Міністерство освіти і науки України

Луцький національний технічний університет

Факультет комп'ютерних та інформаційних технологій

Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ВЕБ-СЕРВІСУ З МЕНЕДЖМЕНТУ ОНЛАЙН ПІДПИСОК

DEVELOPMENT OF A WEB SERVICE WITH MANAGEMENT OF ONLINE SUBSCRIPTIONS

спеціальність 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

освітня програма «Інженерія програмного забезпечення»
(назва освітньої програми)

Виконав: здобувач вищої освіти
Групи ІПЗс-31
Нікішин Володимир Олександрович
_____
(підпис)

Керівник:
к.т.н., доцент
Ліщина Наталія Миколаївна
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«8» 06 2024 р.
к.т.н., доцент
Гарант освітньої програми:
Ліщина Наталія Миколаївна
_____
(підпис)

Луцьк – 2024 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *121 Інженерія програмного забезпечення*
Освітня програма: *«Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Н. Ліщина
«2» 02 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Нікішину Володимиру Олександровичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: *«Розробка веб-сервісу з менеджменту онлайн підписок»*

Керівник роботи: *к.т.н., доцент, Ліщина Наталія Миколаївна*
затверджені наказом закладу вищої освіти від «*30*» *грудня 2023 р.* № *477/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи *«5» червня 2024 р.*
3. Вихідні дані до роботи: *технічне та програмне забезпечення ЕОМ, вимоги до розробки програмного забезпечення, ергономічні вимоги до функціонування програмного засобу.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити):
Аналіз сучасного стану проблеми, існуючих методів і засобів її розв'язання, аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення, опис функціонального наповнення об'єкта проектування, практична реалізація об'єкта проектування.

5. Перелік графічного матеріалу: *34 рисунків; 6 лістингів; 2 діаграми.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Ліщина Н. М.*		
Специфікації вимог до розробленої системи	*Ліщина Н. М.*		
Розробка об'єкта проєктування	*Ліщина Н. М.*		
Нормоконтроль	*Повстяна Ю. С.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту		5,1%	
Академічна доброчесність	*Яцук А. А.*		

7. Дата видачі завдання *«2» лютого 2024 р.*

КАЛЕНДАРНИЙ ПЛАН

№ З/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	*Огляд літературних джерел по темі кваліфікаційної роботи бакалавра*	*07.02.2024 р.*	*викон.*
2	*Аналіз проблеми розробки та впровадження об'єкту проектування*	*27.02.2024 р.*	*викон.*
3	*Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання*	*12.03.2024 р.*	*викон.*
4	*Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних*	*17.04.2024 р.*	*викон.*
5	*Практична реалізація об'єкта проектування та розробка бази даних*	*18.05.2024 р.*	*викон.*
6	*Тестування та налагодження об'єкта проектування*	*01.06.2024 р.*	*викон.*
7	*Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру*	*05.06.2024 р.*	*викон.*

Здобувач вищої освіти _____ (підпис) *Нікішин В. О.* (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ (підпис) *Ліщина Н. М.* (прізвище, ініціали)

Міністерство освіти і науки України
Луцький національний технічний університет

Рецензія

на кваліфікаційну роботу

Здобувач вищої освіти: Нікішин Володимир Олександрович

Тема: «Розробка веб-сервісу з менеджменту онлайн підписок»

Коротка характеристика кваліфікаційної роботи: Робота магістра присвячена розробці веб-сервісу з використанням Nest.js та Next.js, що забезпечує зменшення зусиль розробників та витрат часу на обслуговування проекту.

Самостійні розробки і пропозиції автора: У роботі узагальнено методи розробки веб-сервісу з використанням Nest.js та Next.js підходу, а також доведено ефективність запропонованих технологій та алгоритмів.

Практичне значення роботи: Запропоновану в роботі методику успішно застосовано для обслуговування роботи веб-сервісу. Кваліфікаційна робота має практичне значення, апробована на міжнародній науково-практичній конференції.

Недоліки: не знайдено.

Загальний висновок: Робота виконана на високому науковому рівні, має логічну та послідовну структуру, відповідає встановленим вимогам, може бути представлена до захисту на державній екзаменаційній комісії та заслуговує позитивної оцінки.

Рецензент: Кабак В.В., дав. каф. УОТ
(прізвище, ім'я, по-батькові, посада)

Рецензент кваліфікаційної роботи _____
(підпис)

«07» 06 2024 р.

Міністерство освіти і науки України
Луцький національний технічний університет

Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

ВІДГУК
КЕРІВНИКА НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА
Здобувач вищої освіти *Нікішин Володимир Олександрович*
група *ІПЗс-31*

Тема кваліфікаційної роботи: *Розробка веб-сервісу з менеджменту онлайн підписок*

Керівник *Ліщина Наталія Миколаївна*

Актуальність теми: Створення веб-сервісів з використанням Next.js та Nest.js залишається важливим напр'ямком у галузі програмування. Хоча наявні розробки вже вирішили деякі завдання в цій сфері, все ще є місце для вдосконалення. Зокрема, потребують уваги такі аспекти, як поліпшення взаємодії з користувачами, розширення можливостей інтеграції та адаптація до різноманітних сфер послуг.

Об'єкт дослідження – це комплекс заходів з проектування та кінцевий продукт створення веб-сервісу.

Характеристика теоретичного рівня, наявності самостійних розробок і практичної значимості роботи: у рамках цієї роботи було розроблено ефективний веб-сервіс на базі фреймворку Next.js та Nest.js, що демонструє високий теоретичний рівень та практичну значущість. Система відзначається зручним інтерфейсом управління даними користувача.

Зауваження та недоліки: істотних недоліків не виявлено.

Загальний висновок робота виконана на хорошому рівні та заслуговує оцінки «відмінно» за умови успішного захисту.

Керівник _____ (_____)
(підпис) (прізвище, ім'я, по батькові)

«7» 06 2024 р.

АНОТАЦІЯ

Нікішин В. О. Розробка веб-сервісу з менеджменту онлайн підписок Рукопис.

Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2024.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі здійснено аналіз сучасного стану проблеми розробки веб-сервісу з менеджменту онлайн підписок і сформульовано завдання. В другому розділі визначено вимоги до розроблюваної системи, а також засоби, методи і алгоритми розробки. У третьому розділі розроблено повністю функціонуючий веб-сервіс з менеджменту онлайн підписок. У висновках узагальнено інформацію, відображену у попередніх частинах. Результати розробки демонструють можливості роботи веб-сервісу.

Ключові слова: розробка, менеджмент, онлайн підписка, діаграма, вимоги.

ABSTRACT

Nikishin V. O. Development of a web service for managing online subscriptions Manuscript.

Bachelor’s qualifying thesis of the OP "Software Engineering". Lutsk National Technical University. Lutsk, 2024.

The bachelor’s qualification work consists of an introduction, three sections, conclusions, a list of used sources and appendices.

In the first section, an analysis of the current state of the problem of developing a web service for managing online subscriptions was carried out and the task was formulated. The second section defines the requirements for the developed system, as well as means, methods and development algorithms. In the third section, a fully functioning online subscription management web service is developed. The conclusions summarize the information presented in the previous parts. The development results demonstrate the capabilities of the web service.

Keywords: development, management, online subscription, diagram, requirements.

ЗМІСТ

ВСТУП

Актуальність теми. В сучасному світі швидкий розвиток інформаційних технологій спричинив значне зростання популярності веб-сервісів, таких як Spotify, Megogo, Netflix та програмного забезпечення, таких як Microsoft Word, Adobe, що надають свої послуги за підпискою. Тобто користувачу потрібно регулярно платити кошти. Інколи це раз в місяць, інколи раз в півроку, а буває взагалі раз на два тижні. Такий підхід дозволяє користувачам отримувати регулярний доступ до контенту та сервісів, а компаніям – стабільний прибуток і можливість гнучкого управління своїми ресурсами.

Однак із збільшенням кількості різних веб-сервісів та підписок в користувача виникає потреба в ефективних інструментах для їхнього управління та менеджменту, які б забезпечували зручне відслідковування в яких саме веб-сервісах користувач має місячну підписку, скільки коштів загалом користувач тратить на всі його підписки, та функціонал нагадування про списання коштів у користувача сервісом, бо часто буває, що користувач, придбав місячний план на якийсь сервіс, трохи покористувався ним і забув за нього, і кошти у користувача спише, але користувач не хотів би цього, тому для цього потрібне нагадування про те що скоро буде оплата підписки. Тому виходячи з актуальності тема, розробка веб-сервісу для менеджменту онлайн підписок є актуальною, оскільки надає зручний інструмент який помагає користувачам слідкувати за власними підписками та нагадувати про їх оплату.

Мета кваліфікаційної роботи – розробити сучасний та функціональний веб-сервіс який буде зручний в користуванні та буде нагадувати користувачам про оплату.

Об’єкт дослідження – FullStack (Frontend та Backend application) клієнтська та серверна частина веб-сервісу, розробка документації для програмного продукту.

Предмет дослідження – методи та засоби для розробки веб-сервісу з менеджменту онлайн підписок користувачів, а також технології, які будуть використовуватись для реалізації даного веб-сервісу.

Завдання дослідження:

– здійснити аналіз предметної області;

– здійснити аналіз існуючих веб-сервісів, додатків для менеджменту онлайн підписок;

– обрати потрібні технології та засоби для розробки;

– спроектувати базу даних;

– розробити серверну частину веб-сервісу;

– розробити клієнтську частину веб-сервісу;

– розробити документацію для програмного продукту;

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ ВЕБ-СЕРВІСІВ З МЕНЕДЖМЕНТУ ОНЛАЙН ПІДПИСОК ТА ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Актуальність теми. Сьогодні мільйони людей у всьому світі проводять години за серфінгом в Інтернеті, заробітком та витрачанням грошей, пошуку половинки, отриманням вищої освіти, переглядом фільмів, прослуховування музики, тощо. Список практично нескінченний.

Веб-розробка еволюціонувала від простих веб-сторінок до складних веб-сайтів, які можуть занурити користувачів у певну тему та створити досвід, який було б неможливо відтворити без використання комп’ютера чи іншого девайсу, підключеного до мережі. Тут також можна згадати вислів Білла Ґейтса: «Якщо Ваш бізнес не виходить в Інтернет, тоді ваш бізнес припинить роботу». Вплив веб-присутності на ефективність бізнесу тепер абсолютно зрозумілий. Хороший веб-сервіс дає компаніям можливість залучати клієнтів, дізнаватися про їх поведінку, пропонувати релевантні відвідувачам речі, і, нарешті, заробляти на цьому.

Веб-сервіси є наслідком еволюції Інтернету у відкрите середовище, яке полегшує взаємодію складних ділових і наукових програм. Веб-сервіси пов’язані з проблемами забезпечення систематичної взаємодії між програмами в Інтернеті та інтеграції існуючої мережевої комп’ютерної інфраструктури в мережі. Модель веб-сервісів базується на існуючих галузевих зусиллях зі стандартизації, зосереджених навколо мови розширюваної розмітки (XML) і пропонує двосторонній підхід до вирішення вимог сумісності, пов’язаних із гетерогенними системами: увімкнення базової сумісності за допомогою невеликого набору загальних протоколів і розробка уніфікованого представлення мережевих програм, які доступні за допомогою кількох

протоколів зв’язку. Мета веб-сервісів полягає в тому, щоб забезпечують гнучку структуру, де універсальна взаємодію не перешкоджає ефективній інтеграції.

Загалом, веб-сервіс – це мережева програма, яка здатна взаємодіяти за допомогою стандартних веб-протоколів між програмами через чітко визначені інтерфейси, і яка є описується за допомогою стандартної мови функціонального опису.

