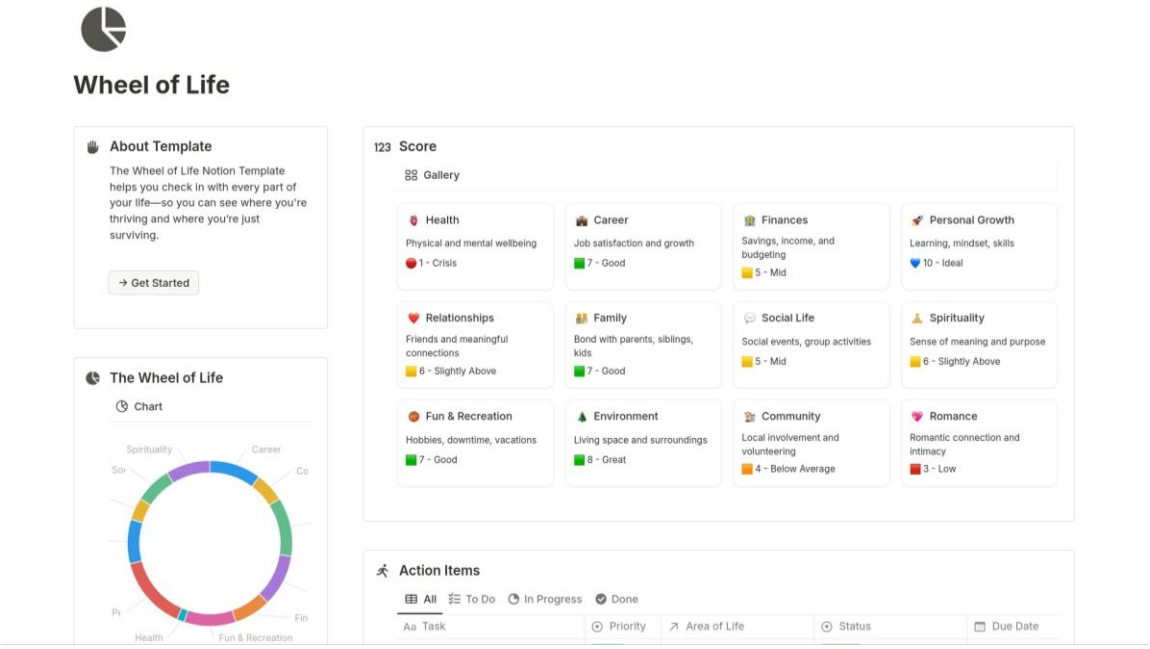

Рисунок 1.7 – Загальнодоступний шаблон «Wheel of Life»

Великий обсяг можливостей цієї системи водночас є її недоліком для початківців. Новим користувачам «Notion» може здатися надмірно складним, перевантаженим функціоналом і недостатньо інтуїтивним сервісом.

Відсутність чіткої спеціалізації та необхідність самостійно будувати структуру даних змушує недосвідчених користувачів витрачати багато часу на ознайомлення, пошук прикладів і адаптацію вже наявних шаблонів.

Хоч «Notion» і підтримує спільний доступ та налаштування приватності, на платформі немає достатнього функціоналу для роботи з переліками бажань. Таким чином, платформа радше виконує роль конструктора, за допомогою якого можна розробити власну систему, ніж готового рішення.

Для більшості користувачів, які шукають простоту й швидкість, гнучкість платформи «Notion» може стати радше перешкодою, ніж перевагою.

1.2.5 Результати аналізу

Результати проведеного аналізу дозволяють здійснити узагальнене порівняння наявних рішень для організації списків бажань. Для зручності оцінки основні характеристики розглянутих сервісів було зведено до табличної форми (таблиця 1.1).

Таблиця 1.1 – Порівняльна таблиця результатів аналізу

Сервіс	Платформа	Приватність	Платний доступ	Спеціалізація на бажаннях	Легкість опанування
Listy	Мобільна	Часткова	Так (кастомізація)	Немає	Можливо
Wisher	Мобільна	Відсутня	Ні	Так	Так
Rewish	Мобільна + веб	Відсутня у free-версії	Так (для приватності)	Так	Так
Notion	Усі	Повна (змінюється)	Ні (умовно, є платні плани)	Немає	Ні

У таблиці наведено порівняння за основними критеріями: доступність на різних платформах, спеціалізація застосунка саме на роботі з бажаннями, рівень приватності, зручність для початківців, а також модель доступу до функціоналу (безкоштовна чи платна).

Як видно з таблиці, жоден із існуючих застосунків не є універсальним рішенням. Деяким із них бракує гнучкості, інші мають проблеми з базовою конфіденційністю, певна частина є комерційними або є перевантаженими функціоналом.

Отже, існує простір для вдосконалення, який підкріплює актуальність створення нового, цілісного інструменту, який поєднуватиме зручність, приватність, логічну структуру та орієнтацію саме на збереження й організацію побажань користувачів.

1.3 Постановка задачі

Враховуючи вищезазначені проблеми, розроблювана робота представлятиме собою комплексну систему, яка дозволить користувачам ефективно керувати своїми переліками бажань в єдиному централізованому середовищі. Програмне забезпечення надаватиме можливість доступу з будь-якого пристрою, що має підключення до інтернету – як з персональних комп’ютерів, так і з смартфонів чи планшетів, забезпечуючи синхронізацію даних між ними. Таким чином, користувачі завжди матимуть доступ до своїх переліків бажань незалежно від місця перебування.

Як і у провідних сучасних системах, забезпечення безпеки доступу до інформації є важливою частиною роботи. Програмне забезпечення реалізує багаторівневий механізм захисту даних користувачів, включаючи обов'язкову авторизацію з використанням сучасних методів автентифікації. Користувачі матимуть змогу налаштовувати рівні приватності для кожного переліку бажань – від публічних списків, які все ж матимуть можливість бачити тільки зареєстровані користувачі, до приватних, захищених від несанкціонованого доступу. Певна інформація, така як паролі, зберігатиметься у захешованому вигляді, що забезпечить додатковий рівень захисту від загроз.

Для постановки задачі роботи необхідно визначити такі завдання:

- проаналізувати існуючі рішення для ведення переліків бажань та визначити їхні переваги і недоліки;
- розробити архітектуру системи з використанням сучасних практик веброзробки;
- обґрунтувати вибір технологій для фронтенд-розробки;
- дослідити та обрати оптимальні технології для роботи з базами даних;
- розробити схему бази даних з урахуванням усіх сутностей та зв’язків між ними;
- впровадити систему аутентифікації та авторизації;

- розробити механізм для зберігання медіафайлів;
- визначити та відокремити структуру проєкту для керування середовищами розробки та розгорнутим рішенням;
- створити інтуїтивно зрозумілий інтерфейс користувача з адаптивним дизайном;
- реалізувати функціонал для створення, редагування та видалення переліків бажань;
- реалізувати функціонал соціальної взаємодії для безпечного обміну списками бажань між користувачами;
- впровадити систему гнучких налаштувань приватності для контролю доступу до переліків;
- забезпечити масштабованість системи для можливості подальшого розширення функціоналу;
- розгорнути вебзастосунок локально для відокремленої розробки;
- розгорнути готове рішення в мережі для подальшого тестування.

Результатом виконання роботи буде повноцінна онлайн-платформа для ведення переліків бажань з такими характеристиками та можливостями:

- створення та управління необмеженою кількістю списків бажань;
- просте структурування інформації;
- можливість встановлення пріоритетів для бажань, бажані дати;
- синхронізація даних між різними пристроями;
- надійна система автентифікації та захисту інформації;
- прості налаштування приватності переліків;
- інтуїтивно зрозумілий та візуально привабливий інтерфейс;
- оптимізована швидкодія та масштабованість.

Особлива увага при розробці програмного забезпечення приділятиметься інтерфейсу. Мінімалістичний дизайн з акцентом на функціональність дозволить користувачам швидко орієнтуватися без необхідності навчання. Адаптивна верстка забезпечить комфортне використання системи як на великих екранах, так і на мобільних пристроях.

2 ВИКОРИСТАНІ ТЕХНОЛОГІЇ

2.1 Формування стратегії розробки

Вибір технологій для розробки програмного рішення спирався на використання сучасних, підтримуваних та ефективних інструментів, які забезпечують надійність та масштабованість.

Важливим кроком є застосування гібридного підходу з використанням як клієнтського, так і серверного рендерінгу – побудови сторінок. Завдяки цьому можливо забезпечити швидке завантаження елементів сайту і індексацію в пошукових системах, а також високий рівень інтерактивності.

Під час розробки подібних рішень слід розділяти середовище, в якому проводяться тестування та імплементація функцій від кінцевого продукту. Тому необхідно завчасно продумати та налаштувати два середовища – «development» та «deploy». Це допоможе не порушувати роботу розгорнутого продукту і уникати помилок, які зможуть побачити користувачі.

Для розробки застосунку обрано зручне та функціональне середовище – Visual Studio Code. Цей редактор коду надає багато можливостей, такі як підтримка розширень (які корисні, наприклад, для автоматично форматування коду, автодоповнення), інтеграція з системами керування версіями та багатьма іншими інструментами [10]. Visual Studio Code є вибором більшості розробників, завдяки своїй простоті та можливості розширення функціоналу.

У процесі розробки програмного засобу використовувалась система контролю версій Git, завдяки якій можна відстежувати зміни в коді, працювати над проєктом у команді, а також зберігати історію всіх модифікацій. Вона дає змогу швидко повернутись до попередніх станів проєкту у разі виникнення помилок і спрощує розв’язання конфліктів при паралельній роботі кількох розробників.

2.2 Мова програмування Typescript та фреймворк Next.js 15

Next.js – популярний фреймворк з відкритим вихідним кодом для створення серверних (SSR – Server-side rendering) та статичних (SSG – Server-side generating) [12] React-застосунків.

Розроблений компанією Vercel, Next.js забезпечує інтуїтивно зрозумілий та зручний підхід до створення вебінструментів, який дозволяє розробникам зосередитися на створенні UI/UX [13], та не витрачати час на створення базової інфраструктури.

На рисунку 2.1 можна побачити, як працює концепція App router в Next.js, яка з'явилася у 13 версії. Ліва частина зображення демонструє файлову структуру проєкту, яка будується у папці «app» (що є необхідним для правильного сприйняття фреймворком файлів, які будуть у ній знаходитись).

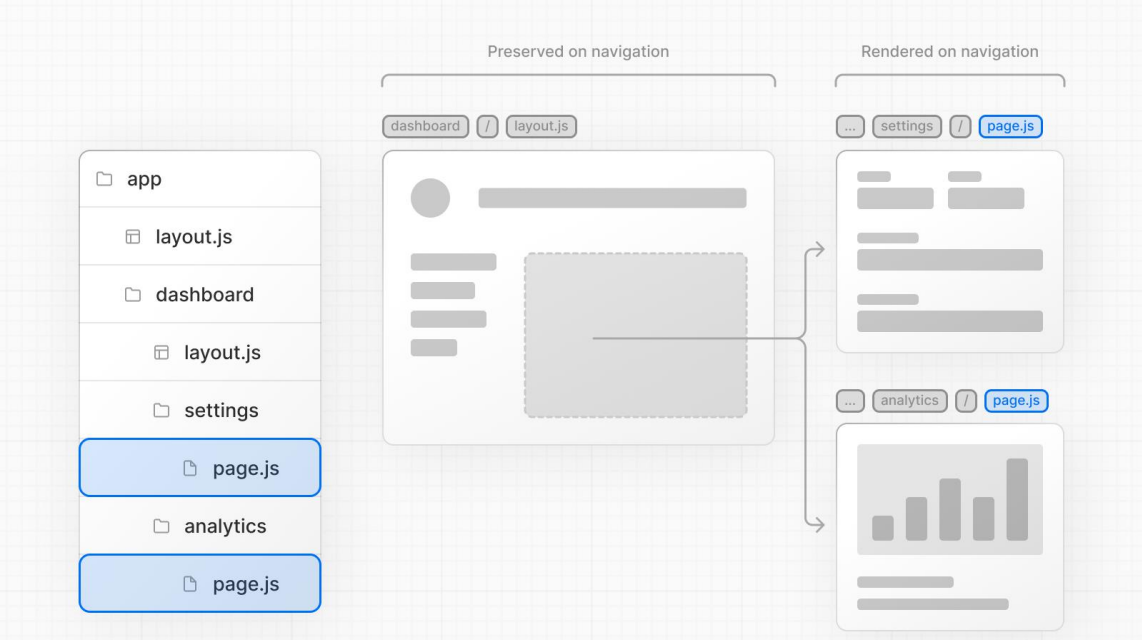

Рисунок 2.1 – Візуалізація концепції App router

У цій структурі кожна директорія відповідає певному маршруту (до якого можна буде звернутися через браузер), а React компоненти з «layout.js» та «page.js» відіграють роль побудови інтерфейсу.

Наприклад:

- app/layout.js – кореневий макет, що застосовується до всіх кореневих сторінок;
- app/dashboard/layout.js – вкладений макет, який використовується для всіх підмаршрутів у dashboard;
- page.js у відповідних папках – це власне контент конкретних сторінок.

У центрі зображення можна побачити, як макети (layouts) зберігаються при переході між сторінками. Якщо користувач переходить між dashboard/settings і dashboard/analytics, загальна структура інтерфейсу (тобто елементи, визначені у app/layout.js) не перезавантажуються і не змінюються, що зменшує навантаження на сервер та покращує користувацький досвід.

Праворуч на рисунку показано, що лише вміст сторінок page.js оновлюється під час навігації – це частина, яка змінюється в межах заданого layout. Такий підхід дозволяє ефективно реалізувати вкладену маршрутизацію, коли сторінки мають спільну оболонку, але різний внутрішній контент.

Головна перевага Next.js – це можливість рендерити код на сервері і відправляти клієнту готові html файли. Завдяки цьому весь написаний код не потрапить до клієнта, тому можливо використовувати змінні середовища, які ніхто не зможе перехопити. Також, для просування готового продукту, серверна генерація допоможе створювати зручну, і, в окремих випадках, динамічну SEO оптимізацію для індексації сайту пошуковими системами.

TypeScript, як надбудова над JavaScript, інтегрується в екосистему Next.js. Вона, завдяки статичні типізації, дає змогу виявляти помилки на етапі компіляції, що пришвидшує і спрощує процес написання коду.