Останнім часом послуги на основі передплати набули значної популярності, оскільки компанії все частіше застосовують цю модель для забезпечення стабільних потоків доходу та культивування лояльності клієнтів. У сфері моделі підписки цифрові підписки користуються більшим успіхом. За останні кілька років попит на віртуальні послуги та цифрові товари значно зріс, особливо після COVID-19. Від потокових платформ до цифрових курсів споживачі активно шукають орієнтовані на підписку рішення, які легко інтегруються в їх цифрове життя. З точки зору бізнесу, цифрові підписки мають ряд переваг. Вони створюють надійний потік постійних доходів, а також дозволяють підприємствам отримати доступ до нових ринків, які можуть залишатися за межами досяжності звичайних операцій. Це зростання в галузі послуг на основі передплати було спричинене прогресом технологій, зміною поведінки споживачів і переходом до більш персоналізованого та зручного досвіду.

Концепція передплати бере свій початок у 19 столітті, коли друковані видання, такі як газети та журнали, пропонували читачам підписку на регулярну доставку. Ця модель набула подальшого поширення з появою телекомунікаційних послуг, таких як стаціонарні телефони та кабельне телебачення. Клієнти сплачували щомісячну плату за доступ до цих послуг, що стало нормою для багатьох домогосподарств. Починаючи з 1990-х років, технологічна індустрія прийняла модель підписки з появою програмного забезпечення як послуги (SaaS). Замість того, щоб купувати ліцензії на програмне забезпечення заздалегідь, користувачі можуть підписатися на хмарні програмні додатки, сплачуючи на регулярній основі. Такий підхід

демократизував доступ до програмного забезпечення, зробивши його більш доступним і доступним для ширшого кола компаній і окремих осіб. Нове тисячоліття відкрило нову зорю в індустрії розваг. Музична індустрія та індустрія розваг зазнали значних змін із запровадженням служб потокового передавання медіа. Такі компанії, як Netflix (для відео) і Spotify (для музики), пропонували величезні бібліотеки вмісту за щомісячну передплату, замінюючи традиційні моделі покупки чи оренди. Відтоді за останнє десятиліття чи близько того цифрові послуги передплати зросли в геометричній прогресії. Netflix, один із піонерів у сфері потокового передавання, у 2007 році перейшов від своєї моделі прокату DVD до потокового передавання. До 2010 року компанія зарекомендувала себе як лідер потокового передавання, пропонуючи зростаючу бібліотеку телешоу та фільмів. Інші сервіси, такі як Hulu та Amazon Prime Video, також вийшли на арену. Період між 2018 і 2019 роками став точкою насичення для послуг підписки на потокове передавання з появою кількох нових гравців, таких як Disney+, Apple TV+, Sony LIV, Zee5, Voot і HBO Max. Пандемія COVID-19 призвела до різкого зростання попиту на потокові послуги, оскільки люди проводили більше часу вдома.

Оскільки кінотеатри закриті та прямі трансляції скасовані, трансляція стала основним джерелом розваги для багатьох. Однак потокова передача була не єдиною послугою підписки, яка зазнала величезного зростання за останнє десятиліття. Office 365 від Microsoft і Creative Cloud від Adobe є яскравими прикладами того, як компанії-виробники програмного забезпечення перейшли від безстрокових ліцензій до моделей на основі передплати. Фітнес-індустрія спостерігала зростання кількості послуг на основі передплати для онлайн-програм тренувань, віртуальних класів і оздоровчих програм. Такі компанії, як Amazon, запровадили такі сервіси, як Amazon Prime, пропонуючи такі переваги, як безкоштовна доставка, потокове передавання та ексклюзивні пропозиції в обмін на річну плату за підписку. Індустрія відеоігор прийняла моделі передплати з такими послугами, як Xbox Game Pass і PlayStation Now, надаючи геймерам доступ до великої бібліотеки ігор. Модель передплати поширилася за

цифрові послуги на фізичні товари через послуги передплатних скриньок. Ці служби підбирають і доставляють продукти на основі вподобань клієнтів, починаючи від косметичних засобів і закінчуючи їжею та книгами. Фактор несподіванки та зручності сприяв їх популярності.

Саме тому актуальності набула така проблема, як трекінг великої кількості підписок. У даній кваліфікаційній роботі представлено простий додаток, який добре вміє робити одну річ: зберігати передплати користувача та нагадувати про продовження, що є дуже зручним рішенням у сучасному постійно прогресуючому світі.

В кінці 2022 року Вардана К. провела невелике дослідження [1], в якому вона поставила дві цілі: перша – перевірити, скільки людей потребують програми для керування підписками; друга – виявити проблемні моменти під час керування підписками. Вона створила чотири запитання та варіанти відповіді А або Б, ось якими були запитання:

– ви підписалися на цифрові продукти? (Так / Ні);

– на скільки цифрових продуктів ви підписалися? (Менше 3 (≤2) / Більше 2 (>2);

– ви часто забуваєте продовжити або припинити підписку? (Так / Ні);

– вам потрібен додаток для керування підписками? (Так / Ні).

Результати показали, що 97 % із 34 опитаних підписалися на цифрові продукти, а 45 % із них підписалися на більше ніж 2 цифрові продукти. 45 % або 15 із 33 підписників часто забувають вжити заходів, щоб продовжити або призупинити свою підписку. А якщо подивитися глибше, то 9 з 15 підписників часто забувають вжити заходів, вони підписуються на більше ніж 2 продукти одночасно. Таким чином, існує кореляція між кількістю підписок і забуттям вжити заходів.

Якщо подивитись на загальну кількість передплатників, які підписалися на цифрові продукти, 39,4 % потрібен додаток для керування підписками, щоб допомогти їм керувати підпискою, а якщо ми подивимося на загальну кількість

передплатників, які часто забувають вжити заходів, 66,7 % потрібен додаток для керування підписками.

Це дослідження показало, що є потенціал у майбутньому, коли пропонуватиметься все більше цифрових продуктів, ця програма дійсно знадобиться.

Зрозуміло, що ідея не нова, однак, що не так із додатками та веб-сервісами конкурентів? Майже всі програми зі схожим функціоналом ніби зроблені під копірку. У них передплати створюються за готовими шаблонами. Якщо потрібного сервісу немає в списку — значить потрібно створювати свій власний шаблон, який не матиме логотипу, а значить дизайн передплати помітно відрізнятиметься від інших. Реалізації проєктів з менеджментом підписок представлено на рисунках 1.1, 1.2, 1.3.

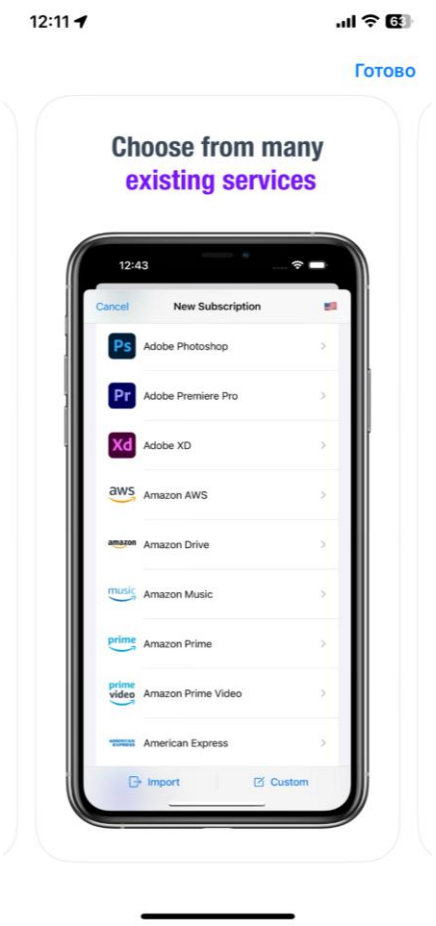

Рисунок 1.1 – Інтерфейс додатку Subscriptions – Track Expenses [2]

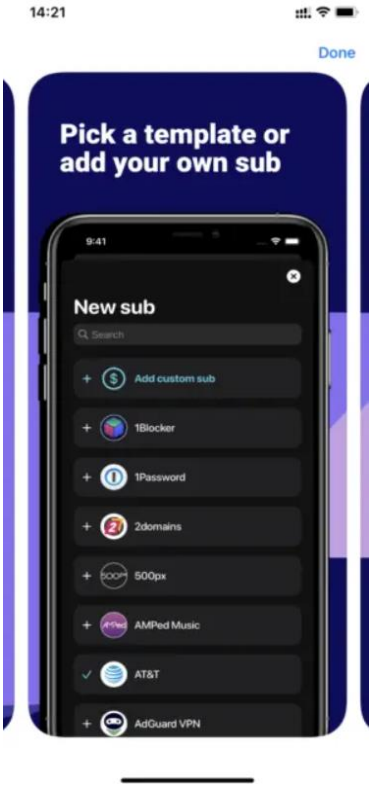

Рисунок 1.2 – Інтерфейс додатку Subsee [3]

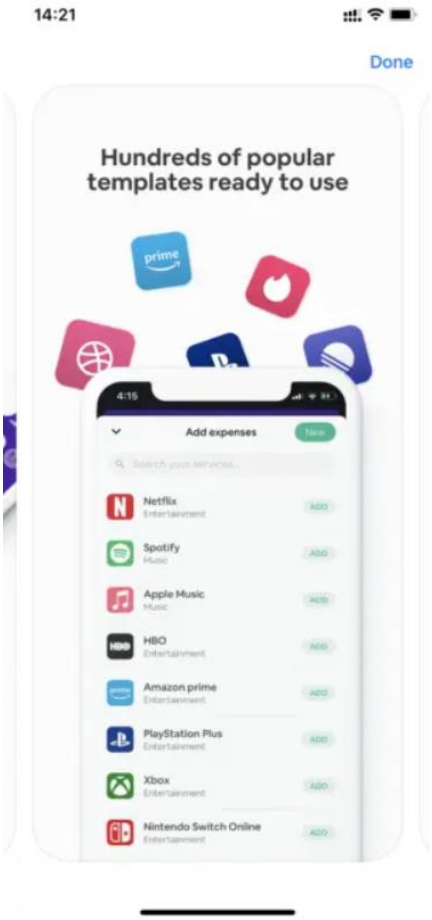

Рисунок 1.3 – Інтерфейс додатку Billbot [4]

До того ж, не всі програми вміють вчасно нагадувати про продовження передплат. В основному, у користувачів такі скарги: або програма не надсилає повідомлення, або повідомлення надходять пізно, коли підписку вже не можна скасувати і гроші списані. В Subs-Managik App, проекті, який розроблено в даній кваліфікаційній роботі, було враховано це і реалізовано надійний сервіс, який вміє коректно встановлювати нагадування про платежі та синхронізувати їх між пристроями.

У більшості додатків конкурентів дизайн не корелює з сучасними трендами UX/UI. У Rocket Money – Bills & Budgets якісний додаток, але вони не працюють в України та багатьох інших країнах та їх продукт дещо більше, ніж трекер підписок (рис. 1.4).