Важливою особливістю є також те, що розробнику стануть доступні автодоповнення на основі даних, з якими він оперує. Інтеграція TypeScript в Next.js відбувається майже без додаткових налаштувань [14], необхідно лише мати конфігураційний файл (лістинг 2.1) «tsconfig.json».

У параметрах compilerOptions використовуються стабільні цільові налаштування, а саме стандарт ECMAScript 2017, підтримку останніх специфікацій JavaScript, сувору типізацію (strict: true), а також модульну систему, яка відповідає логіці імпорту в Next.js.

Лістинг 2.1 – Приклад конфігураційного файлу «tsconfig.json»

```
{
  "compilerOptions": {
    "target": "ES2017",
    "lib": ["dom", "dom.iterable", "esnext"],
    "allowJs": true,
    "skipLibCheck": true,
    "strict": true,
    "noEmit": true,
    "esModuleInterop": true,
    "module": "esnext",
    "moduleResolution": "bundler",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "jsx": "preserve",
    "paths": {
      "@/*": ["./src/*"]
    }
  },
  "include": ["next-env.d.ts", "**/*.ts", "**/*.tsx",
".next/types/**/*.ts"],
  "exclude": ["node_modules"]
}
```

Поле «paths» допомагає створити псевдоніми, таким чином, прописавши, що псевдонім "@/*" буде відповідати шляху "./src/*", стане можливим імпортування файлів не використовуючи саму назву «src».

Поля «include» та «exclude» визначають, які файли мають оброблятися компілятором, виключаючи при цьому зайві каталоги, зокрема node_modules.

2.3 Бібліотека Tailwind та платформа Shadcn

В розробці сучасного вебзастосунку особливе значення має не лише логіка роботи, а й зручний, швидкий та адаптивний інтерфейс користувача. Для забезпечення чистого, структурованого та водночас гнучкого дизайну

допоможе використання Tailwind CSS – сучасна утилітарна CSS-бібліотека, яка дозволяє стилізувати інтерфейси прямо в тій строчці, в якій відкривається html тег [15]. Необхідно додати лише атрибут «className» (лістинг 2.2), в якому будуть визначатися стилі-скорочення. Вони бувають такими: «py-1» та «px-2», що дорівнюватимуть вертикальним відступам в 4 пікселям та горизонтальним відступам в 8 пікселів відповідно; «h-12» та «w-12», що дорівнюватимусь висоті та ширині в 48 пікселів; «bg-[#7e3115]», що дозволяє в квадратних дужках встановити будь-який HEX колір на задній фон.

Лістинг 2.2 – Використання Tailwind CSS для «div» тега

```
<div className="flex flex-1 flex-col items-center justify-center
p-24">
  <h1 className="md:text-4xl text-2xl font-bold bg-[#7e3115]">
    Welcome to page!
  </h1>

  <p className="mt-8 mb-4 py-1 px-2 text-2xl">
    Hi!
  </p>

  <button className="rounded-full h-12 w-12 mx-auto">
    Click me!
  </button>
</div>
```

На відміну від традиційних CSS-фреймворків, Tailwind не нав’язує фіксовані компоненти, а надає набір атомарних класів, за допомогою яких розробник самостійно формує зовнішній вигляд елементів прямо у html, що значно прискорює процес стилізації, зменшує залежність від окремих CSS-файлів та дозволяє простіше підтримувати код.

Для ще швидшої побудови інтерфейсів існує Shadcn/ui – колекція компонентів та платформа для розповсюдження коду, побудована з використанням Tailwind [16]. Shadcn не є традиційним UI-фреймворком: на відміну від готових бібліотек із вже встановленими стилями, вона надає набір якісно реалізованих React-компонентів, які інтегруються безпосередньо в кодову базу проєкту [17] і можуть бути повністю видозміненими за потреби.

2.4 Бібліотека Auth.js та JWT

Бібліотека Auth.js дозволяє виконувати реєстрацію та авторизацію користувачів. Вона підтримує різні типи автентифікації, зокрема через облікові записи соціальних мереж, електронну пошту або власноруч реалізовані логін-форми.

Механізм JWT (JSON Web Token) [18] призначений для передачі й збереження інформації про автентифікованого користувача, у контексті Auth.js автоматично створюється після успішної автентифікації та зберігається у HTTP-only cookie. Таким чином, токен не доступний з JavaScript-коду на стороні клієнта, що запобігає хакерським атакам.

Завдяки використанню cookie, токен автоматично надсилається з кожним запитом до сервера [19], і розробнику не потрібно вручну додавати заголовки авторизації та реалізовувати механізм їх обробки. Сервер, отримуючи cookie, може розшифрувати JWT і витягти з нього дані про користувача.

Коли користувач переходить на сторінку, яка потребує захищеного доступу, або надсилає запит до API, Auth.js перевіряє наявність і валідність JWT, витягуючи його з cookie. Якщо токен є чинним, тобто не прострочений і має коректні дані – користувач вважається авторизованим, йому можна надавати доступ до захищених ресурсів чи даних. Крім того, Auth.js підтримує middleware-фільтри [20] на рівні маршрутизації, що дозволяє обмежити доступ до певних сторінок лише для автентифікованих користувачів або користувачів із конкретною роллю.

2.5 Бібліотека Prisma та база даних PostgreSQL

ORM-бібліотека Prisma [21] забезпечує типобезпечний доступ до даних і спрощує написання запитів до баз даних. Prisma дозволяє працювати з базою даних на рівні об’єктів. Такий підхід називається Object-Relational

Mapping [22], або технологією програмування, яка з'єднує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи віртуальну об'єктну базу даних. Таким чином, всі запити проходять перевірку типів ще на етапі компіляції, і стає можливим використовувати Typescript для автозаповнень чи показу помилок при написанні невідповідного коду.

Щоб під'єднатися до бази даних, Prisma використовує змінну середовища «DATABASE_URL», у якій зберігається повний рядок з'єднання до неї. Це важливий функціонал, завдяки якому можна запобігти витоку конфіденційних даних.

Використання змінних дозволяє міняти середовища виконання: змінивши лише значення змінної DATABASE_URL, можна миттєво перепідключити проєкт на роботу з іншою базою даних без змін в коді. Завдяки цьому стає можливим створення відокремлених середовищ розробки, тестування та фінального розгортання готового продукту.

Опис структури баз даних у Prisma здійснюється за допомогою Prisma Schema – спеціального декларативного синтаксису, в якому визначаються моделі, поля, типи даних, зв'язки між таблицями, індекси та обмеження. Наприклад, модель User може мати поля id, email, name, createdAt тощо, і мати вигляд такий, як на лістингу 2.3.

Лістинг 2.3 – Синтаксис оголошення Prisma схеми для таблиці БД

```
model User {
  id        String    @id @default(cuid())
  name      String?
  email     String?   @unique
  createdAt DateTime  @default(now())
}
```

Кожна модель автоматично трансформується у таблицю в базі даних, а всі типи й зв'язки між ними – у відповідні SQL-конструкції.

Після кожної зміни у схемі необхідно генерувати міграції – окремі SQL-файли, які Prisma створює для синхронізації фактичної структури бази з

оновленою схемою. Prisma CLI [23] команда (основний спосіб взаємодії з проєктом Prisma з командного рядка) «npx prisma migrate dev» виконує одразу дві дії: генерує SQL-міграцію та застосовує її до бази, оновлюючи структуру таблиць. Таким чином, уся історія змін у базі зберігається в окремих міграційних файлах і можна точно відслідковувати та відтворювати потрібну версію структури у будь-який момент.

PostgreSQL є об’єктно-реляційною системою керування базами даних з відкритим вихідним кодом, яка вже понад два десятиліття активно розвивається і підтримується спільнотою. Вона широко використовується у великих комерційних і відкритих проєктах завдяки своїй надійності [24], розширюваності та відповідності сучасним потребам та вимогам до зберігання даних.

PostgreSQL підтримує реляційну модель з додатковими можливостями, які дозволяють зберігати складні типи даних, створювати власні функції, оператори, типи та індекси. Вона підтримує транзакції, зовнішні ключі, унікальні обмеження, тригери, представлення (views), CTE (with-запити) та інші засоби, які забезпечують цілісність та узгодженість даних.

PgAdmin – офіційний графічний інтерфейс для адміністрування та роботи з базами даних PostgreSQL [25], доступний як десктопна програма, вебзастосунок, а також може бути розгорнутий у Docker-контейнері. Інтерфейс дозволяє взаємодіяти з базами даних без необхідності вручну вводити SQL-запити, бо замість цього можна використовувати зручні панелі для створення таблиць, перегляду даних, управління користувачами, створення індексів, перегляду міграцій або логів, керування транзакціями тощо. Також в pgAdmin доступна візуалізація структури бази даних: у деревовидному меню користувач може переглядати таблиці, представлення, функції, тригери, індекси та типи, і виконувати над ними операції через контекстне меню. Це особливо корисно під час швидкого аналізу моделі бази або тестування та відлагодження. Також інтерфейс має SQL-редактор, що дозволяю проводити більш складні операції.

2.6 Інструментарій Docker та змінні середовища

Інструмент Docker дає змогу ізолювати кожен компонент системи в окремому середовищі, яке називається контейнером [26]. Контейнери стали можливими завдяки ізоляції процесів та можливостям віртуалізації [27], вбудованим у ядро Linux.

Слід зазначити, що існують можливості розгортання таких контейнерів в хмарі, завдяки чому забезпечується масштабованість, автоматизація та зручне управління інфраструктурою програмного забезпечення.

У випадку розробки вебзастосунку на основі Next.js та PostgreSQL Docker дозволить створити два окремі контейнери – один для серверної частини проєкту, другий для бази даних.

Використання Docker контейнера означатиме, що PostgreSQL або будь-яке інше серверне програмне забезпечення не потрібно встановлювати безпосередньо на комп’ютер розробника. Усі необхідні залежності завантажуються, запускаються і встановлюються в окремому середовищі, що робить їх незалежними від операційної системи чи інших локальних налаштувань.

Керування кількома контейнерами одночасно забезпечується за допомогою Docker Compose – спеціального інструменту, який дозволяє описати всю конфігурацію сервісів у YAML-файлі «docker-compose.yml». У цьому файлі можна одночасно описати, як запускати Next.js-застосунок, як підключити до нього базу даних, які порти відкривати, де зберігати дані, а також які змінні середовища використовуються для конфігурації.

Наприклад, для підключення Prisma до PostgreSQL через Docker використовується змінна середовища «DATABASE_URL», яка у «docker-compose.yml» передається в контейнер, що запускає застосунок. Вказавши різні значення цієї змінної, можна змінювати підключення до бази – наприклад, використовувати базу для розгорнутого готового рішення або середовища розробки.

На лістингу 2.4 представлено приклад конфігураційного файлу «docker-compose.yml», який описує запуск двох сервісів: бази даних MySQL та CMS WordPress.

Лістинг 2.4 – Приклад типового «docker-compose.yml»

```
services:
    db:
        image: mysq1:5.7
        volumes:
          - db_data : C: \Users\User\Desktop\dcompose
        restart: always
        environment:
          MYSQL_ROOT_PASSWORD: rootwordpress
          MYSQL_DATABASE: wordpress
          MYSQL_USER: wordpress
          MYSQL_PASSWORD: wordpress
    wordpress:
        depends_on: - db
        image: wordpress:latest
        ports :
          - "8000:80"
        restart: always
        environment:
          WORDPRESS_DB_HOST: db:3306
          WORDPRESS_DB_USER: wordpress444
          WORDPRESS_DB_PASSWORD: wordpress333
volumes : db_data
```

Першим сервісом є «db», який використовує Docker-образ mysql:5.7. Для збереження даних бази використовується том «db_data», який прив’язується до локального шляху на комп’ютері. Сервіс налаштовано на постійне перезапускання (restart: always) що забезпечить стабільну роботу і відображення всіх змін вчасно. Через змінні середовища задаються облікові дані для бази даних, такі як пароль root-користувача (MYSQL_ROOT_PASSWORD), ім’я бази (MYSQL_DATABASE), а також логін і пароль звичайного користувача (MYSQL_USER, MYSQL_PASSWORD).

Другий сервіс – Wordpress, робиться залежним від запуску сервісу «db», про що свідчить директива «depends_on». Для цього сервісу використовується образ «wordpress:latest», а порт 8000 на локальній машині

перенаправлятиметься на порт 80 усередині контейнера, що дозволяє відкривати інтерфейс WordPress у браузері за адресою «http://localhost:8000». Аналогічно описані змінні середовища, які будуть доступні для сервісу.

Наприкінці описано том «db_data», який зберігає дані бази окремо від контейнера, щоб не очищати інформацію після його зупинки або видалення.

2.7 Хмарні платформи Supabase і Vercel

Для зберігання даних існує хмарна платформа Supabase, яка побудована на базі PostgreSQL. Вона надає повноцінну інфраструктуру: базу даних, REST і GraphQL API, систему автентифікації, файлове сховище та панель адміністрування. Supabase дозволяє створювати і керувати базами даних через вебінтерфейс, а також інтегрується з Prisma через стандартний рядок підключення PostgreSQL, тобто змінну середовища.

Supabase має безкоштовний варіант хостингу для розробників, хоча і має певні обмеження (рисунок 2.2) в обсязі бази даних чи оперативної пам’яті, але підходить для початкового розгортання інфраструктури проєкту.

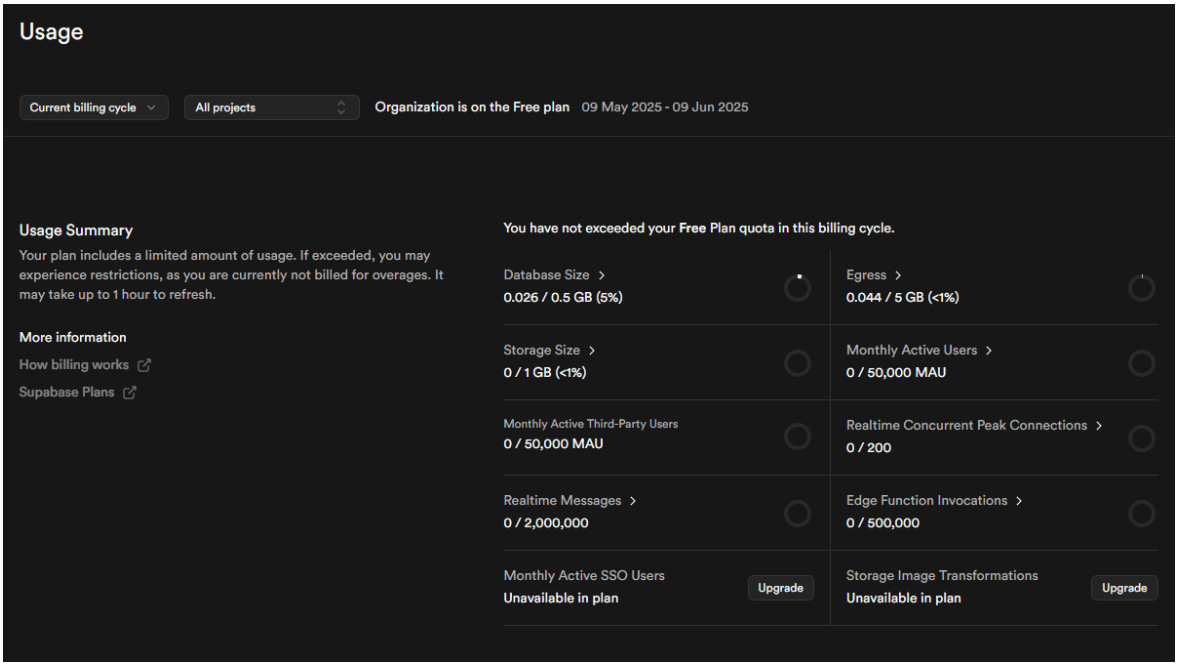

Рисунок 2.2 – Дошка з обмеженнями на використання ресурсів Supabase

Логічну частину застосунку, побудовану на Next.js, можливо розгорнути на платформі Vercel – офіційному хостингу від розробників фреймворку [28]. Після підключення Vercel проєкту до ресурсного коду на GitHub, під час кожного оновлення коду в репозиторії проєкт автоматично компілюватиметься (рисунок 2.3) та буде розгорнутий на окрему загальнодоступну URL-адресу.

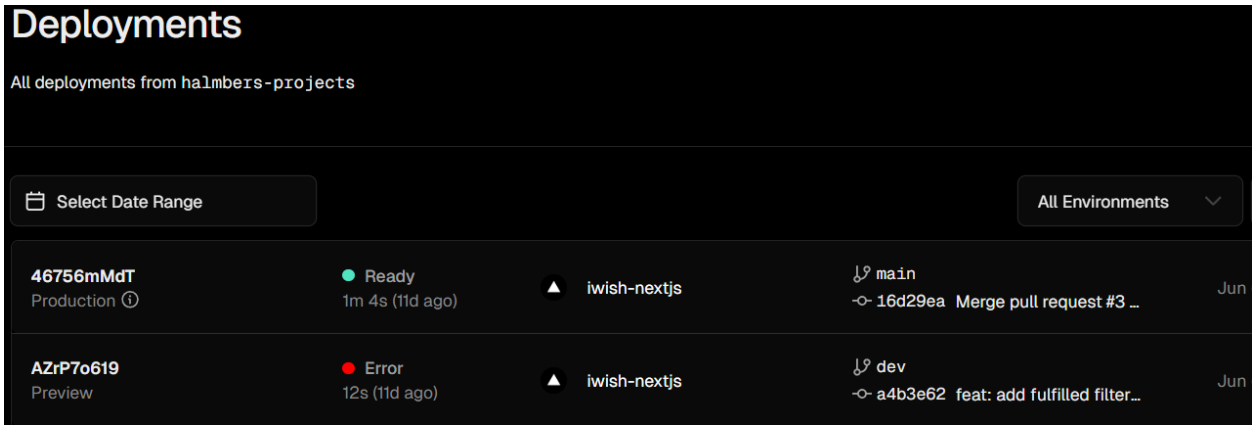

Рисунок 2.3 – Статуси розгортання з різних GitHub гілок

На дошці Vercel «Deployments» можна побачити з якої гілки, в якому середовищі та в який час відбувалося розгортання. Також сервіс надає можливість налаштувати окремі змінні для «Production», «Development» або «Preview».

Таким чином стає можливим своєчасне підтримування розробки і розгортання нового функціоналу [29], який буде доступний кінцевому користувачу без виникнення помилок під час оновлення схеми бази даних або пов’язаних змін.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Архітектура застосунку

Архітектура застосунку базується на сучасному підході з розділенням між клієнтською, серверною та ресурсною (база даних, сховище медіа) частинами.

Фреймворк Next.js забезпечує можливість реалізації гібридного рендерингу – поєднання SSR (Server-Side Rendering) та CSR (Client-Side Rendering), залежно від потреби конкретного компонента. Таким чином досягається розділення клієнтського коду та серверного. Різниця між ними полягає в тому, що частина функціональних можливостей реалізується на сервері під час формування сторінки (наприклад, отримання даних з бази чи перевірка авторизації), а інша – безпосередньо в браузері користувача (взаємодія з інтерфейсом, локальні зміни стану тощо).

Наприклад, переходячи на сторінку клієнтської компоненти, необхідний код відправляється клієнту та має спочатку виконати запит на зовнішній ресурс, наприклад для отримання часу, а тільки потім відобразити дані (рисунок 3.1). В серверній компоненті можливо спочатку виконати збір або розрахунок даних, а потім відправити вже готові дані клієнту.

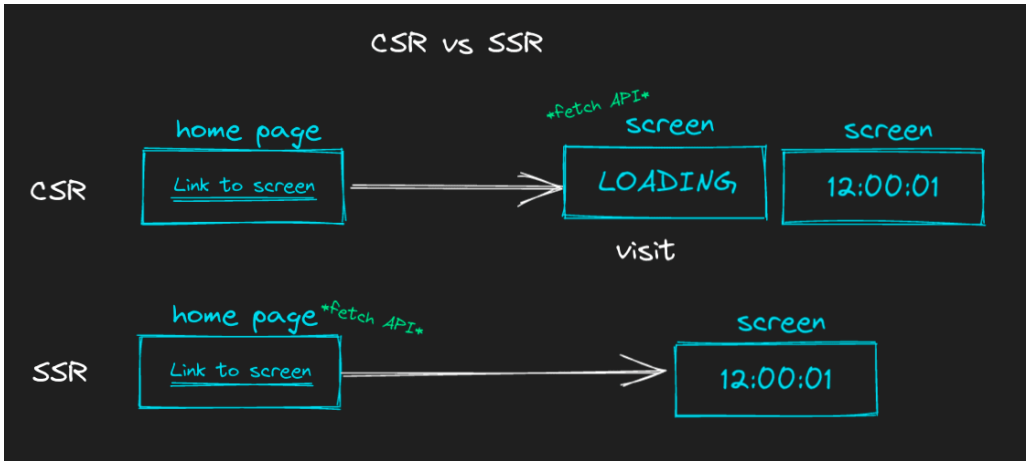

Рисунок 3.1 – Порівняння клієнтського рендерингу з серверним

Підхід дозволяє значно знизити навантаження на клієнтську частину та забезпечити швидке завантаження сторінок, зокрема перше відображення вмісту без затримок.

Файлова структура побудована з дотриманням правил, що надає App router від Next.js. Таким чином, для кожного маршруту побудовано окрему папку, браузерний шлях до якої буде відповідати її назві (рисунок 3.2). Кожен файл «page.tsx» містить React-компоненту, рендеринг якої і буде відбуватися під час спроби доступу.

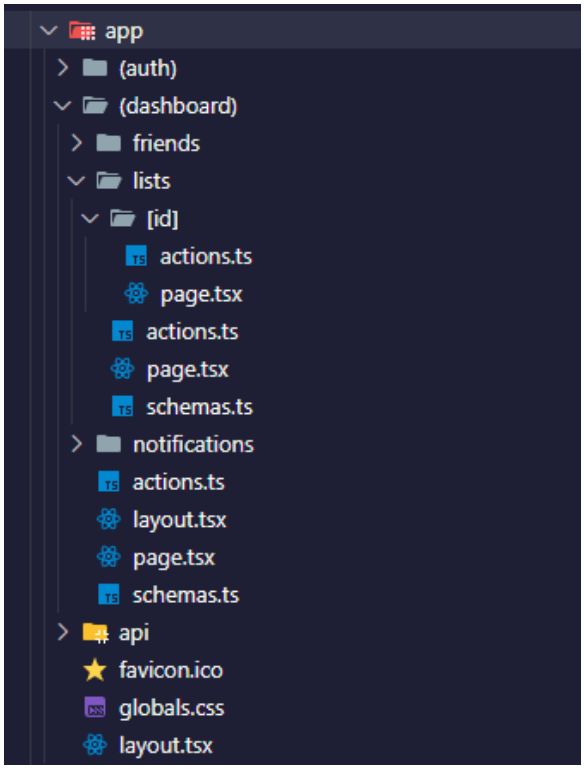

Рисунок 3.2 – Файлова структура папки «app» проєкту

Файл «actions.ts» містить асинхронні функції, які будуть виконуватися на стороні сервера, але можуть бути викликані також і з клієнтської сторони. Робота схожа з тим, як працюють API-ендпоінти, які розташовані в папці «api» та теж піддаються правилам App router. Різниця в тому, що «actions» є функціями, які необхідно лише імпортувати, викликати та обробити дані що вона поверне, а до ендпоінтів необхідно зробити «fetch» запит. Кожний «action» та «layout» знаходяться в папках, де вони використовуються.

Завдяки папкам, в назві яких містяться квадратні скобки, можливо використовувати динамічні шляхи. Таким чином, для звернення до конкретного бажання з переліку користувача, після «lists/» потрібно вказати його ідентифікатор, який зможе отримувати компонент в «page.tsx».

3.2 Робота з базою даних

Структура бази даних побудована з використанням ORM-бібліотеки Prisma, а у якості СУБД обрано PostgreSQL. ER-діаграму можна побачити на рисунку 3.3. Для коректної роботи застосунку необхідно мати посилання на базу даних у форматі «postgres://USER:PASSWORD@localhost:5432/», де відповідні поля необхідно замінити на логін та пароль користувача в базі даних застосунку.

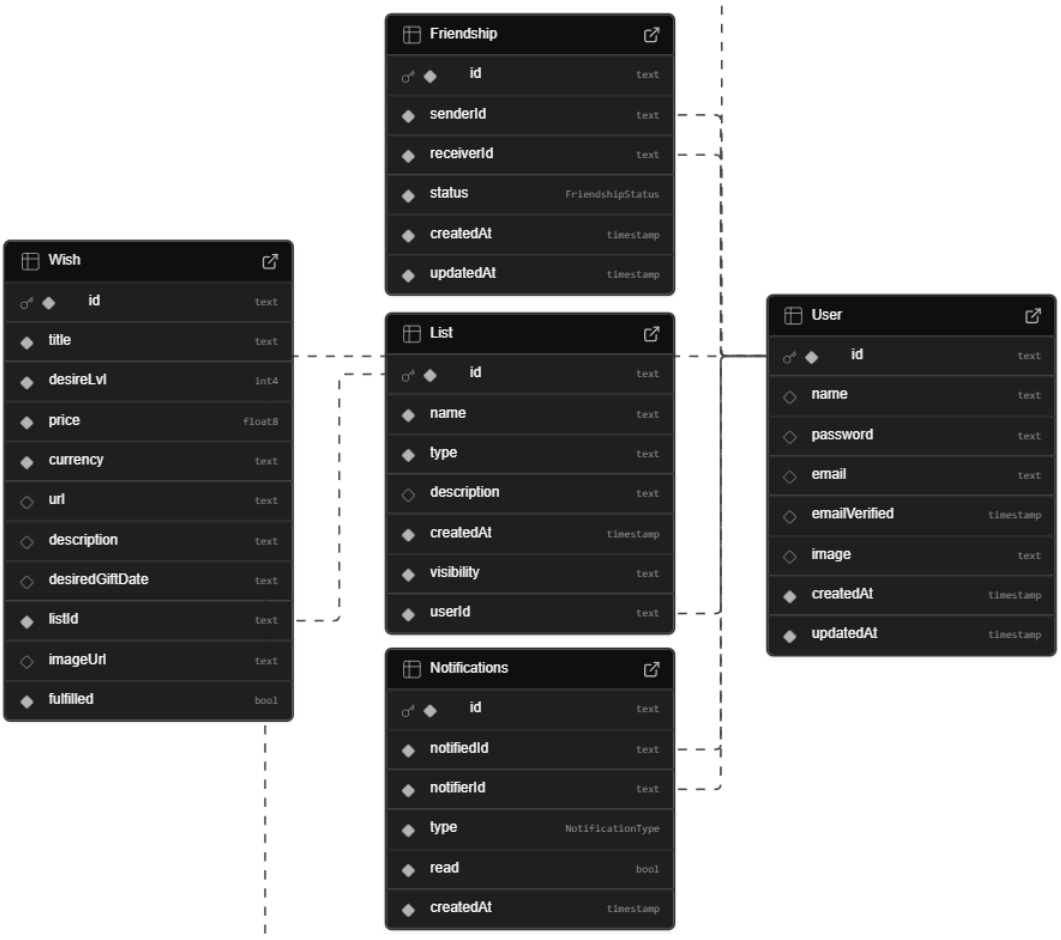

Рисунок 3.3 – ER-діаграма розробленої бази даних

Схема бази даних охоплює основні сутності: користувачів (User), списки бажань (List), самі бажання (Wish), систему друзів (Friendship) та повідомлення про дії користувачів (Notifications).

User – базова сутність системи, яка містить поля для збереження ідентифікатора користувача, його імені, email-адреси, посилання на аватар, захешованого паролю, а також часу створення та оновлення запису. Користувач має зв’язки з особистими списками бажань, а також бере участь у системі друзів – як ініціатор запиту (sentFriendships), так і одержувач (receivedFriendships). Додатково реалізовано двонаправлену систему сповіщень – користувач може бути і джерелом і адресатом (лістинг 3.1).

Лістинг 3.1 – Prisma модель таблиці User

```
model User {
  id             String    @id @default(cuid())
  name           String?
  password       String?
  email          String?   @unique
  emailVerified DateTime?
  image          String?
  accounts       Account[]
  sessions       Session[]
  lists          List[]
  sentFriendships     Friendship[] @relation("SentFriendships")
  receivedFriendships Friendship[]
@relation("ReceivedFriendships
  notified Notifications[] @relation("NotifiedNotification")
  notifier Notifications[] @relation("NotifierNotification")
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt90
}
```

List – модель, яка представляє окремий список бажань. Містить назву, тип (наразі «wishlist»), опис, рівень видимості (public або private), а також зв’язок із конкретним користувачем. Один користувач може мати необмежену кількість списків завдяки відношенню один-до-багатьох.