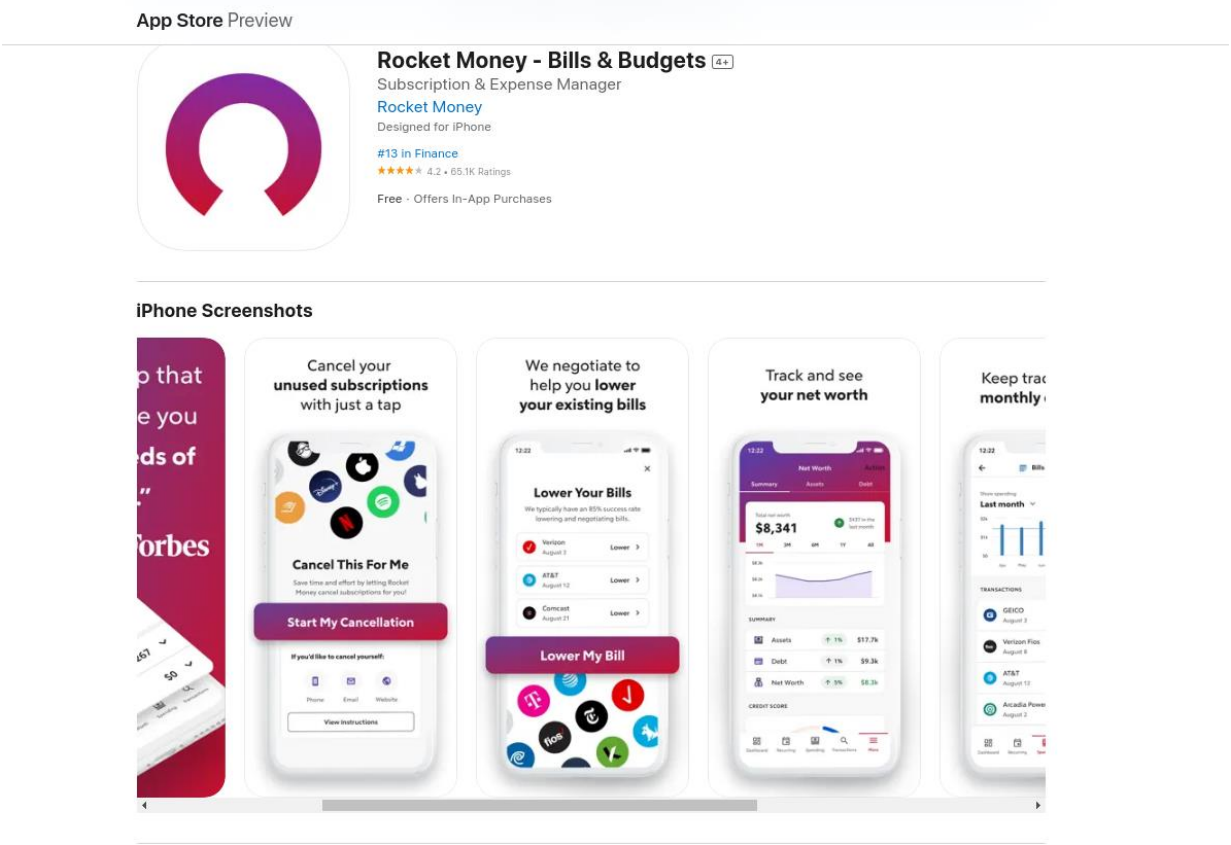

Рисунок 1.4 – Додаток Rocket Money – Bills & Budgets в App Store [5]

На додачу, про ідею в додатках конкурентів використання шаблонів підписок. Це потрібно, щоб підписки мали логотипи. Але чи користувачам потрібні логотипи? Як на мене – ні. Користувачеві потрібний не гарний логотип

Apple Music, а зрозуміла інформація про підписку – ось, що справді важливо. До того ж, у користувачів у середньому по 3-5 активних передплат. Логотипи корисні, коли список великий та очам потрібна допомога у вигляді логотипу, щоб простіше було знайти потрібний елемент у списку. Наприклад, як це робиться у більшості банківських додатків.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Кваліфікаційна робота розробляється для того, щоб клієнт міг зручно управляти всіма своїми підписками на різні сервіси в одному веб-сервісі.

Інтерфейс користувача – це безпосередньо сам сайт з яким взаємодіє клієнт. Інтерфейс користувача повинен бути розроблений з урахуванням сучасних UX/UI трендів але і водночас простим, аби клієнт інтуїтивно розумів, куди саме йому потрібно переходити та тиснути аби зробити бажану дію.

Інтерфейс користувача повинен містити такий функціонал:

– додавання підписки на сервіс;

– редагування вибраної підписки користувача;

– видалення вибраної підписки користувачем;

– можливість створювати заявку на додавання сервісу, який відсутній;

– реєстрація та авторизація користувача;

– особистий кабінет користувача зі списком підписок та інформацією про самого ж користувача/редагування профілю користувача;

– авторизація сервісу, на якому куплено підписку;

– посторінкова навігація;

Сформулюємо такі завдання дослідження:

– здійснити аналіз предметної області;

– здійснити аналіз існуючих веб-сервісів, додатків для менеджменту онлайн підписок;

– обрати потрібні технології та засоби для розробки;

– спроектувати базу даних;

– розробити серверну частину веб-сервісу;
– розробити клієнтську частину веб-сервісу;
– розробити документацію для програмного продукту.

Висновки до розділу 1

За останні кілька років галузь послуг передплати зазнала значного зростання, змінивши спосіб взаємодії компаній зі споживачами. Для споживачів моделі підписки часто пропонують зручність, відчуття спільності та привабливість регулярних оновлень або сюрпризів. Хоча перспективи для постачальників послуг здаються багатообіцяючими, вони не позбавлені проблем. У деяких секторах, особливо в потокових послугах, спостерігається насичення ринку, що призводить до того, що споживачі відчувають себе переповненими великою кількістю вибору.

Саме тому виникла така проблема, як необхідність автоматизації процесу підписок користувачів.

Основною метою проекту є розробка веб-сервісу, який буде мати змогу автоматизувати процес керування підписками користувача. Під час розробки повинні бути враховані помилки допущені реалізаціями конкурентів. Веб-сервіс повинен працювати швидко та якісно, для цього слід також підібрати хостинг з відповідними параметрами.

Веб-сервіс повинен чітко виконувати покладені на нього функції без помилкових спрацювань.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Починаючи з визначення вимог до розроблюваного програмного забезпечення та аналізу, слід чітко зрозуміти мету розробки, цільову аудиторію, яка потенційно зможе користуватися веб-додатком та інші важливі моменти, саме тому слід почати з опису.

Ця програма призначена для користувачів, які мають певну кількість підписок на цифрові сервіси, як-от: Amazon Prime, Netflix, Starlink, Spotify та інші, і мають проблеми з керуванням ними. Для людей, які забули, що перший місяць підписки безкоштовний, а потім їм доведеться заплатити кошти. Або для людей, які забули, що ціни в майбутньому зміняться. Також, до прикладу, якщо ви хочете протестувати якийсь сервіс лише на місяць, він вам не сподобався, але ви забули від нього скасувати підписку, і він буде забирати ваші кошти, поки ви не помітите цього в мобільному додатку свого банку. Але з цим також є велика проблема. Коли ви бачите свої щомісячні платежі, ви не завжди бачите, яка саме послуга забирає ваші гроші. Або ви просто хочете побачити суму вартості всіх ваших послуг в одному місці. Саме тому метою веб-додатку є автоматизація цього процесу.

Перш ніж починати розробку веб-застосунку, потрібно описати функціональні вимоги для розроблюваного застосунку.

Функціональні вимоги для клієнтської частини сайту:

– користувач повинен мати можливість створити власний обліковий запис вказавши такі дані, як: ім’я, електронна пошта, номер телефону, пароль та поле підтвердження паролю;

– користувач повинен мати можливість увійти в систему за допомогою електронної пошти та паролю;

– користувач повинен мати можливість ввести бажаний сервіс, на якому у нього є існуюча підписка, до веб-додатку;

– користувач повинен мати можливість редагувати вже додані ним підписки.

Нефункціональні вимоги для клієнтської частини сайту:

– система повинна захищати від несанкціонованого входу в облікові записи користувачів;

– всі паролі користувачів повинні зберігатися в захешованому вигляді в базі даних;

– сайт повинен швидко завантажуватися та працювати без помилок;

– оптимізовані запити до бази даних.

Щодо архітектури веб-додатку у даному проекті, то вона зображена на рисунку 2.1.

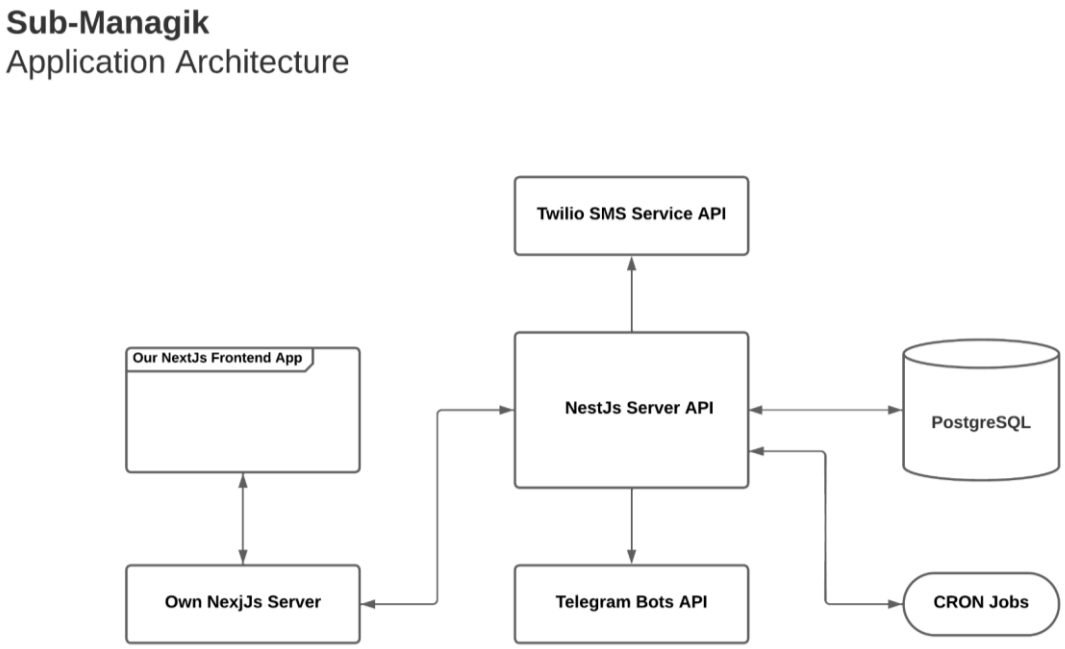

Рисунок 2.1 – Архітектура веб-додатку

Інтерфейсна частина, оскільки ми використовуємо Next.Js, має власний сервер який займається SSR (сервер-сайд-рендеринг, потрібних для кращого SEO, та швидкого завантаження нашого веб-сервісу), і ми використовуємо наш бекенд, написаний на Nest.Js, який є окремим від нашого інтерфейсу. Як базу даних ми використовуємо PostgreSQL. Також у нас є деякі інтеграції з Twilio для SMS-сповіщень і Telegram API для сповіщень бота для адміністраторів про деякі події, у нашому випадку, якщо сповіщення адміністратора про нові запитані послуги, яких немає в нашій програмі. Також у нас є окремий модуль на нашому сервері для виконання певних задач в певний час. В нашому випадку ми виристовуємо цей модуль для надсилання SMS нагадувань користувачу про майбутню оплату його підписки.

Проект організований у двох основних каталогах:

– Backend. Містить серверну частину з використанням коду Nest.Js і схеми бази даних;

– Frontend. Містить клієнтський код, створений за допомогою React і Next.js.

Більш детальніше та розгорнутіше структуру побудови веб-сервісу можна переглянути у додатку А.

2.2 Вибір засобів, методів і алгоритмів поставленого завдання

Для створення веб-сервісу було обрано наступні технології, які слід поділити на дві категорії: інтерфейс веб-додатку (Frontend) та частину, яка схована від користувача всередині веб-застосунку (Backend).

Інтерфейс:

– React v18 [6];

– Next.Js v14 [7] – з використанням TypeScript для кращої типизації нашого додатку;

– Lucide – красиві іконки на нашому веб-сервісі;

– React Hook Form [8] – керування формами;

– Yup [9] – валідація введених даних в формах;

– Tailwindcss [10] – для швидкого написання стилів;

– Tanstack [11] – для кешування наших API запитів на стороні клієнта та автоматичне оновлення даних;

– Flowbite [12] – UI-бібліотека з готовими компонентами;

– date-fns [13] – керування датами в веб-сервісі;

– Axios [14] — HTTP запити до нашого API (сервера);

– sonner – сповіщення на веб-сервісі;

– js-cookie – обробка файлів cookie в веб-сервісі.

Backend:

– Nest.Js v10 [15] – з використанням TypeScript для кращої типизації нашого додатку;

– PostgreSQL – як база даних;

– Prisma [16] – ORM база даних;

– @nestjs/jwt і @nestjs/passport – керування автентифікацією та авторизацією;

– @nestjs/cron – створення cron завдань;

– cookie-parser – керування файлами cookie в веб-сервісі;

– twilio [17] – інтеграція SMS;

– node-telegram-bot-api [18] – як API телеграм-бота;

– date-fns – керування датами в програмі.

Також додаткові залежності можна знайти в проекті у файлі package.json у папці інтерфейсу та package.json у внутрішній папці відповідно.

Висновки до розділу 2

Підсумовуючи другий розділ, слід сказати, що правильне проектування додатку є важливим етам у розробці будь-якого програмного застосунку. Також важливим кроком у розробці є планування, адже саме з даного етапу

розпочинається процес роботи над проектом у більшості спеціалістів в даній галузі, що, на мою думку, є правильним рішенням.

Чітко спроектований проект показує розробнику чіткий план дій, візуалізований план реалізації поставлених завдань. А щоб досягти цього, фахівці зазвичай будують блок-схеми, графи, таблиці для більш детального розуміння процесу.

Останньою не менш важливою складовою будь-якого проекту є правильний підбір технологій, що, в свою чергу, дозволить спроектувати легку інтеграцію між різними бібліотеками, додати сучасності та забезпечити безпеку в проекті.

РОЗДІЛ 3
РОЗРОБКА ВЕБ-СЕРВІСУ З МЕНЕДЖМЕНТУ ОНЛАЙН ПІДПИСОК НА NEST.JS ТА NEXT.JS

3.1 Практична реалізація об’єкта проектування

Загалом, дана кваліфікаційна робота – це простий додаток, який містить авторизацію по JWT-токену з використанням серверних cookie та базові CRUD операції на додавання, редагування, оновлення та видалення підписок користувача. Але й в цьому додатку є цікаві рішення, детальний розгляд яких наведено нижче.

Для початку, слід розглянути, як відбувається авторизація з використанням JWT-токенів та серверних cookie на Backend сервері. Для цього слід розглянути файл backend/src/auth/auth.service.ts (додаток Б).

В додатку Б зображено валідацію даних користувача, перевірку на унікальність під час реєстрації. Далі відбувається генерація JWT-токену та повертаються дані клієнту, які потрібні для авторизація до веб-сервісу.

У наступній функції описується як саме відбувається надсилання сповіщень користувачам з нагадуванням про оплату підписки. Для реалізації функціональної частини потрібно:

– створити CRON Job і вибрати регулярність цієї роботи. В майбутньому даних елементів може бути багато під різні задачі, але наразі в проекті для демонстрації створено один елемент, який буде виконуватись кожного дня о 21:00;

– в CRON Job вибираються всі користувачі з бази даних (лістинг 3.1).

Лістинг 3.1 – Код методу, який вибирає всіх користувачів з бази даних

```
getAllWithSubscriptionsAndService() {
   return this.prisma.user.findMany({
     include: {
       subscriptions: {
         include: {
           service: true,
```

```
          }
        }
      },
    })
  }
```

Кінець лістингу 3.1

Наступним кроком потрібно перевірити чи дата проплати підписки на сервіс (subscription.nextPaymentAt) сьогоднішня. Але тут є нюанс. Так як користувач один раз вибирає дату оплати підписки, сервісу кожного разу потрібно актуалізувати цю дату, тобто змінювати рік на теперішній, якщо підписка була створена в минулому, також міняти місяць на актуальний. Тільки так буде змога актуалізувати дату оплату підписки, навіть якшо вона була введена тільки один раз. За дану логіку відповідає метод adjustNextPaymentDate (лістинг 3.2).

Лістинг 3.2 – Код методу adjustNextPaymentDate

```
private adjustNextPaymentDate(nextPaymentAt: string): Date {
    const now = new Date();
    let nextPaymentDate = parseISO(nextPaymentAt);
    nextPaymentDate = setYear(nextPaymentDate, now.getFullYear());
    nextPaymentDate = setMonth(nextPaymentDate, now.getMonth());

    if (isBefore(nextPaymentDate, now) && !isToday(nextPaymentDate)) {
      nextPaymentDate = addMonths(nextPaymentDate, 1);
    }

    return nextPaymentDate;
  }
```

Кінець лістингу 3.2

Потім, оскільки тепер сервіс у проекті має актуальну дату і є змога перевірити чи це відбулось сьогодні (перевірка актуальності) і чи користувач вибрав опцію сповіщення через SMS-нагадування про оплату (лістинг 3.3).

Лістинг 3.3 – Код, який перевіряє чи надсилати користувачу SMS чи ні

```
if (isToday(adjustedDate) && subscription.isNotifying) {
          this.logger.log(`Sending SMS notification to user ${user.name} to
${user.phone} for service ${subscription.service.fullName} with price
${subscription.price}$...`);
          totalSubscriptions++;

          try {
            await this.sendSubscriptionNotificationSMS(
              user.phone,
              subscription.service.fullName,
              adjustedDate
            );

            successCount++;
          } catch (error) {
            failureCount++;
          }
        }
```

Кінець лістингу 3.3

Тепер, використовуючи налаштований Twilio Client, є змога надіслати користувачу SMS-сповіщення (лістинг 3.4).

Лістинг 3.4 – Код методу, який надсилає SMS-сповіщення користувачу

```
private async sendSubscriptionNotificationSMS(phone: string, serviceName:
string, nextPaymentDate: Date) {
   const formattedDate = format(nextPaymentDate, 'dd.MM.yyyy');
   const message = `Reminder: Your subscription to ${serviceName} is due
on ${formattedDate}. So today will be payment day.`;

   try {
     await this.twilioClient.messages.create({
       body: message,
       from: this.configService.get<string>('TWILIO_PHONE_NUMBER'),
       to: phone,
     });
     this.logger.log(`Succesfully sent SMS to ${phone}: ${message}`);
   } catch (error) {
```

```
      this.logger.error(`Failed to send SMS to ${phone}:
${error.message}`);
      throw error;
    }
  }
```

Кінець лістингу 3.4

Повний код зображений в додатку В.

Також, як виглядає саме SMS-сповіщення, надіслане веб-сервісом, зображене на рисунку 3.1.

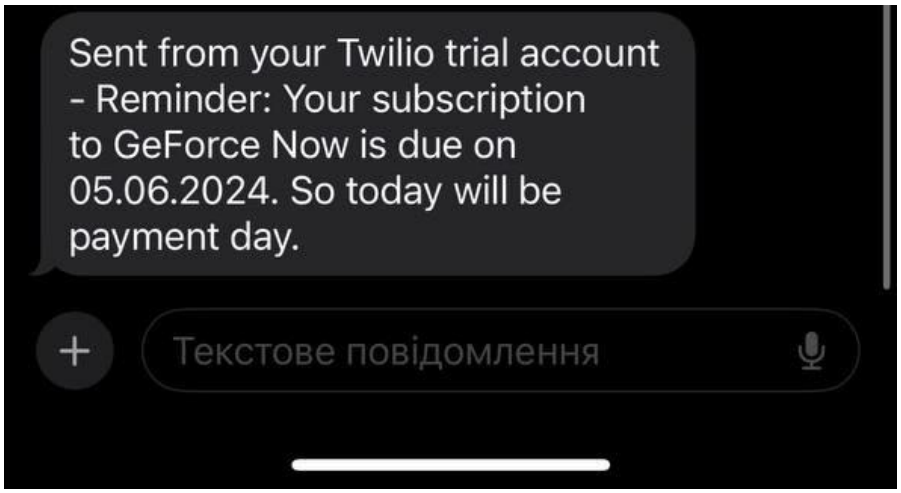

Рисунок 3.1 – SMS-сповіщення, реалізовані у проекті

Якщо розглянути як саме відбувається надсилання повідомлення в Telegram Bot з сповіщенням адміністратору про те, що користувач не знайшов сервіс, який він хотів в веб-сервісі, то це описано нижче.

Для початку користувачу потрібно створити заявку на сторінці https://sub-managik.com/request-service як показано на рисунку 3.2.

Після того як користувач створить заявку, вона з’явиться в Telegram Bot як показано на рисунку 3.3.

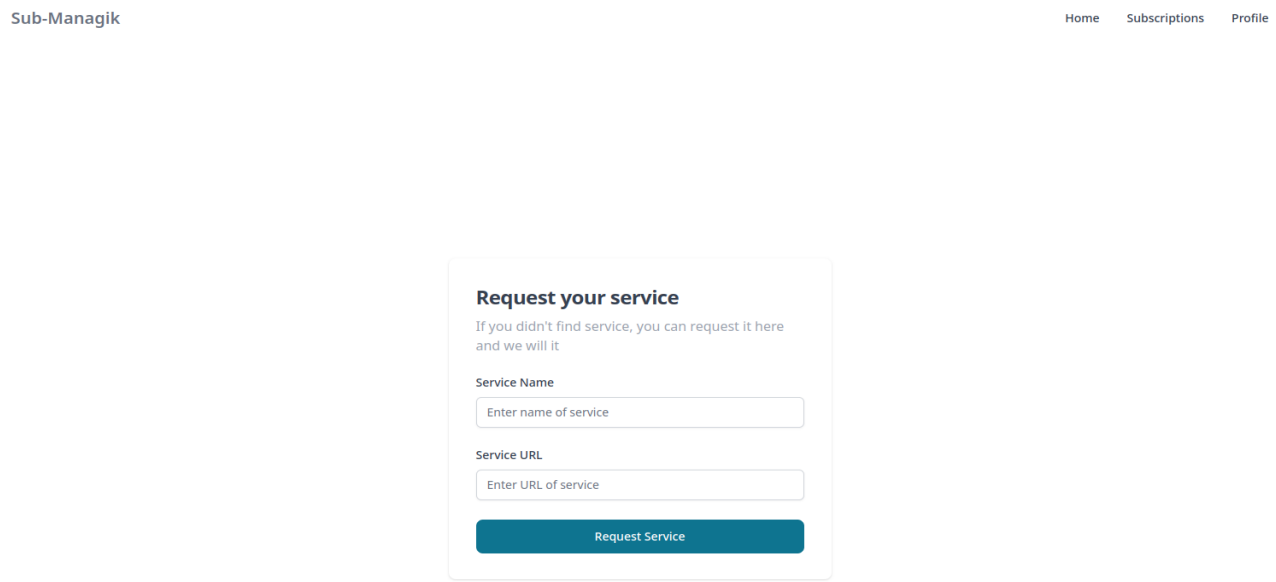

Рисунок 3.2 – Створення заявки

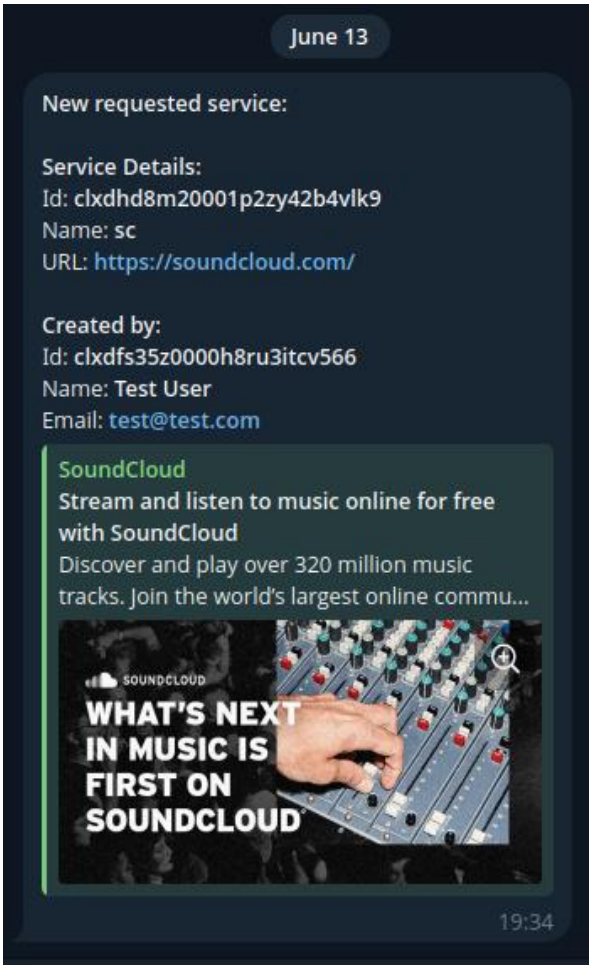

Рисунок 3.3 – Оформлена у Telegram Bot заявка від користувача

На рисунку 3.3 зображено ID заявки, посилання на сервіс з назвою та URL на сам сервіс, це потрібно для адміністратора. Також не менш важливим є те, що на рисунку 3.3 також присутні дані користувача, який залишив цю заявку. Для реалізації даних опцій потрібно:

Створити роутер, який буде обробляти запити з клієнтської сторони (лістинг 3.5).