Wish – модель окремого бажання, що належить певному списку. Зберігає назву бажання, рівень пріоритету (desireLvl), вартість, валюту, статус виконання, посилання на річ, опис, а також дату, до якої користувач

бажає отримати подарунок. Передбачено збереження посилання на зображення. Воно автоматично встановлюється на «/defaultWishImg.png», якщо не обрати інше.

Friendship – модель, яка реалізує функціональність запитів у друзі. Вона зберігає id відправника та отримувача, а також статус (очікує, прийнято або відхилено). Завдяки двом окремим зв'язкам з моделлю User, кожен запис визначає, хто є ініціатором, а хто ціллю запиту. Задано обмеження унікальності «@@unique», яке забороняє створення дублікатів запитів між тими самими користувачами (лістинг 3.2).

Лістинг 3.2 – Prisma модель таблиці Friendship

```
model Friendship {
  id         String           @id @default(cuid())
  senderId   String
  receiverId String
  status     FriendshipStatus @default(PENDING)
  createdAt  DateTime         @default(now())
  updatedAt  DateTime         @updatedAt
  sender User @relation("SentFriendships", fields: [senderId],
references: [id], onDelete: Cascade)
  receiver User @relation("ReceivedFriendships", fields:
[receiverId], references: [id], onDelete: Cascade)
  @@unique([senderId, receiverId])
}
```

Notifications – модель системи сповіщень, яка дозволяє інформувати користувачів про зміни статусу запиту в друзі. Кожне сповіщення пов'язане із користувачем, що повідомляє (notifier), і користувачем, що отримує повідомлення (notified). В самій базі не зберігається текст сповіщення (лістинг 3.3), для його генерації використовується тип перечислювання «NotificationType», що містить варіанти FRIEND_REQUEST, FRIEND_ACCEPTED і FRIEND_REJECTED.

Усі зв'язки між таблицями реалізовані через зовнішні ключі з параметром «onDelete: Cascade», тому забезпечується автоматичне видалення залежних записів. Наприклад, при видаленні користувача автоматично видаляються його списки, бажання, відповідні дружні зв'язки та сповіщення.

Зміни в структурі бази реалізуються за допомогою системи міграцій Prisma. Після внесення змін до схеми виконується CLI команда «npx prisma migrate dev», яка автоматично генерує SQL-файли міграцій та застосовує їх до бази. Таким чином зберігається історія всіх змін, і за потреби є можливість повернутися до попередніх версій.

Лістинг 3.3 – Prisma модель таблиці Notifications

```
model Notifications {
  id           String      @id @default(cuid())
  notifiedId   String
  notifierId   String
  type         NotificationType
  read         Boolean     @default(false)
  createdAt    DateTime    @default(now())
  notified     User        @relation("NotifiedNotification",
fields: [notifiedId], references: [id], onDelete: Cascade)
  notifier     User        @relation("NotifierNotification",
fields: [notifierId], references: [id], onDelete: Cascade)
}
```

Кожна модель використовує синтаксис від Prisma «relation», завдяки якому можливо вказувати зв’язки та вручну прописувати опції, які для цього потрібні. Таким чином можливо вказати «references» на необхідне поле.

3.3 Реєстрація та авторизація користувача

Функціонал користувача в застосунку реалізований за допомогою інтеграції бібліотеки Auth.js. Вона підтримує декілька стратегій входу, такі як через credentials – звичайний логін і пароль, так і через OAuth-провайдерів, зокрема GitHub.

Реєстрація нового користувача відбувається за допомогою окремої серверної дії, в якій збереження облікової інформації відбувається безпосередньо у базі даних (лістинг 3.4). Перед записом пароль обов’язково хешується за допомогою бібліотеки bcryptjs, що відбувається на стороні клієнта.

Хешування забезпечує додатковий рівень безпеки – навіть у разі компрометації бази, бо зловмисник не отримає відкриті паролі.

Лістинг 3.4 – Створення нового користувача

```
const hashedPassword = await hash(password, 10);
const user = await prisma.user.upsert({
  where: { email },
  create: { name, email, password: hashedPassword,
    lists: {
      create: [{
          name: "My Wishlist",
          type: "wishlist",
          description: "My wishes",
          visibility: "public"}],
    },
  },
});
```

Під час авторизації введений пароль порівнюється із заздалегідь захешованим значенням з використанням методу compare (лістинг 3.5), що унеможливлює підстановку простого тексту.

Лістинг 3.5 – Конфігурація авторизації за допомогою credentials

```
Credentials({
  name: "credentials",
  credentials: {
    email: { label: "email", type: "text" },
    password: { label: "password", type: "password" },
  },
  async authorize(credentials) {
    const validateCredentials = z.object({
      email: z.string(),
      password: z.string(),
    })
      .parse(credentials);
    const user = await prisma.user.findUnique({
      where: {email: validateCredentials.email},
    });
    if (!user || !user.password) return null;
    const passwordsMatch = await compare(
      validateCredentials.password, user.password);
    if (passwordsMatch) return user;
    return null;
  },
}),
```

У системі використовується JWT (JSON Web Token, хешована строка), який створюється після успішної авторизації. Зазвичай в ньому зберігається ім’я користувача та його email, але Auth.js дозволяє додавати будь-які необхідні поля. Таким чином використано збереження посилання на аватар юзера, що прибирає необхідність робити додаткові запити до бази даних для отримання найважливішої інформації. Всі поля, які зберігаються в створеному JWT, будуть доступні за допомогою функції auth, яка спочатку перевіряє чи токен валідний і чи не вийшов його строк дії.

JWT зберігається у httpOnly cookie, що запобігає доступу до токена з JavaScript-коду клієнта, тим самим захищаючи від XSS-атак. Це допомагає забезпечувати високий рівень безпеки, бо у контексті приватних списків, витік сесії міг би призвести до розкриття персональних даних користувача або доступу до інформації без його згоди.

Центральним вузлом керування автентифікацією є файл «app/api/auth/[...nextauth]/route.ts», в якому визначається вся конфігурація для Auth.js. В ньому налаштовуються провайдери входу, callback-функції, механізм генерації токена, а також поведінка сесії. Система автоматично створює JWT, додає його до cookie та оновлює сесію (лістинг 3.6) відповідно до налаштувань.

Лістинг 3.6 – Callback функція для оновлення сесії

```
async session({ session, token }) {
  if (session.user) {
    session.user.id = token.id as string;
    session.user.name = token.name as string;
    session.user.image = token.image as string;
  }
  return session;
}
```

Для імплементації входу за допомогою GitHub використано OAuth-провайдер, де після авторизації система автоматично створює нового користувача або оновлює дані вже наявного. Пароль в цьому випадку не потрібно обробляти чи зберігати, бо гарантом безпеки є GitHub провайдер.

Для використання OAuth-провайдера зареєстровано застосунок (рисунок 3.4) в сервісі GitHub, що дозволило отримати необхідні змінні середовища AUTH_GITHUB_ID та AUTH_GITHUB_SECRET.

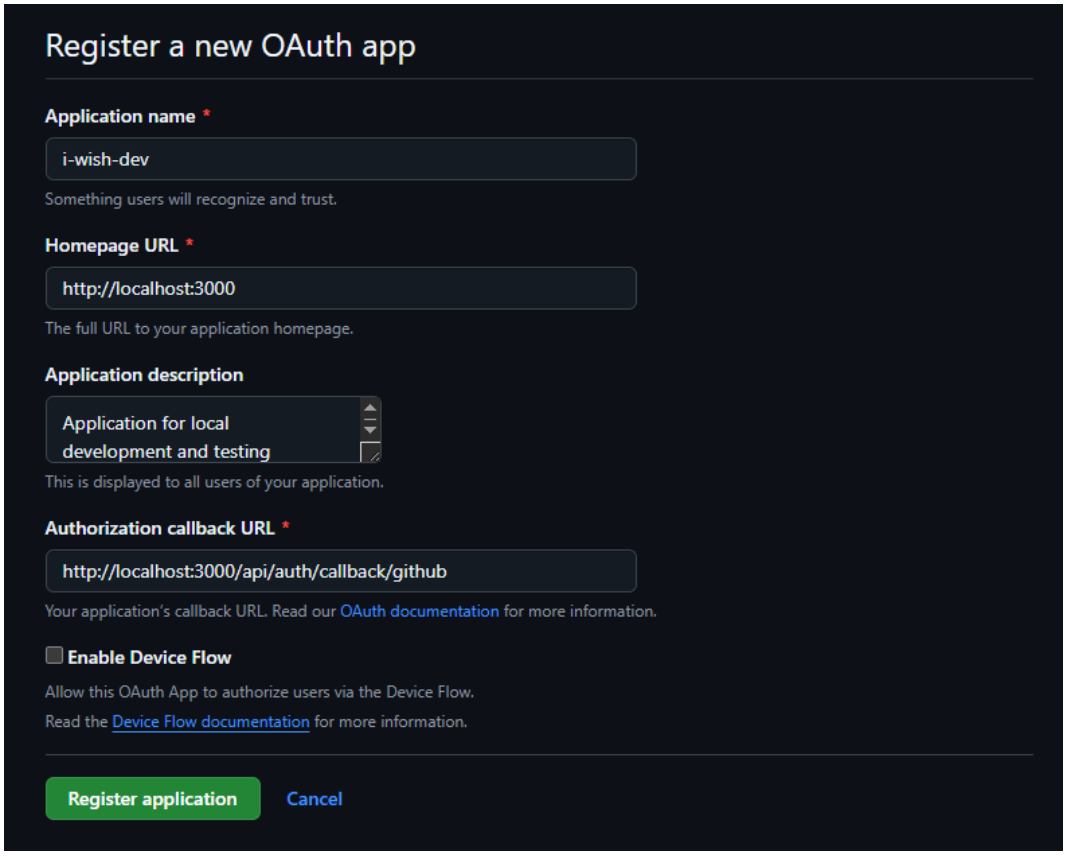

Рисунок 3.4 – Реєстрація застосунку для OAuth-провайдера

Коли користувач підтверджує вхід в провайдері, застосунку відправляються його дані і авторизація стає успішно виконаною. Збереження інформації у базі даних відбувається автоматично за допомогою використання адаптера PrismaAdapter, якому надається клієнтський об’єкт prismaClient. Таким чином Auth.js має змогу сам виконувати записи до бази.

3.4 Робота з переліками бажань

Однією з головних функціональних частин застосунку є управління переліками бажань користувача. Його реалізовано з розрахунком на гнучкість, масштабованість та можливість персоналізації кожного списку

відповідно до індивідуальних потреб користувача. Усі операції реалізовано через взаємодію з базою даних за допомогою Prisma ORM та серверних функцій, а також інтегровано з сучасним React-інтерфейсом із використанням директив «use client» для створення і імплементації інтерактивності.

Користувач має можливість створювати нові списки бажань, кожен з яких має назву, опис і рівень приватності (публічний або приватний). Після натискання на відповідну кнопку (лістинг 3.7), на екрані з’явиться діалогове вікно «WishlistDialog», в якому розташована форма.

Лістинг 3.7 – React-компонент кнопки для створення переліку бажань

```
export function CreateWishlistBtn() {
  const [open, setOpen] = useState(false);
  return (
    <>
      <Button
        onClick={() => {
          setOpen((prev) => !prev);
        }}
      >
        New wishlist
      </Button>
      {open && (
        <WishlistDialog
          wishlist={null}
          onClose={() => setOpen((prev) => !prev)
          }
          onSubmitAction={createWishlistAction}
        />
      )}
    </>
  );
}
```

Підтвердження відправки форми викличе і передасть дані у «createWishlistAction» – серверну дію, що і буде взаємодіяти з базою даних та перевіряти чи користувач авторизований і має право на проведення такої операції (лістинг 3.8). Успішна валідація викличе після себе створення за допомогою Prisma нового запису у таблиці List, автоматично зв’язуючи його з автором за полем userId.

Автоматичне оновлення сторінки здійснюється за допомогою функції revalidatePath, яка очищає збережений кеш для вказано шляху і натомість отримує нові дані з сервера.

Лістинг 3.8 – Серверна дія «createWishlistAction»

```
export async function createWishlistAction(data:
WishlistSchemaType) {
  const session = await auth();
  if (!session?.user?.id) {
    return {success: false, error: "Unauthorized"};
  }
  const { success, data: safeData, error } =
wishlistSchema.safeParse(data);
  if (!success) {
    return {success: false, error: error.toString()};
  }
  try {
    const wishlist = await prisma.list.create({
      data: {
        ...safeData,
        type: "wishlist",
        userId: session.user.id,
      },
    });
    revalidatePath(`/lists`);
    return {success: true, data: wishlist};
  } catch (error) {
   return { success: false, error: `Failed creating wishlist:
${error}` };
  }
}
```

Обробка помилок здійснюється з використанням шаблона для серверних дій. Кожні повертають об’єкт з однаковими полями success та error, що на стороні клієнта використовується для інтерактивного сповіщення користувача (лістинг 3.9) про успіх операції чи її невдачу.

Сервіс Shadcn, у контексті зворотного зв’язку, надає можливість використання React-хука «toast», який відображає малі сповіщення в нижньому куті екрана. До них додається опис та заголовок. Це робить використання застосунку більш інтерактивним.

Лістинг 3.9 – Обробка та відображення зворотнього зв’язку користувачу

```
const response = await onSubmitAction(values,
updateOrCreateEntityId);
if (response.success) {
  toast({
    title: `Wishlist ${isEditing ? "edited" : "created"}
successfully!`,
  });
  onClose();
} else {
  toast({
    title: "Error",
    description: response.error,
    variant: "destructive",
  });
}
```

Генерація сторінки зі списком переліків бажань відбувається майже повністю на сервері, окрім інтерактивного елементу ListMenuDropdown (лістинг 3.10). Всередині компонента використовується директива «use client», тому доступні «onClick» події, після яких відбувається обрані дії.