Лістинг 3.5 – Код роутера, який обробляє запити клієнта на створення заявки

```
import { Injectable } from 'nestjs/common';
import { PrismaService } from 'src/prisma.service';
import { CreateRequestedServiceDto } from './dto/requested-service.dto';
import { TelegramService } from 'src/telegram/telegram.service';
import { UserService } from 'src/user/user.service';

@Injectable()
export class RequestedServiceService {
 constructor(
   private prisma: PrismaService,
   private userService: UserService,
   private telegramService: TelegramService,
 ) {}

 async create(userId: string, dto: CreateRequestedServiceDto) {
   const requestedService = await this.prisma.requestedService.create({
     data: {
       ...dto,
       userId: userId,
     },
   })

   const user = await this.userService.getById(userId)

   await
this.telegramService.sendRequesterServiceNotifyMessage(requestedService,
user)

   return requestedService
 }

 async getAllByUserId(userId: string) {
   return this.prisma.requestedService.findMany({
```

```
      where: {
        userId,
      },
    })
  }
}
```

Кінець лістингу 3.5

Додати функціонал надсилання повідомлення в Telegram Bot (лістинг 3.6).

Лістинг 3.6 – Код надсилання повідомлення в Telegram Bot

```
import { Injectable } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { RequestedService, User } from '@prisma/client';
import * as TelegramBot from 'node-telegram-bot-api';
@Injectable()
export class TelegramService {
 private bot: TelegramBot;
 constructor(
   private configService: ConfigService
 ) {
   const token = this.configService.get<string>('TELEGRAM_BOT_TOKEN');
   this.bot = new TelegramBot(token, { polling: true });
 }
 sendPlainMessage(message: string) {
   const chatId = this.configService.get<string>('TELEGRAM_BOT_CHAT_ID');
   return this.bot.sendMessage(chatId, message);
 }
 sendRequesterServiceNotifyMessage(requestedService: RequestedService, user:
User) {
   const chatId = this.configService.get<string>('TELEGRAM_BOT_CHAT_ID');
   return this.bot.sendMessage(
     chatId,
     `<b>New requested service:</b>\n\n<b>Service Details:</b>\nId:
<b>${requestedService.id}</b>\nName: <b>${requestedService.name}</b>\nURL:
<b>${requestedService.url}</b>\n\n<b>Created by:</b>\nId:
<b>${user.id}</b>\nName: <b>${user.name}</b>\nEmail:
<b>${user.email}</b>\n`,
     { parse_mode: 'HTML' }
   );
 }}
```

Кінець лістингу 3.6

3.2 Розробка бази даних

Дуже важливим кроком під час реалізації розроблюваного веб-сервісу є проектування бази даних з полями, які будуть в таблицях і зв’язками між таблицями:

1) модель «Користувач». Модель User представляє користувача в системі. Поля:

– id(String) – унікальний ідентифікатор, згенерований за допомогою cuid();

– name(String) – ім’я користувача;

– email(String) – електронна адреса користувача, унікальна;

– phone(String, необов’язково) – номер телефону користувача, унікальний;

– password(String) – пароль користувача;

– createdAt(DateTime) – позначка часу створення користувача, за замовчуванням поточні дата й час;

– updatedAt(DateTime) – позначка часу останнього оновлення користувача, автоматично встановлюється під час оновлення;

– deletedAt(DateTime, необов’язково) – позначка часу, коли користувача було видалено;

– subscriptions(Subscription[]) – список підписок, пов’язаних із користувачем;

– requestedServices(RequestedService[]) – список бажаних користувачем сервісів. Назва таблиці – user;

2) модель підписка – модель Subscription являє собою підписку на послугу. Поля:

– id(String) – унікальний ідентифікатор, згенерований за допомогою cuid();

– user(User) – користувач, якому належить підписка;

– userId(String) – посилання на зовнішній ключ в таблиці User.id;

– service(Service) – послуга, на яку ви підписалися;

– serviceId(String) – посилання на зовнішній ключ в таблиці Service.id;

– price(Float) – ціна підписки;

– note(String, необов’язково) – додаткові примітки щодо підписки;

– isNotifying(Boolean) – вказує, чи ввімкнено сповіщення для підписки;

– nextPaymentAt(DateTime) – позначка часу наступної дати платежу;

– createdAt(DateTime) – мітка часу створення підписки, за замовчуванням поточні дата й час;

– updatedAt(DateTime) – позначка часу останнього оновлення підписки, автоматично встановлюється під час оновлення;

– deletedAt(DateTime, необов’язково) – позначка часу видалення підписки. Назва таблиці – subscription;

3) модель сервіс – модель Service представляє послугу, на яку можна підписатися. Поля:

– id(String) – унікальний ідентифікатор, згенерований за допомогою cuid();

– fullName(String) – повна назва послуги;

– shortName(String) – коротка назва послуги;

– backgroundColor(String) – колір фону, пов’язаний із послугою. Це основний колір логотипу сервісу;

– subscriptions(Subscription[]): список підписок, пов’язаних із послугою;

– createdAt(DateTime): мітка часу створення сервісу, за замовчуванням поточні дата й час;

– updatedAt(DateTime): позначка часу останнього оновлення сервісу, автоматично встановлюється під час оновлення;

– deletedAt(DateTime, необов’язково): позначка часу видалення сервісу. Назва таблиці – service;

4) модель RequestedService: модель RequestedService представляє послугу, бажану користувачем, яка ще не доступна. Поля:

– id(String): унікальний ідентифікатор, згенерований за допомогою cuid();

– user(User): користувач, який запитав послугу;

– userId(String): посилання на зовнішній ключ в таблиці User.id;

– name(String): назва запитуваної послуги;

– url(String): URL-адреса запитуваної служби;

– createdAt(DateTime): позначка часу створення бажаного сервісу, за замовчуванням поточні дата й час;

– updatedAt(DateTime): позначка часу останнього оновлення бажаного сервісу, автоматично встановлюється під час оновлення;

– deletedAt(DateTime, необов’язково): позначка часу, коли бажаного сервісу було видалено. Назва таблиці: requested_service.

Далі опишемо відносини в базі даних:

– користувач і підписка:

1) кожен User може мати декілька Subscriptions;
2) це представлено полем subscriptions у User моделі;
3) відношення встановлюється через відношення «один до багатьох»;

– користувач і бажанний сервіс:

1) кожен User може мати декілька RequestedServices;
2) це представлено полем requestedServices у User моделі;
3) відношення встановлюється через відношення «один до багатьох»;

– підписка та користувач:

1) кожен Subscription належить до одного User;
2) це представлено полем user у Subscription моделі;
3) відношення встановлюється через відношення «багато до одного»;

– підписка та сервіс:

1) кожен з них Subscription пов’язаний з одним Service;
2) це представлено полем service у Subscription моделі;
3) відношення встановлюється через відношення «багато до одного»;

– сервіс і підписка:

1) кожен Service може мати декілька Subscriptions;
2) це представлено полем subscriptions у Service моделі;
3) відношення встановлюється через відношення «один до багатьох»;

– потрібний сервіс та користувач:

1) кожен RequestedService належить до одного User;
2) це представлено полем user у RequestedService моделі;
3) відношення встановлюється через відношення «багато до одного».

Зображуючи структуру схематично, це буде виглядати як на рисунку 3.4.

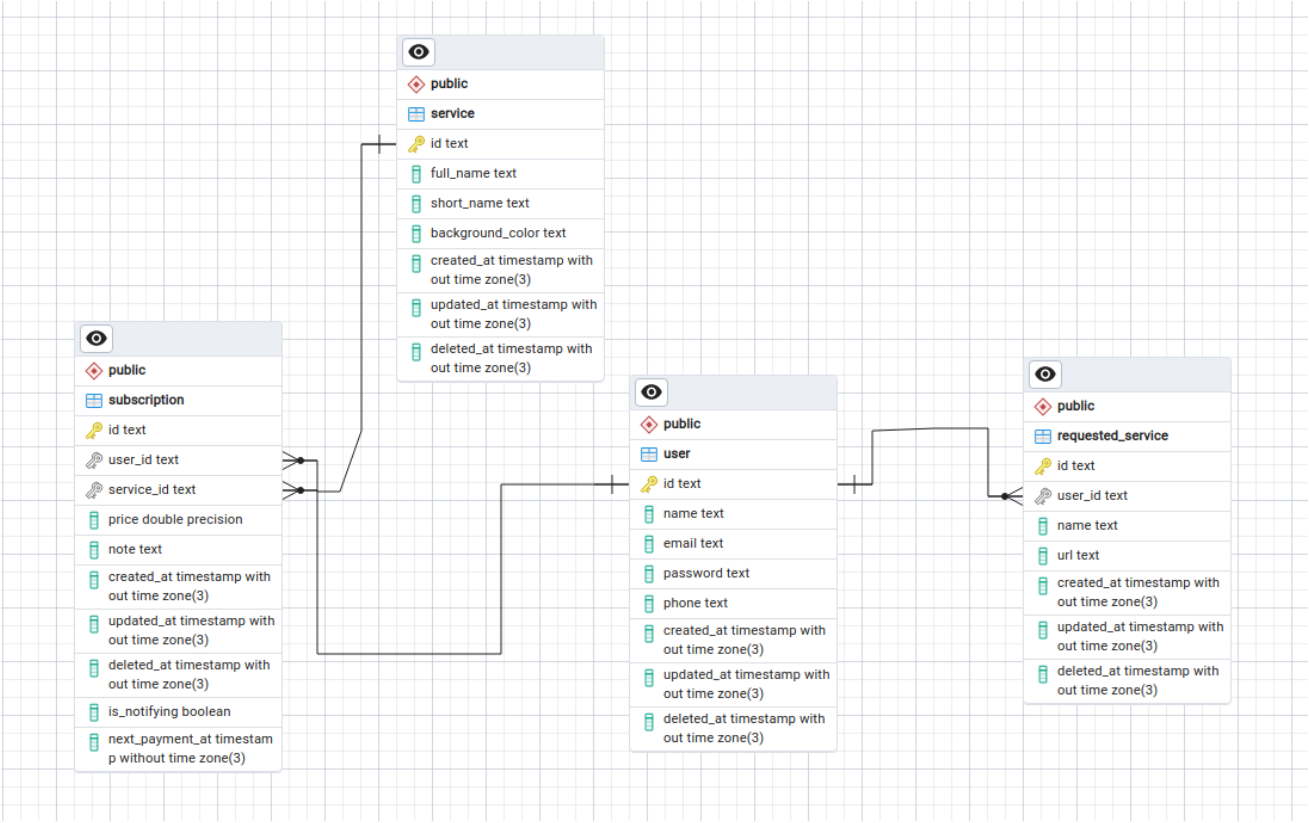

Рисунок 3.4 – Структура таблиць бази даних

3.3 Розробка документації для програмного продукту

Дуже важливим етапом в розробці веб-сервісу є його деплоймент і правильне налаштування, щоб цим веб-сервісом могли користуватись люди з будь-якої точки світу. Цей етап ділиться на декілька кроків:

– вибір VPS провадера, у випадку даної кваліфікаційної роботи це hostinger.com;

– налаштування VPS серверу;

– розгортання нашого веб-сервісу на сервері;

– додавання домену;

– додавання SSL сертифікату.