Лістинг 3.10 – Компонента випадаючого списку ListMenuDropdown

```
export default function ListMenuDropdown({ list }: { list: List
}) {
const [isDialogOpen, setIsDialogOpen] = useState(false);
return (<><DropdownMenuWrapper>
    <DropdownMenuItem onClick={() => setIsDialogOpen((prev) =>
!prev)}>
      <Pencil />
      <span>Edit</span>
    </DropdownMenuItem>
    <ConfirmDialog
  trigger={<DropdownMenuItem className="flex items-center gap-2"
          onSelect={(e) => e.preventDefault()}>
          <Trash2 className="text-rose-500" size={16} />
          <span className="text-rose-500">Delete</span>
        </DropdownMenuItem>}
      description="This action cannot be undone. This will
permanently delete your wishlist."
      confirmText="Delete"
      onConfirm={() => deleteListAction(list.id)}/>
  </DropdownMenuWrapper>
  {isDialogOpen && ( <WishlistDialog wishlist={list}
      onClose={() => setIsDialogOpen(false)}
      onSubmitAction={editWishlistAction}/>)}
</>)}
```

Компонент WishlistDialog розроблений таким чином, щоб використовувати його повторно в будь-якій точці застосунку. Таким чином, передаючи йому різний аргумент «wishlist» та «onSubmitAction», можливо використовувати одну компоненту і для створення і для редагування даних переліку бажань. Серверна подія editWishlistAction проведе валідації та змінить інформацію у базі даних за потреби.

До кожного переліку користувач може додати необмежену кількість бажань. Для цього реалізовано окрему форму, універсальну компоненту WishDialog, яка відповідає як за створення так і за редагування. Вона відкривається у вигляді модального вікна та адаптується залежно від переданих параметрів – якщо передано об’єкт бажання (wish), діалог працює в режимі редагування, інакше – у режимі створення нового бажання.

Інтерфейс компоненти побудований на базі бібліотеки React Hook Form у поєднанні з валідацією через Zod (лістинг 3.11), завдяки чому поля форми мають інтерактивну поведінку у вигляді підказок та повідомлень про помилки. У випадку редагування, значення полів ініціалізуються на основі даних існуючого бажання.

Лістинг 3.11 – Ініціалізація об’єкта форми з React Hook Form

```
const form = useForm<WishSchemaType>(
{
  resolver: zodResolver(wishSchema),
  defaultValues: {
    listId,
    title: wish?.title || "",
    desireLvl: wish?.desireLvl || 1,
    price: wish?.price || 0,
    currency: wish?.currency || "UAH",
    url: wish?.url || "",
    description: wish?.description || "",
    desiredGiftDate: wish.desiredGiftDate ?
      new Date(wish.desiredGiftDate)
      : undefined,
    imageUrl: wish?.imageUrl || DEFAULT_IMAGE_URL,
  },
  mode: "onTouched",
}
);
```

Форма містить усі необхідні поля: назву бажання, рівень пріоритету (від 1 до 5), вартість, валюту, посилання на джерело, дату бажаного отримання подарунка, опис, а також список, до якого належить бажання (лістинг 3.12). Вибір переліку реалізований у вигляді спадного списку і при редагуванні бажання його не можна змінити.

Лістинг 3.12 – Приклад поля форми у WishDialog

```
<FormField
  control={form.control}
  name="title"
  render={({ field }) => (
    <FormItem className="flex flex-col">
      <FormLabel>Title</FormLabel>
      <FormControl>
        <Input {...field} placeholder="The title of the wish."
/>
      </FormControl>
      <FormMessage />
    </FormItem>
  )}/>
```

Якщо дата не потрібна, її можна очистити відповідною кнопкою. Також реалізована функція вставки посилання з буфера обміну в одне натискання.

Кожне бажання також має статус реалізації, який представлений булевим значенням і дозволяє позначати, чи було бажання здійснене. Поле використовується для кращої навігації на сторінці з відображенням всіх бажань переліку: створена можливість відфільтрувати записи і показувати всі, реалізовані або ще не реалізовані.

Після підтвердження відправки форми в WishDialog, дані передаються у відповідний метод (onSubmitAction), який виконує серверну дію, а саме створення або оновлення запису в базі. Після успішного збереження відбувається відображення toast-повідомлення і закриття діалогового вікна.

Також створена можливість сортування бажань за такими полями як ціна та пріоритет (лістинг 3.13). У випадку, коли користувач обирає певну категорію (усі, реалізовані або не реалізовані бажання), компонент оновлює стан фільтру fulfilledFilter, після чого за допомогою useMemo створюється

новий масив, який відповідає обраній умові – або всі, або тільки виконані, або тільки невиконані. Функція useMemo гарантує, що перерахунок масиву буде відбуватися тільки при зміні даних, з яким вона працює, збільшуючи ефективність коду.

Сортування реалізоване подібним чином: зберігається поточний стан сортування (field і order) і при кожному оновленні даних або напрямку сортування відбувається перерахунок за допомогою useMemo. Сортування виконується через вбудовану функцію sort і базується на числовому порівнянні значень у полі, яке відповідає обраному критерію (desireLvl або price). Залежно від напрямку (asc або desc), значення сортуються зростанням або спаданням.

Лістинг 3.13 – Функціональна частина компоненти відображення бажань

```
const [sortConfig, setSortConfig] = useState<{
    field: SortField | null;
    order: "asc" | "desc";
  }>({ field: null, order: "desc" });
const [fulfilledFilter, setFulfilledFilter] =
  useState<FulfilledFilter>("all");
const toggleSort = (field: SortField) => {
  setSortConfig((prev) => ({
    field,
    order: prev.field === field && prev.order === "desc" ? "asc"
: "desc"}
));
};
const filteredWishes = useMemo(() => {
  if (fulfilledFilter === "all") return wishes;
  if (fulfilledFilter === "fulfilled")
    return wishes.filter((w) => w.fulfilled);
  return wishes.filter((w) => !w.fulfilled);
}, [wishes, fulfilledFilter]);
const sortedWishes = useMemo(() => {
  if (!sortConfig.field)
    return filteredWishes;
  return [...filteredWishes].sort((a, b) => {
    if (!sortConfig.field) return 0;
    const diff = a[sortConfig.field] - b[sortConfig.field];
    return sortConfig.order === "asc" ? diff : -diff;
  });
}, [filteredWishes, sortConfig]);
```

У результаті, завжди показуються лише ті бажання, які відповідають поточним умовам фільтра і сортування, та автоматично оновлюється відображення при зміні будь-якого з параметрів.

3.5 Соціальна взаємодія

Для підвищення залученості користувачів і зручності спільного використання переліків бажань в застосунку реалізований механізм соціальної взаємодії, який включає можливість додавання друзів, перегляд їх списків та контроль доступу до вмісту. На окремій сторінці реалізований пошук і відображення друзів, а також показ запитів на становлення дружби.

Процес додавання друзів реалізовано за допомогою моделі Friendship, що описує відправника, отримувача та поточний статус запиту. Користувач має можливість надсилати запит на додавання до друзів, який буде позначено в базі даних із початковим статусом PENDING (лістинг 3.14).

Лістинг 3.14 – Створення запису про запит в друзі та сповіщення адресату

```
const friendship = await prisma.friendship.create({
  data: {
    senderId,
    receiverId,
    status: FriendshipStatus.PENDING,
  },
});
// Create a notification for the receiver
await prisma.notifications.create({
  data: {
    notifiedId: receiverId,
    notifierId: senderId,
    type: NotificationType.FRIEND_REQUEST,
  }});
revalidatePath("/notifications");
revalidatePath("/friends");
```

Інший користувач, відкривши відповідну сторінку друзів, може бачити вхідний запит і прийняти або відхилити його, під час чого створюються записи в базі з відповідною інформацією на основі моделі Notifications.

Після встановлення дружби між користувачами активується можливість перегляду публічних списків іншого користувача. Це реалізовано через перевірку статусу відносин у базі даних – користувач бачить тільки ті списки, які мають значення visibility, що дорівнює public, та належать відповідному другу (лістинг 3.15).

При переході до профілю друга, користувач може ознайомитися зі списками бажань, що доступні для перегляду, переглянути окремі бажання, оцінити пріоритети та побачити деталі кожного з них.

У випадку встановленого значення visibility на private (що робиться за допомогою відповідного елемента меню керування списком) доступ до такого списку має лише його власник. Такий перелік бажань не буде відображатися на сторінці друга.

Окрім контролю на рівні перегляду, реалізовано також обмеження прав на редагування в серверній дії. Змінювати зміст списків або бажань має право виключно їх власник, що досягається шляхом перевірки userId в сесії.

Лістинг 3.15 – Запит в базу даних на відкриті переліки друга

```
const friendship = await prisma.friendship.findFirst({
where: {
  OR: [
    {
      senderId: session.user.id, receiverId: id,
      status: FriendshipStatus.ACCEPTED,
    },
    {
      senderId: id, receiverId: session.user.id,
      status: FriendshipStatus.ACCEPTED,
    },
  ]}
});
if (!friendship) {
  notFound();
}
const list = await prisma.list.findUnique({
  where: { id: listId, userId: id, visibility: "public", type:
"wishlist"},
  include: { user: {
  select: {id: true, name: true, email: true, image: true}
  }, wishes: true}
});
```

Сповіщення відображаються на окремій сторінці «/notifications». Вони можуть нести в собі таку інформацію, як отримання запиту в друзі, підтвердження та відхилення (лістинг 3.16).

Лістинг 3.16 – UI частина компоненти сповіщення

```
<Card
className={cn(
  "transition cursor-pointer hover:bg-accent",
  nft.read ? "opacity-70" : "border-primary border")}
onClick={handleClick}
>
<CardContent className="p-4">
  <div className="flex items-center gap-4">
    <UserAvatar name={nft.notifier.name}
image={nft.notifier.image} />
    <p>{getNotificationMessage(nft)}</p>
  </div>
  <p className="text-xs text-muted-foreground mt-1">
    {new Date(nft.createdAt).toLocaleString("uk-UA")}
  </p>
</CardContent>
</Card>
```

Натискання на сповіщення перенаправляє користувача на сторінку друзів або нових запитів, залежно від типу. В реалізації використано query-параметри, що додаються до шляху через знак «?» і вказують, на яку саме вкладку направити клієнта (лістинг 3.17).

Лістинг 3.17 – Побудова посилання для перенаправлення

```
const tab =
  nft.type === NotificationType.FRIEND_REQUEST
    ? FRIENDS_TABS.REQUESTS
    : FRIENDS_TABS.FRIENDS;
if (nft.read) {
  router.push(`/friends?tab=${tab}`);
  return;
}
const { succes, error } = await markNotificationAsRead(nft.id);
if (!succes || error)
{
  toast({ title: "Error", description: error, variant:
"destructive" });
  return;
}
```

Якщо ж сповіщення позначене як прочитане, новий запит скоріш за все вже закритий, тому перенаправлення йде на головну сторінку друзів.

3.6 Робота з медіа-файлами

Для кожного бажання можливо додати зображення, для чого реалізовано універсальний механізм обробки, завантаження та збереження файлів. Робота з медіафайлами побудована з урахуванням середовища запуску: різна логіка використовується для розробки та для розгорнутого рішення.

Клас FileUploadServiceImpl (лістинг 3.18) абстрагує логіку завантаження файлів та визначає функцію, за допомогою якої зберігатимуться файли.

Лістинг 3.18 – Конструктор класу FileUploadServiceImpl

```
private uploadFn: (
  file: File,
  userId: string,
  bucket: string,
) => Promise<string>;
constructor() {
  const isDev = process.env.NODE_ENV === "development";
  this.uploadFn = isDev ? this.uploadLocal :
this.uploadToSupabase;}
```

У середовищі розробки файли надсилаються на next.js ендпоінт «/api/upload-avatar-local» (лістинг 3.19), де зберігаються на сервері у відповідних директоріях. Після чого клієнту повертається згенероване посилання для доступу до медіа.

Лістинг 3.19 – Функція-обробник локального збереження медіафайлу

```
async function parseForm(req: NextRequest) {
  const formData = await req.formData();
  const file = formData.get("file") as File;
  const userId = formData.get("userId") as string;
  const bucket = formData.get("bucket") as string;
```

```
  const uploadDir = path.join(process.cwd(), "public", bucket);
  if (!file || !userId) {throw new Error("Missing file or
userId")}
  const bytes = await file.arrayBuffer();
  const buffer = Buffer.from(bytes);
  const ext = path.extname(file.name) ||
getExtensionFromMime(file.type);
  const uniqueName = `${crypto.randomUUID()}-
${file.name}${ext}`;
  const userFolder = path.join(uploadDir, userId);
  const filePath = path.join(userFolder, uniqueName);
  fs.mkdirSync(userFolder, { recursive: true });
  fs.writeFileSync(filePath, buffer);
  const publicPath = `/${bucket}/${userId}/${uniqueName}`;
  return { path: publicPath };
}
```

У продакшен-середовищі зображення завантажуються безпосередньо в хмірне сховище Supabase storage. Кожне зображення зберігається за унікальним шляхом у форматі «${userId}/${uuid}-${filename}» (лістинг 3.20). Під час завантаження застосовується обмеження на максимальну ширину або висоту зображення (512px) для уникнення надлишкової ваги файлу, а також встановлюється cacheControl і на стороні клієнта кешується медіавміст.

Лістинг 3.20 – Функція обробник продакшен збереження медіафайлу

```
private async uploadToSupabase(file: File,userId: string,bucket:
string,): Promise<string> {
  const path = `${userId}/${crypto.randomUUID()}-${file.name}`;
  const options = {
    cacheControl: "3600",
    maxWidthOrHeight: 512,
    upsert: true,
  };
  const { data, error } = await this.getStorage()
    .from(bucket)
    .upload(path, file, options);
  if (error) throw error;
  return getPublicStorageUrl(bucket, data.path);
}
```

Компресія зображень виконується безпосередньо на клієнті перед їх завантаженням на сервер. Цей процес для бажань реалізовано через спеціалізовану компоненту ImageUploader, яка приймає файл від

користувача, стискає його до оптимального розміру, створює попередній перегляд і передає результат у батьківську компоненту форми перед збереженням змін.

Для зменшення розміру зображення використовується бібліотека browser-image-compression. Обсяг файлу обмежується до 1MB і встановлюється максимальні розміри в 512px по ширині або висоті. Після успішного стискання зображення передається у вигляді File-об’єкта через функцію зворотного виклику onCompressed, а також генерується його попередній перегляд (лістинг 3.21) через FileReader.