Отже, для цього потрібно:

– створити новий акаунт на hostinger.com (рис. 3.5);

– додати новий сервер.

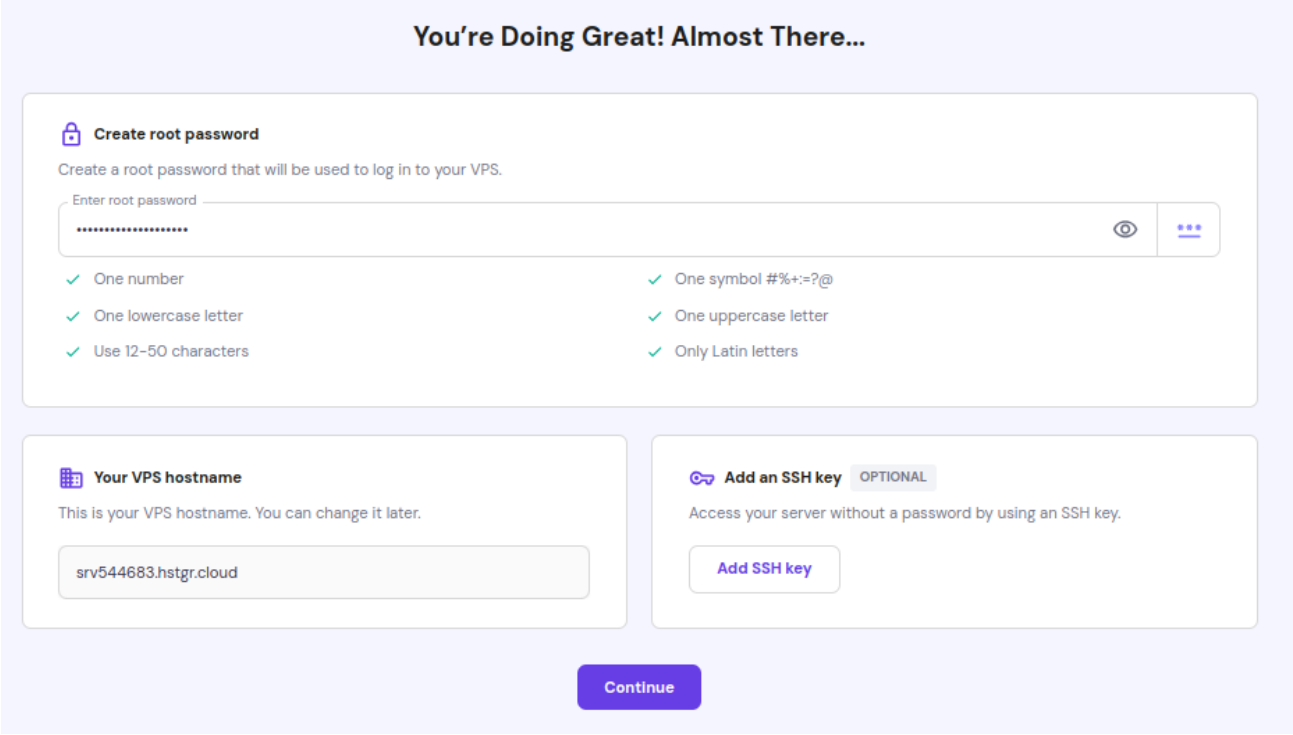

Рисунок 3.5 – Екран підтвердження створення VPS серверу на hostinger.com

Додати SSH-ключ, щоб кожного разу не авторизуватись, також це не дасть підключитись до сервера без SSH-ключа.

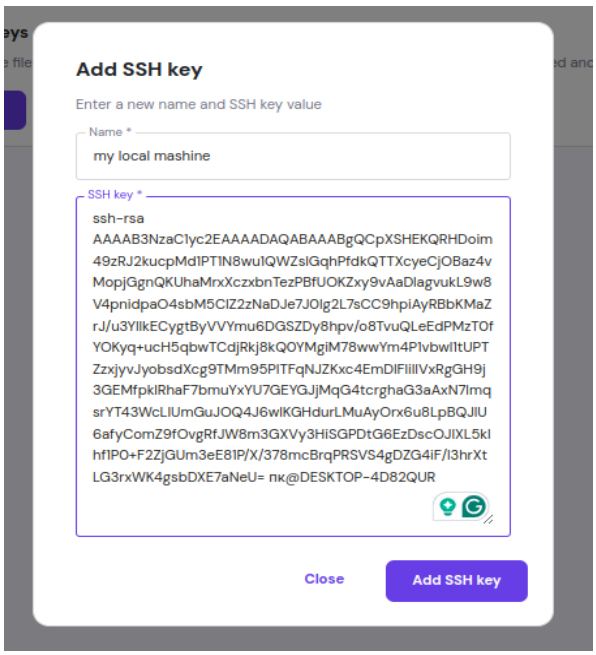

Рисунок 3.6 – Додавання SSH-ключа

Підключитись до сервера і почати налаштовувати його через термінал як показано на рисунку 3.7.

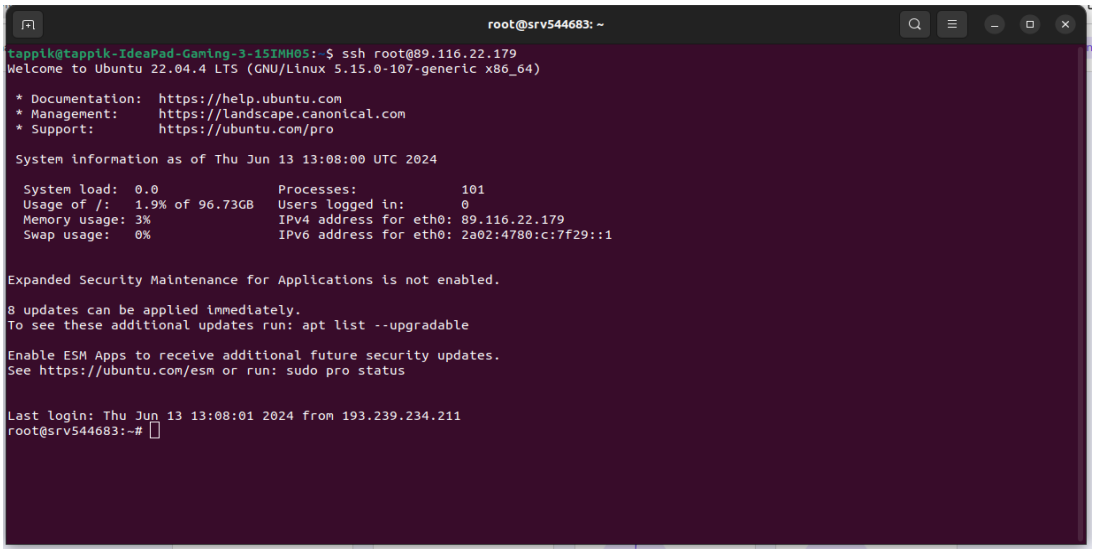

Рисунок 3.7 – Налаштування серверу через термінал

Створити нового юзера з root правами на сервері, бо робити все через root користувача це недоречна практика і це небезпечно з точки зору безпеки:

– додається юзер та ім’я – команда – addusers <yourusername>. У випадку розроблюваного проекту <yourusername> - vova (рис. 3.8);

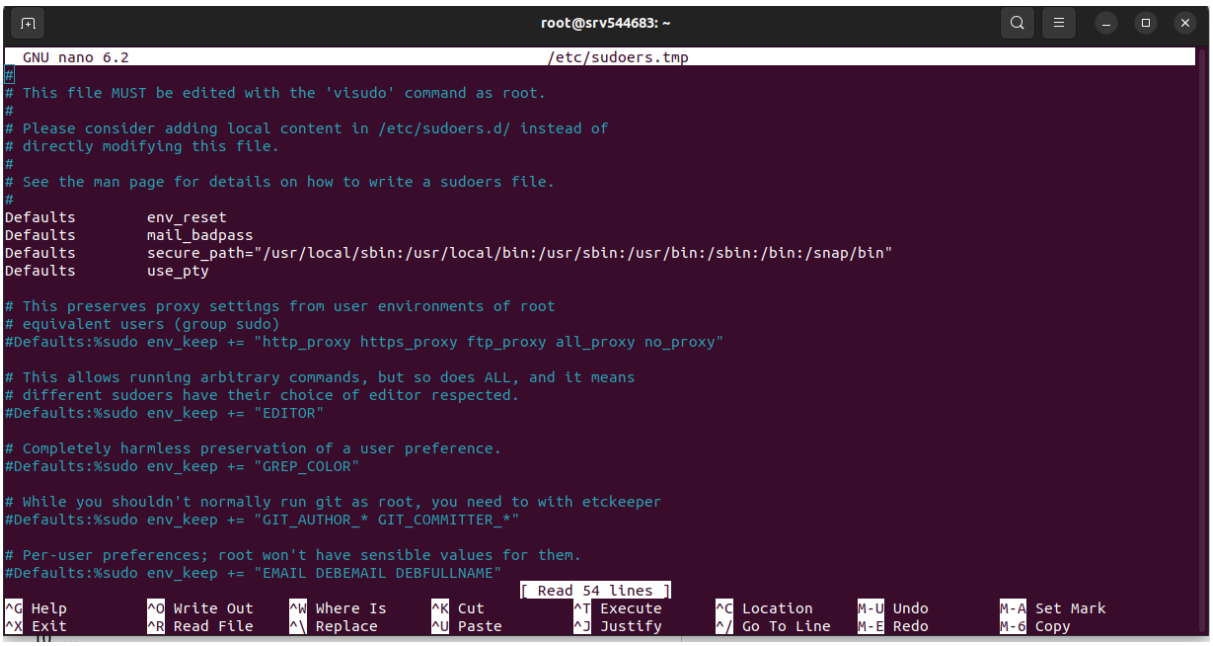

Рисунок 3.8 – Додаткові налаштування серверу через термінал

– даються права адміністратора, тобто такі права як в root користувача:

1) для цього потрібно відкрити файл за допомогою команди – visudo;
2) після цього вийти з файлу;
3) потім написати строку – <yourusername> ALL=(ALL:ALL) ALL.

Тепер всі права на адміністрування додані (рис. 3.9).