Лістинг 3.21 – Обробка події вибору файлу

```
const handleImageChange = async (e:
React.ChangeEvent<HTMLInputElement>) => {
  const file = e.target.files?.[0];
  if (!file) return;
  setImageName(file.name);
  setIsCompressing(true);
  const compressed = await imageCompression(file, {
    maxSizeMB: 1,
    maxWidthOrHeight: 512,
    useWebWorker: true,
  });
  onCompressed(compressed);
  const reader = new FileReader();
  reader.onloadend = () => { setPreviewImage(reader.result as
string)};
  reader.readAsDataURL(compressed);
setIsCompressing(false);
```

Інтерфейс ImageUploader дозволяє переглядати вибране зображення до його завантаження, видаляти його за потреби, а також відображає статус обробки. Якщо зображення не обране, користувач бачить запрошення до завантаження у вигляді області з пунктирною рамкою. У випадку помилки стиснення виводиться відповідне повідомлення через toast-сповіщення.

Після завершення компресії зображення і надсилання форми для зміни даних, посилання зберігається в базі даних в полі imageUrl відповідного бажання або аватара.

4 ІНСТРУКЦІЯ КОРИСТУВАЧА

4.1 Розгортання та CI/CD

Розгортання застосунку реалізоване із застосуванням принципів безперервної інтеграції та доставки (CI/CD). Локальне середовище побудоване на базі Docker та забезпечує ізольовану та автоматизовану інфраструктуру для розробки. За налаштування цього відповідає конфігураційний файл docker-compose.yml (лістинг 4.1), який визначає два основні сервіси: app та postgres-db.

Лістинг 4.1 – Вміст файлу docker-compose.yml

```
services:
  app:
    build:
      context: .
      dockerfile: Dockerfile.dev
    container_name: iwish-app-dev
    ports: - "3000:3000"
    env_file: - .env
    environment: - DATABASE_URL=
postgresql://postgres:admin@postgres-db:5432/iwish
    volumes:
      - .:/app
      - /app/node_modules
    depends_on: - postgres-db
    command: >
      sh -c "npx prisma generate && npx prisma migrate dev &&
npm run dev"
  postgres-db:
    image: postgres:15
    container_name: iwish-db
    restart: always
    env_file: - .env
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: admin
      POSTGRES_DB: iwish
    ports: - "5432:5432"
    volumes: - postgres_data:/var/lib/postgresql/data
volumes:
  postgres_data:
```

Сервіс app представляє собою контейнер із Next.js застосунком. Під час запуску контейнер виконує генерацію Prisma-клієнта, застосовує останні міграції (npx prisma migrate dev) до бази даних і запускає сервер у режимі розробки. Сервіс postgres-db піднімає контейнер з PostgreSQL, у якому визначено базову конфігурацію користувача, пароля та назви бази даних. Дані бази зберігаються у volume postgres_data, дозволяє уникнути втрати інформації після перезапусків контейнера.

Синхронізація бази даних між середовищами розробки та продакшену забезпечується за допомогою Prisma міграцій. Усі структурні зміни фіксуються в окремих міграційних файлах і можуть бути застосовані як до локальної, так і до хмарної БД. Після виконання команди «npx prisma migrate dev» ці зміни застосовуються автоматично при запуску локального середовища. У хмарному середовищі замість цього використовується «npx prisma migrate deploy», яка безпечно застосовує міграції до вже працюючої бази даних.

Для первинного заповнення таблиць у режимі розробки використовується Prisma Seed – окрема команда, що виконує файл prisma/seed.ts з логікою ініціалізації даних.

Для розгортання в продакшені використовується платформа Vercel. Вона автоматично інтегрується з GitHub-репозиторієм і виконує побудову застосунку після кожного пушу в основну гілку. Весь процес деплою відбувається без потреби ручного втручання: запускається команда «npm run build», і, за потреби, викликається виконання міграцій. Змінні середовища, такі як DATABASE_URL, SUPABASE_URL, NEXTAUTH_SECRET та інші, передаються через інтерфейс керування в панелі Vercel.

Запуск розгорнутого рішення доступний за посиланням: https://iwish-nextjs.vercel.app/.

Для локального розгортання застосунку необхідно мати встановлені Docker та Docker Compose. Після клонування репозиторію з сервісу Github потрібно створити файл .env у корені проєкту та заповнити його

відповідними змінними середовища, зокрема DATABASE_URL, яка повинна відповідати шаблону підключення до PostgreSQL, та AUTH_SECRET, яка генерується автоматично після виконання команди «npm exec auth secret».

Після налаштування середовища необхідно виконати команду «docker-compose up», яка підніме два контейнери – застосунок і базу даних. На етапі запуску всередині контейнера виконується генерація Prisma-клієнта, застосування актуальних міграцій до бази та запуск сервера у режимі розробки. Після завершення ініціалізації застосунок буде доступний за адресою «http://localhost:3000».

Якщо база даних піднімається вперше, застосунок автоматично створить усі необхідні таблиці відповідно до визначеної схеми, і виконає початкове заповнення тестовими даними. Усі зміни, які розробник вносить у код, будуть відображатися в реальному часі, оскільки застосунок працює в режимі watch з монтованою локальною файловою системою в контейнер.

4.2 Реєстрація та авторизація

Перший запуск застосунку на будь-якому шляху перенаправить користувача на сторінку авторизації (рисунок 4.1). У нього буде можливість увійти за допомогою email та пароля, або Github.

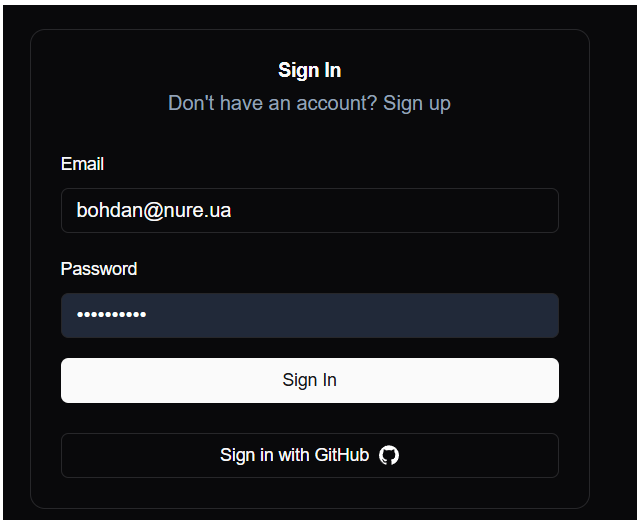

Рисунок 4.1 – Сторінка авторизації

Під час введення даних біля полів буде з’являтися повідомлення підказки з інформацією про довжину паролю або важливістю вказання email.

Безпарольна авторизація за допомогою Github перенаправить користувача на окрему сторінку, де запросить дозвіл на використання персональних даних (фото профілю, email, ім’я користувача). Підтвердження запиту поверне клієнта на головну сторінку застосунку, він стане авторизованим.

Якщо користувач ще не має акаунту, його можна створити перейшовши на сторінку реєстрації (рисунок 4.2), натиснувши на «Don’t have an account? Sign up». Необхідно заповнити 3 поля: ім’я, email та пароль, який має бути мінімум з 6 символів. При реєстрації з вказанням вже існуючого email, клієнт побачить toast-сповіщення, яке вкаже на помилку.

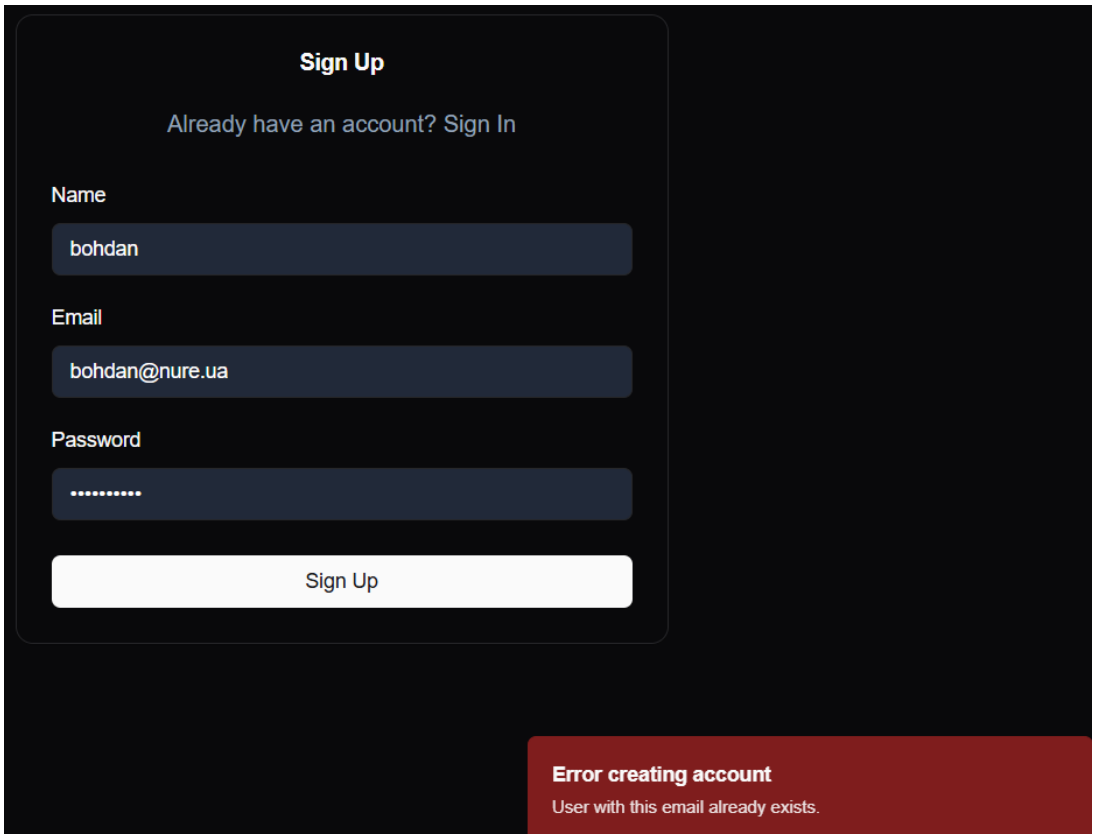

Рисунок 4.2 – Реєстрація користувача з вже існуючим email

Якщо введені дані коректні і користувача з такою поштою ще не існує – акаунт створиться і юзера переведе на сторінку авторизації.

4.3 Зміна даних акаунта

Головна сторінка застосунку після авторизації відображатиме відкрите бічне меню, знизу якого можна побачити власні аватар, ім’я та пошту. Натиснувши на панель, відкриється меню з деякими налаштуваннями акаунта (рисунок 4.3). Елемент «Log out» виведе користувача з акаунта і перенаправить на сторінку авторизації. «Notifications» переводить на сторінку з налаштуваннями.

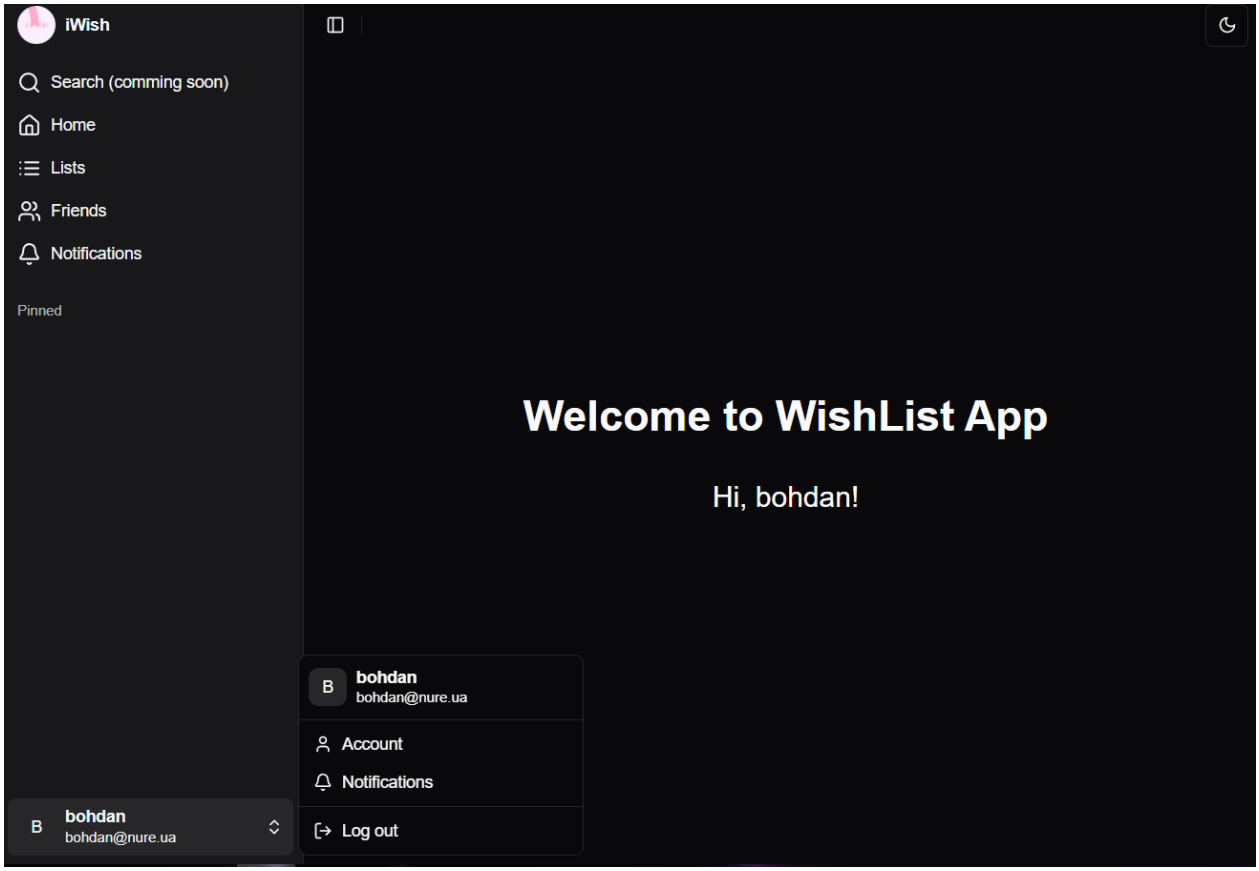

Рисунок 4.3 – Головна сторінка застосунку

Елемент меню «Account» відкриє модальне вікно (рисунок 4.4), в якому головні дані акаунта можливо буде оновити. Зміна електронної пошти буде означати, що під час авторизації необхідно буде використовувати оновлену версію email. Якщо акаунт з обраною поштою вже існує, користувач побачить відповідне повідомлення.

Натиснувши на значок камери, перед користувачем з’явиться вікно, в якому він зможе обрати фотографію, що буде встановлена як аватар. Дозволений тип файлу відповідає jpg, png, svg або webp.

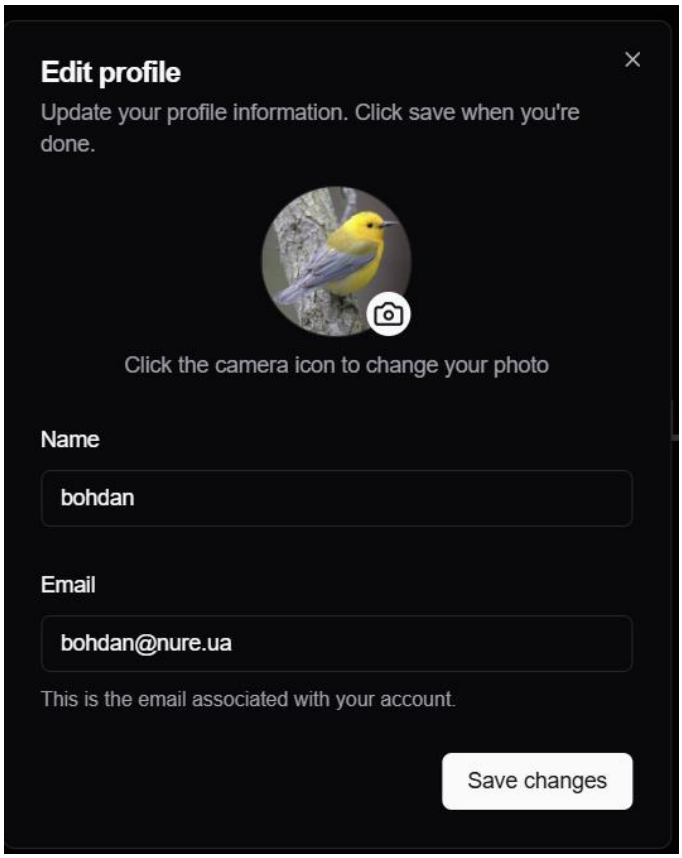

Рисунок 4.4 – Модальне вікно редагування акаунта

Вибір картинки перед збереженням змін можна буде побачити у передперегляді, у вигляді кола, а успішне збереження даних закриє діалогове вікно та відобразить toast-сповіщення (рисунок 4.5) з відповідним повідомленням.

Рисунок 4.5 – Сповіщення та вигляд оновленої аватарки

Аватар в бічній панелі буде автоматично оновлений без перезавантаження сторінки.

4.4 Створення та редагування переліку бажань

Перейшовши у вкладку Lists можна буде побачити список власних переліків і кнопки для їх створення і для створення бажань. Після створення нового акаунта, сторінка (рисунок 4.6) матиме 1 список під назвою «My Wishlist». Його можна буде редагувати, обравши відповідний елемент меню, але видалити неможливо, бо, для кращої організації даних, акаунт має мати хочаб один перелік бажань.

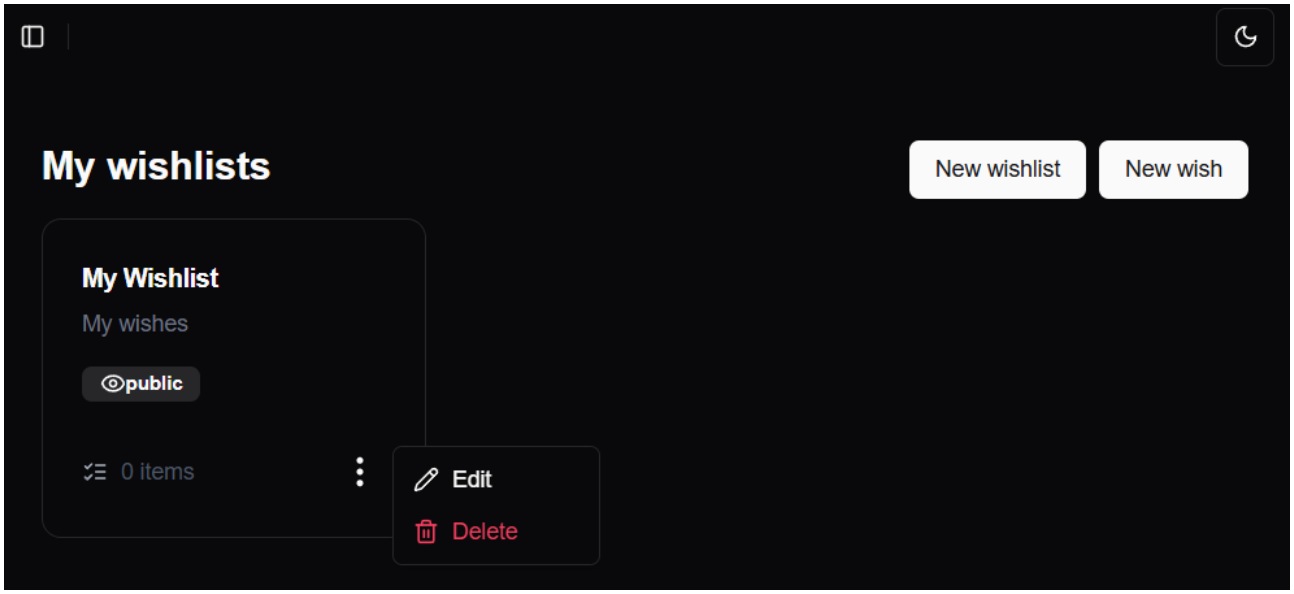

Рисунок 4.6 – Сторінка переліків бажань

На сторінці для кожного списку можна побачити тип його видимості, опис та кількість елементів всередині.

Для створення нового переліку бажань необхідно обрати кнопку «New wishlist». Клік по кнопці відкриє модальне вікно з полями для вказання назви, опису та видимості (рисунок 4.7). Натискання на кнопку «Create» створює відповідні записи в базі даних, закриває модальне вікно та сповіщає користувача про успішність операції. Новий список автоматично з’являється на сторінці без необхідності оновлення.

На етапі, коли існує 2 переліка бажань чи більше, будь-який можна видалити.

Поле з назвою є обов’язковим, тому не можна залишити його пустим, але опис додається за бажанням.

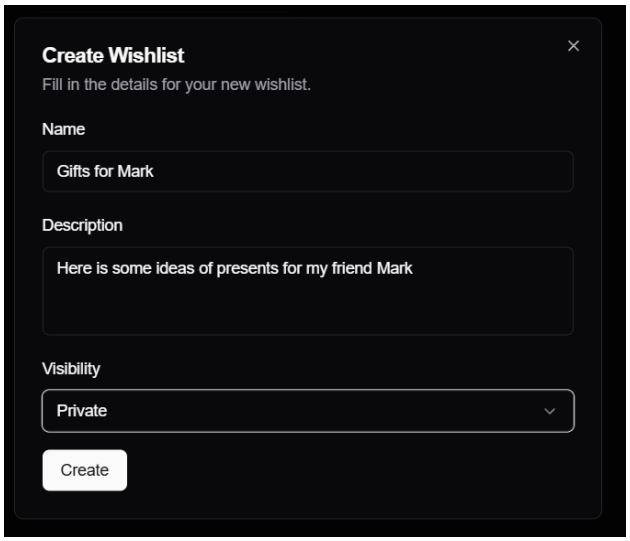

Рисунок 4.7 – Модальне вікно створення/редагування переліку

Іконки видимості поруч з текстом «public» та «private» на переліку відрізняються: це око та закреслене око відповідно.

4.5 Створення і редагування бажання

Бажання можна створити на сторінці Lists або на сторінці вибраного вішліста (рисунок 4.8) натиснувши на кнопку «New wish».

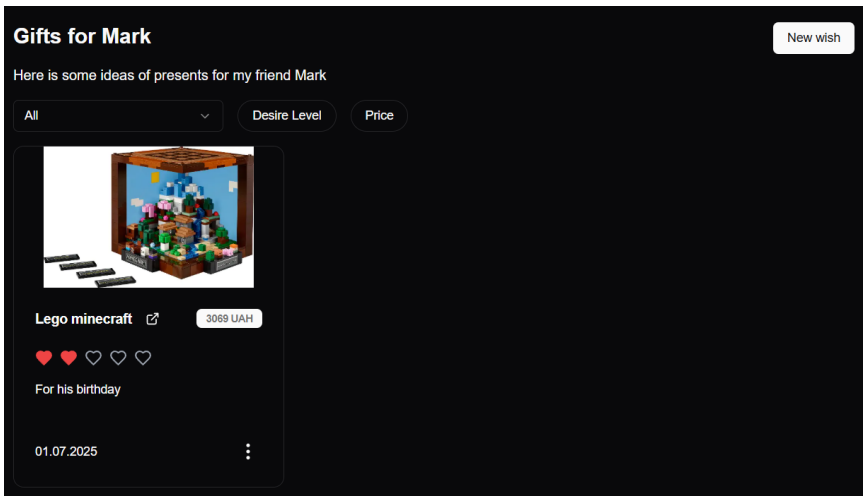

Рисунок 4.8 – Сторінка обраного вішліста

Після цього відкриється модальне вікно з формою (рисунок 4.9), де можна ввести всю необхідну інформацію. У першу чергу потрібно обрати список, до якого належить бажання, вказати його назву (обов’язкове поле), бажаний рівень пріоритету (від 1 до 5), ціну, валюту, опис, посилання на джерело та дату, коли бажано отримати подарунок.

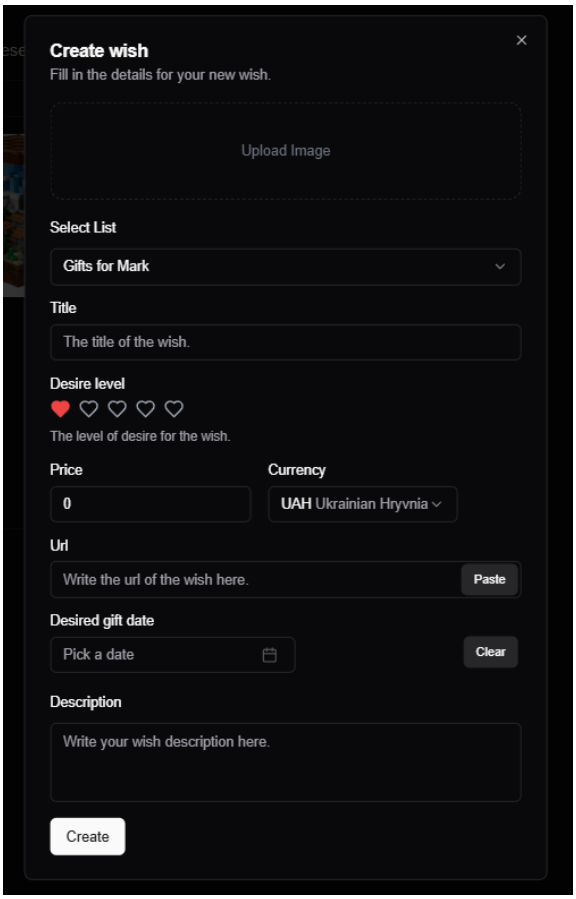

Рисунок 4.9 – Модальне вікно створення бажання

Обробка натискання кнопки «Paste» в кінці поля для вставки url автоматично використає з буферу обміну скопійований текст і впише на місце посилання. Кнопка «Clear» біля поля «Desired gift date» очистить встановлене значення.

Окремим кроком є завантаження зображення. Для цього достатньо обрати файл зі свого пристрою. Обране зображення стискається до оптимального розміру, щоб пришвидшити його завантаження на сервер, а після цього фото додається до форми та буде відправлене на сервер разом з

усіма іншими полями під час збереження даних. Також можливо видалити обране зображення за допомогою відповідної кнопки прямо біля поля. Поки форма не була відправлена, користувач може бачити назву файлу, який він обрав для завантаження.

Завдяки встановленню значень в такі поля як ціна та рівень бажаності, на сторінці з бажаннями певного переліку можливо відсортувати дані, як по зростанню так і по спаданню (рисунок 4.10). Наприклад, натиснувши 1 раз по кнопці «Desire Level» всі бажання будуть відображатися по спаданню значення desireLvl.

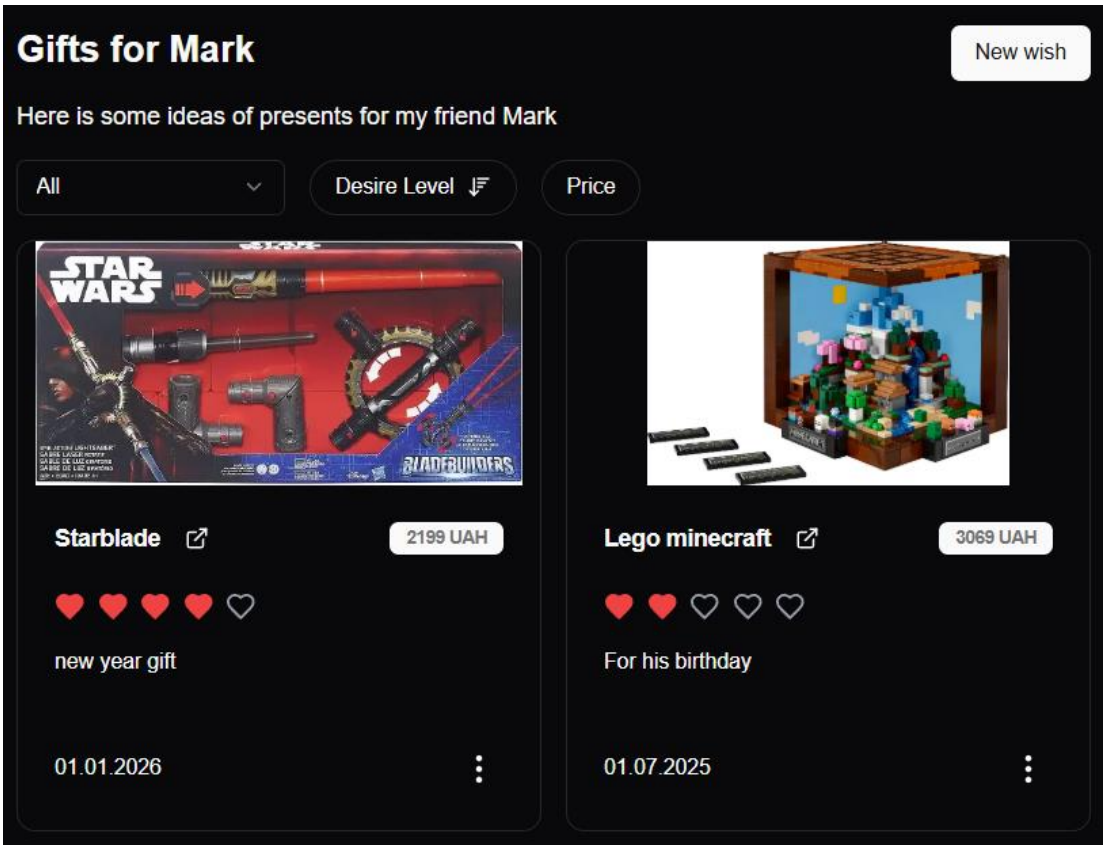

Рисунок 4.10 – Відсортовані елементи переліку за рівнем бажаності

Фільтрація бажань доступна за значенням реалізації. В їх меню доступне поле «Mark as fulfilled». Вибір цього пункта запрошує підтвердження дії за допомогою модального вікна і потім позначає обране бажання як реалізоване, що буде відображатися у вигляді зеленої галочки (рисунок 4.11) у верхньому лівому куті картки.