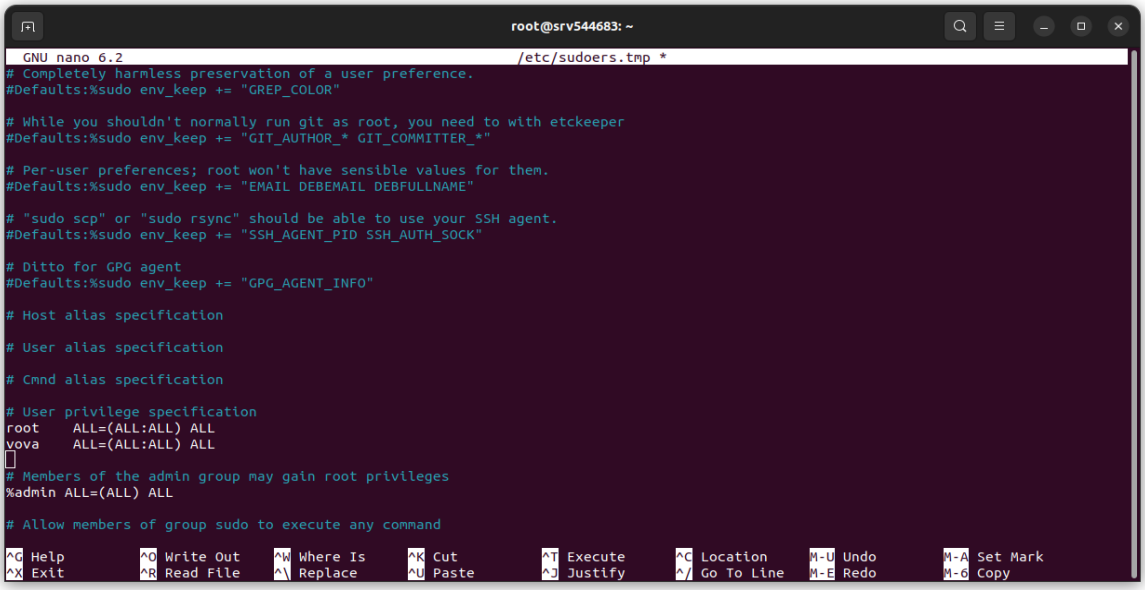

Рисунок 3.9 – Додаткові налаштування серверу через термінал

Після цього файл даний файл зберігається. Після цього відбувається зміна користувача на новоствореного – команда su - <yourusername>.

Оновити всі пакети, які є в операційній системі – команда sudo apt-get update.

Встановити бібліотеки, потрібні для додатку node, npm, npm:

1) встановити curl – команда sudo apt install curl;

2) встановити nvm – команда curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash;

3) скопіювати те, що видав запит на встановлення і вставити в командну строку, адже якщо цього не зробити, nvm не буде працювати (рис. 3.10).

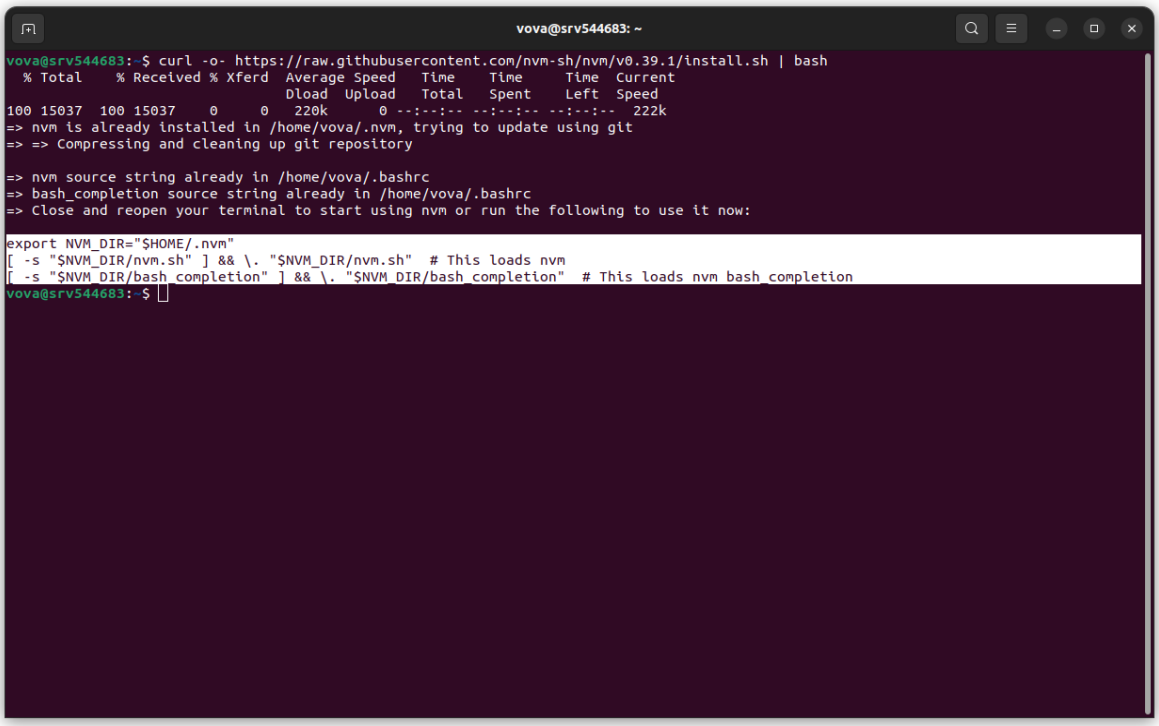

Рисунок 3.10 – Додаткові налаштування серверу через термінал

4) перевірити чи встановити nvm - nvm -v;

5) встановити node.js – nvm install 18.

Тепер сервер потрібно прив’язати до сервісу github, для цього потрібно:

1) згенерувати SSH-ключ – ssh-keygen -o -t rsa -C "ssh@github.com";

2) скопіювати ключ командою –- cat id_rsa.pub (рис. 3.11);

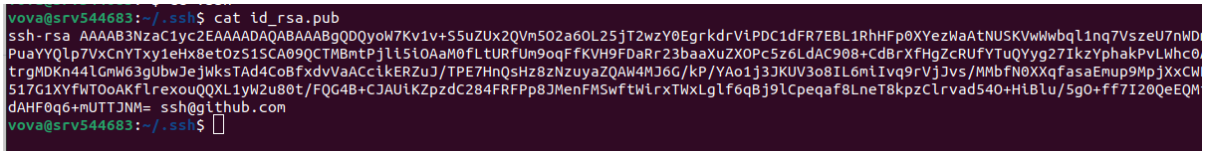

Рисунок 3.11 – Копіювання SSH-ключа

3) налаштувати ключ на платформі github у власному профілі (рис. 3.12).

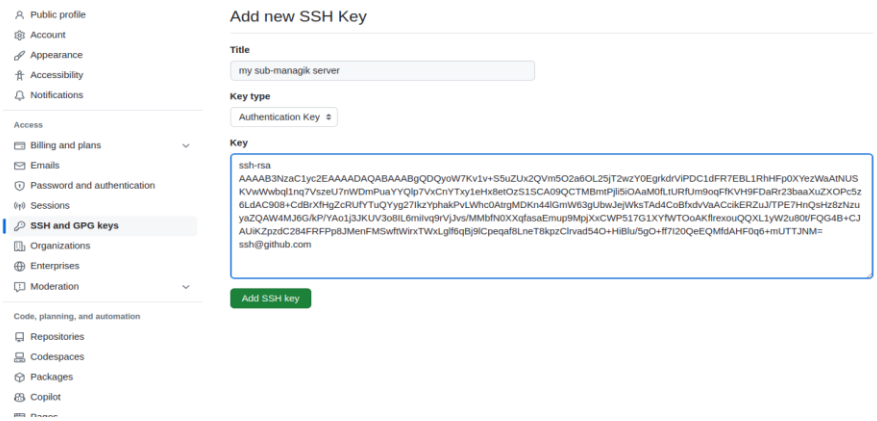

Рисунок 3.12 – Додавання SSH-ключа до профілю github

4) встановити git – sudo apt-get install git-all.

Завантажили git-проект:

1) спочатку треба створити папку – mkdir sub-managik;
2) клонувати проект – git clone git@github.com:nkshn/subs-managik-v2.git.

Вставити postgresql базу даних:

1) оновити всі пакети– команда sudo apt update;
2) завантажити базу даних – sudo apt install postgresql postgresql-contrib;
3) налаштувати базу даних:
   – для початку треба зайти в базу даних – sudo -u postgres psql;
   – створити нового користувача – CREATE ROLE <new_username> WITH LOGIN PASSWORD ‘<yourdbpassword>‘ CREATEDB;
   – створити базу даних – CREATE DATABASE <database_name> OWNER <new_username>;

– дати новому користувачу всі права – GRANT ALL PRIVILEGES ON DATABASE <database_name> TO <new_username>;

– подивитись чи точно нового користувача було створено – \du (рис. 3.13).

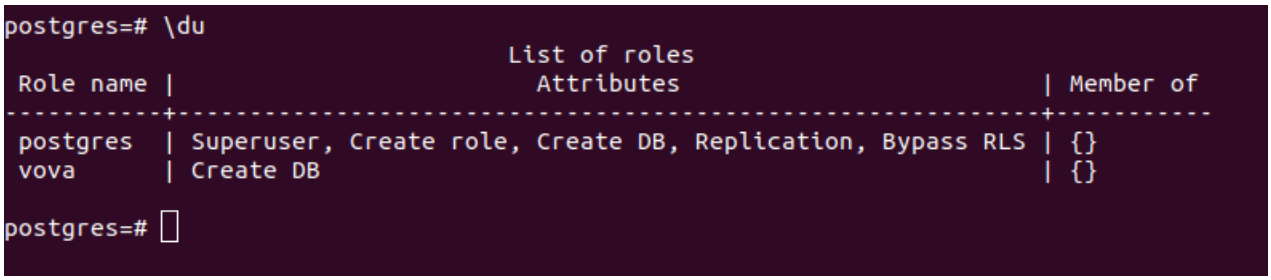

Рисунок 3.13 – Перевірка правильності введених вище операцій

З рисунку 3.13 можна зауважити, що новий користувач тільки з правами Create DB, бо лише такі права йому були видані, цього буде достатньо для поставлених перед проектом задач, а також краще давати менше прав ніж більше з точки зору безпеки.

Також слід переглянути чи база даних створена – \l (рис. 3.14).

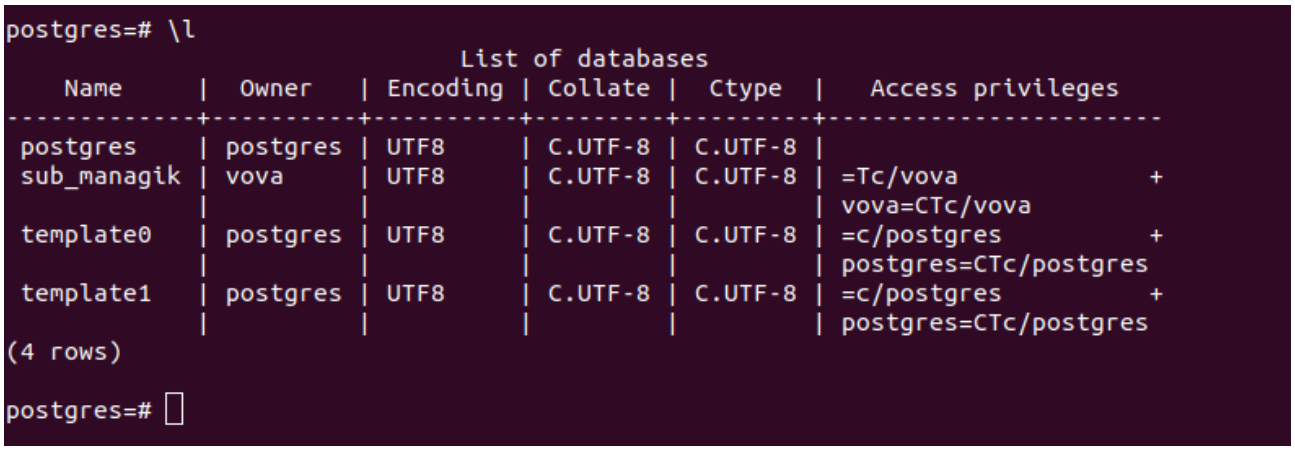

Рисунок 3.14 – Перевірка створення бази даних

Встановити всі необхідні пакети і файли з environment variables, для цього:

– перейти до директорії бекенду – cd backend;

– створити файл .env - nano .env;

– вставити необхідні змінні в цей файл (рис. 3.15);