Після чого «реалізовані» бажання можна побачити або в загальному списку, або використавши фільтрацію «Fulfilled».

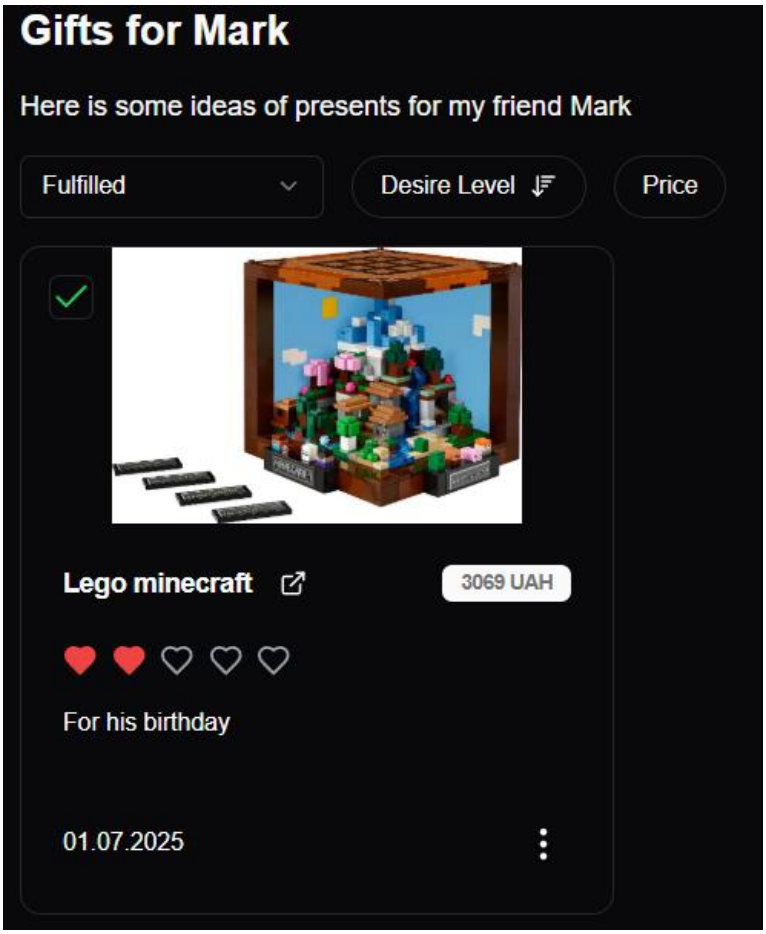

Рисунок 4.11 – Відображення реалізованих бажань

Одночасно можливо обирати як налаштування фільтрації, так і сортування. Це допомагає легше орієнтуватися, якщо перелік має велику кількість бажань.

4.6 Створення запиту на дружбу

Щоб додати друга, користувач може скористатися функцією пошуку, яка доступна на сторінці друзів. Для цього потрібно ввести ім’я або електронну адресу в пошукове поле й натиснути кнопку пошуку. Після цього система відобразить список користувачів, що відповідають введеному запиту

(рисунок 4.12). Навпроти кожного знайденого користувача буде доступна кнопка «Add», яка дозволяє надіслати запит на дружбу. Якщо на неї натиснути, вона зміниться на «Pending», що означатиме успіх.

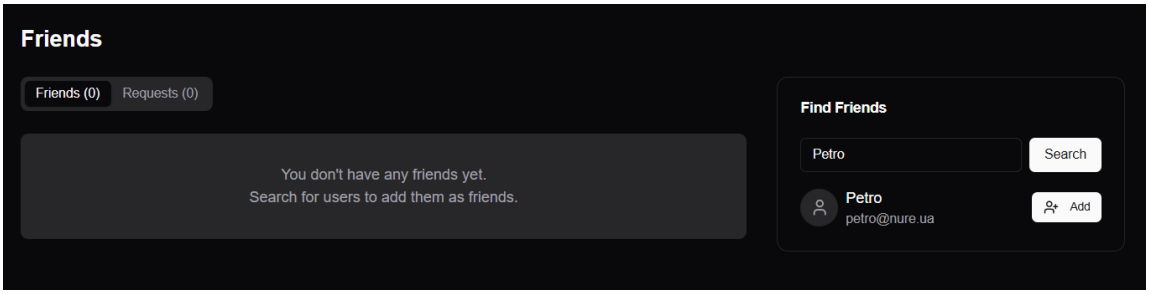

Рисунок 4.12 – Пошук користувача на сторінці друзів

Коли адресат отримує запит, він також отримує сповіщення (рисунок 4.13). Воно містить інформацію про подію, наприклад, «Запит у друзі», і при натисканні перенаправляє користувача на сторінку друзів, де відкривається відповідна вкладка зі списком вхідних запитів чи підтверджених друзів.

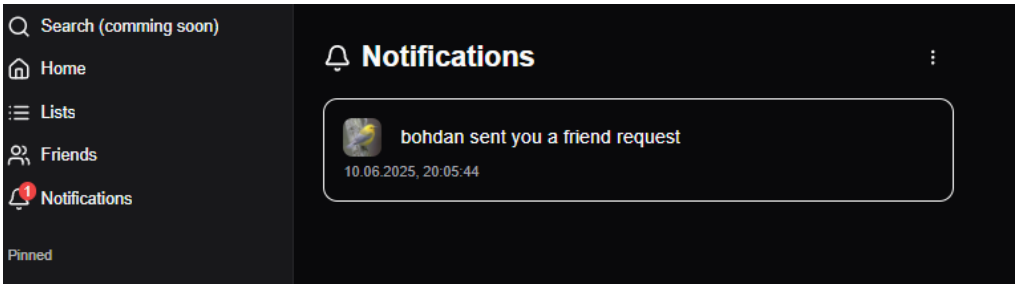

Рисунок 4.13 – Сторінка зі сповіщеннями

Коли користувач відкриває вкладку з запитами, він бачить усі вхідні запити та може або прийняти, або відхилити їх. Після підтвердження статус змінюється на «Друзі», і обидва користувачі отримують можливість бачити публічні списки бажань один одного. Усі дії супроводжуються сповіщеннями про успіх або помилку, а саме: «Запит надіслано», «Запит прийнято» або «Запит відхилено».

ВИСНОВКИ

Робота присвячена розробці програмного забезпечення для створення, зберігання та організації переліків бажань. Актуальність теми зумовлена відсутністю на ринку простого, інтуїтивного і водночас функціонального інструменту, орієнтованого саме на ведення списків бажань.

У межах дослідження проведено аналіз наявних сервісів та платформ, таких як Listy, Wisher, Rewish і Notion, та виявлено їхні основні недоліки: відсутність структури, нестачу приватності, перевантаженість інтерфейсу або недостатню адаптацію під потреби користувача.

Проєкт реалізовано на базі фреймворку Next.js з використанням App Router, що забезпечило гнучку маршрутизацію та зручне розділення клієнтських і серверних компонентів. Для оформлення інтерфейсу використано Tailwind CSS у поєднанні з платформою Shadcn/ui, і таким чином створено адаптивний і сучасний дизайн без надмірної складності. Система автентифікації реалізована з використанням Auth.js та JWT, що забезпечило безпечний доступ до облікових записів, а також гнучкий контроль сесій користувача через cookie.

У якості хмарної інфраструктури для розгортання готового рішення використано Supabase як базу даних і сховище файлів, а для деплою фронтенду – Vercel. Використано також механізм контейнеризації завдяки Docker, який дозволяє проводити локальну розробку програмного забезпечення без втручання до хмарних даних.

Результати роботи були представлені у рамках XXIX Міжнародного молодіжного форуму [30] «Радіоелектроніка і молодь в XXI столітті».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Jones W., Teevan J. Personal Information Management. University of Washington Press, 2007. 340 p. ISBN: 9780295987378

2. Barreau D., Nardi B. A. Finding and reminding: File organization from the desktop. *ACM SIGCHI Bulletin*. 1995. Vol. 27, No. 3. P. 39-43. DOI: https://doi.org/10.1145/221296.221307

3. Bonneau J., Herley C., Van Oorschot P. C., Stajano F. The quest to replace passwords: A framework for comparative evaluation of Web authentication schemes. *IEEE Symposium on Security and Privacy*. 2012. P. 553-567. DOI: https://doi.org/10.1109/SP.2012.44

4. Odom W., Sellen A., Harper R., Thereska E. Lost in translation: Understanding the possession of digital things in the cloud. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2012. P. 781-790. DOI: https://doi.org/10.1145/2207676.2207789

5. Listy. URL: https://listy.is/ (дата звернення: 22.05.2025).

6. Thelwall M., Kousha K. Goodreads: A social network site for book readers. *Journal of the Association for Information Science and Technology*. 2017. Vol. 68, No. 4. P. 972-983. DOI: https://doi.org/10.1002/asi.23733

7. Wisher: лист бажань та мрій. URL: https://play.google.com/store/apps/details?id=com.wisherandgiftapp (дата звернення 22.05.2025).

8. Create your own wishlist, fill collections, and share with friends around the world. URL: https://rewish.io/welcome (дата звернення 22.05.2025).

9. Notion: Your connected workspace for wiki, docs & projects. URL: https://www.notion.com/ (дата звернення 22.05.2025).

10. Your code editor. Redefined with AI. URL: https://code.visualstudio.com/ (дата звернення 22.05.2025).

11. Pipinellis A. GitHub Essentials: Unleash the power of collaborative

development workflows using GitHub, 2nd Edition. Packt Publishing Ltd, 2018. 178 p. ISBN: 9781789138467

12. Lazuardy M. F. S., Anggraini D. Modern front end web architectures with React.js and Next.js. *Research Journal of Advanced Engineering and Science*. 2022. Vol. 7, No. 1. P. 132-141.

13. Jain S., Dony M. Modern Web Applications with Next.JS: Learn Advanced Techniques to Build and Deploy Modern, Scalable and Production Ready React Applications with Next.JS. Orange Education Pvt Limited, 2023. 374 p. ISBN: 9789388590983

14. Riva M. Real-World Next.js: Build scalable, high-performance, and modern web applications using Next.js, the React framework for production. Packt Publishing Ltd, 2022. 366 p. ISBN: 9781801079877

15. Tailwind CSS. URL: https://tailwindcss.com/ (дата звернення 22.05.202).

16. Shadcn UI. URL: https://ui.shadcn.com/ (дата звернення 22.05.2025).

17. Rajala O. Impact of React component libraries on developer experience – An empirical study on component libraries' styling approaches. 2024. DOI: http://doi.org/10.13140/RG.2.2.18504.33282

18. Ahmed S., Mahmood Q. An authentication based scheme for applications using JSON web token. *International Multitopic Conference*. IEEE, 2019. P. 1-6. DOI: https://doi.org/10.1109/INMIC48123.2019.9022766

19. Cahn A., Alfeld S., Darford P., Muthukrishnan S. An empirical study of web cookies. *25th International Conference on World Wide Web*. 2016. P. 891-901. DOI: https://doi.org/10.1145/2872427.2882991

20. Auth.js: Session Management. URL: https://authjs.dev/getting-started/session-management/protecting. (дата звернення 22.05.2025).

21. Prisma. URL: https://www.prisma.io/docs/orm/overview/introduction (дата звернення 22.05.2025).

22. Sivakumar V., Balachander T., Logu, Jannali R. Object relational mapping framework performance impact. *Turkish Journal of Computer and*

Mathematics Education. 2021. Vol. 12, No. 7. P. 2516-2519.

23. Sampath H., Merrick A., MacVean A. Accessibility of command line interfaces. *CHI Conference on Human Factors in Computing Systems*. 2021. P. 1–10. DOI: https://doi.org/10.1145/3411764.3445544

24. Le Q. H., Diaz M. Developing Modern Database Applications with PostgreSQL. Packt Publishing, 2021. 440 p.

25. PgAdmin. URL: https://www.pgadmin.org/ (дата звернення 22.05.2025).

26. Miell I., Sayers A. Docker in Practice. Simon and Schuster, 2019. 372 p. ISBN: 9781617292729

27. What is Docker? URL: https://www.ibm.com/think/topics/docker (дата звернення 22.05.2025).

28. Vercel. URL: https://vercel.com/home (дата звернення 22.05.2025).

29. Upadhyaya N. Deployment and Continuous Integration / In: Advanced Front-End Development: Building Scalable and High-Performance Web Applications with React. Berkeley, CA : Apress, 2025. P. 317-335. DOI: https://doi.org/10.1007/979-8-8688-1318-4

30. Рогозянський Б. Ю., Іващенко Г. С. Автоматизація ведення переліків бажань. *Радіоелектроніка та молодь у XXI столітті*. 2025. Т. 2, №2. С. 19.