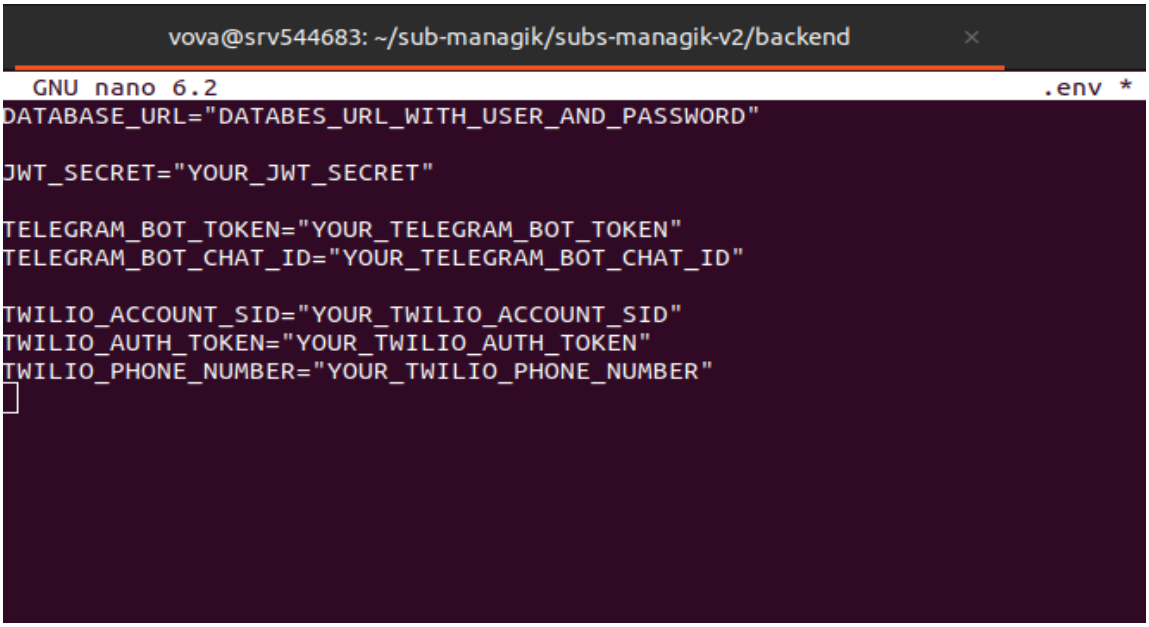

Рисунок 3.15 – Конфігурація файлу зі змінними на бекенді

– перейти до директорії фронтенду – cd frontent;

– створити файл .env – nano .env;

– вставити необхідні змінні в цей файл (рис. 3.16);

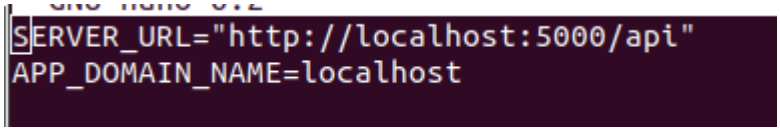

Рисунок 3.16 – Конфігурація файлу зі змінними на фронтенді

1) зайти на бекенд і ввести команду для встановлення всіх бібліотек – npm i;

2) зайти на фронтенд і ввести команду для встановлення всіх бібліотек – npm i.

Налаштувати локально на сервері бази даних:

– перейти до директорії бекенду, команда – cd backend;

– щоб запустити міграції таблиць на базу даних, адже зараз база даних пуста, потрібно виконати команду – npx prisma db push (рис. 3.17);

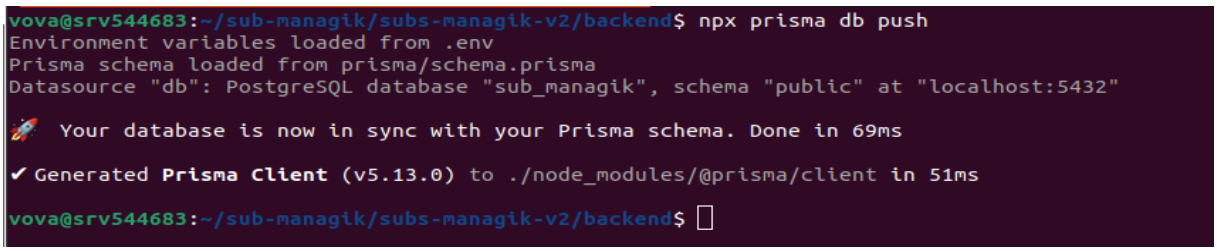

Рисунок 3.17 – Результат виконання команди npx prisma db push

– щоб заповнити базу даних необхідними даними (список доступних сервісів), потрібно запустити seeders, для цього виконати команду – npm run seed (рис. 3.18).

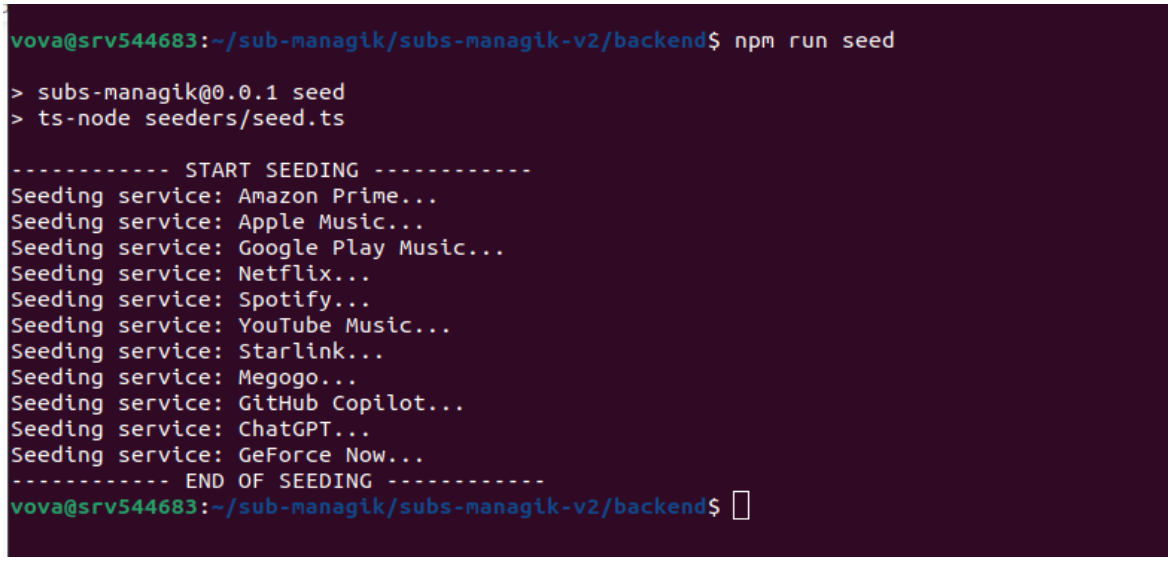

Рисунок 3.18 – Результат виконання команди npm run seed

Збірка проекту:

– перейти до директорії бекенду, команда – cd backend;

– виконати команду для збірки бекенду – npm run build;

– перейти до директорії фронтенду, команда – cd frontend;

– виконати команду для збірки фронтенда – npm run build (рис. 3.19).

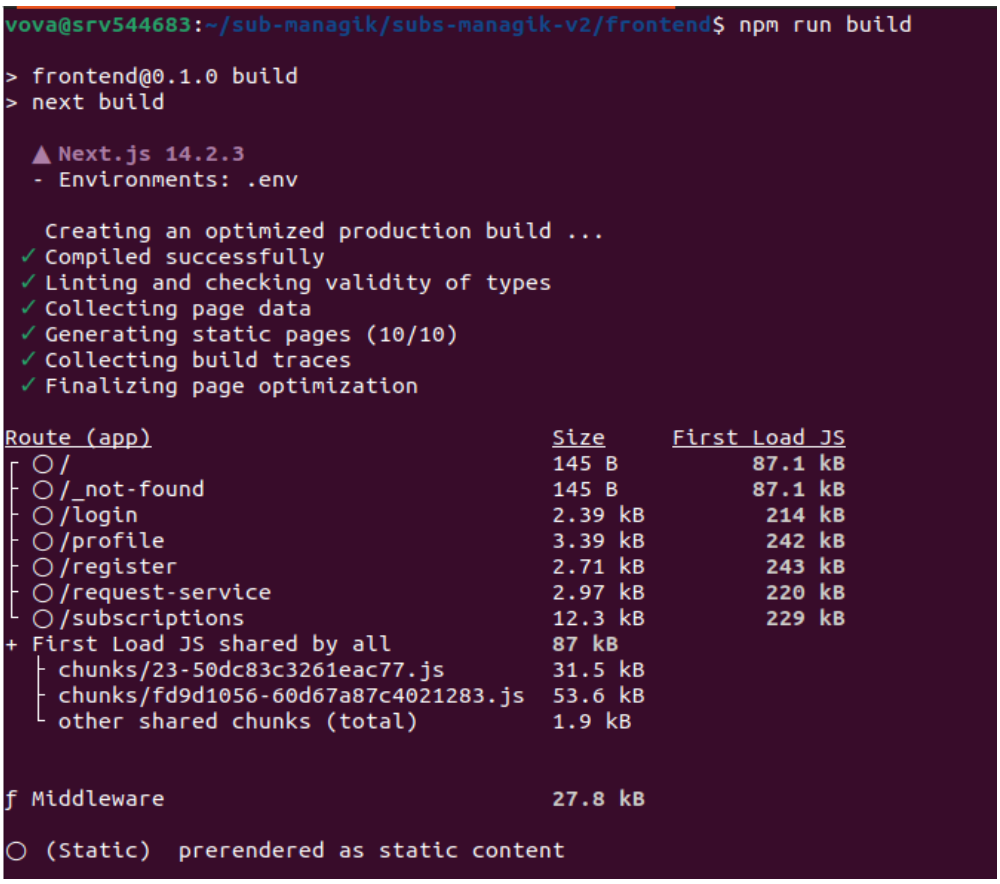

Рисунок 3.19 – Результат виконання команди npm run build

Налаштування запуску проекту за допомогою pm2:

– для початку встановити пакет pm2, який буде займатись процесами, такими як запуск проекту, перезавантаження проекту, коли будь-які зміни відбулись на сервері. Щоб встановити pm2 потрібно виконати команду – npm install pm2 -g;

– щоб запустити бекенд за допомогою pm2 потрібна команда – pm2 start npm--name server-- start (рис. 3.20);

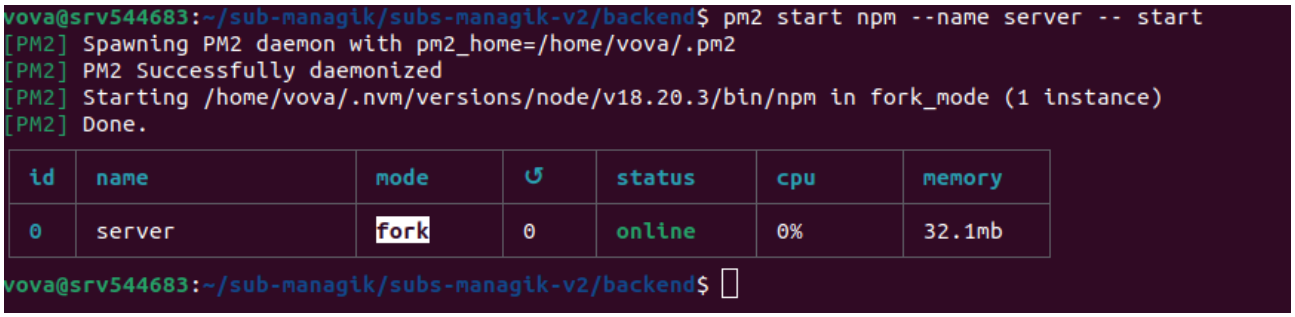

Рисунок 3.20 – Результат виконання команди pm2 start npm --name server -- start

– щоб запустити наш фронтенд за допомогою pm2 нам потрібна команда – pm2 start npm--name client – start (рис. 3.21);

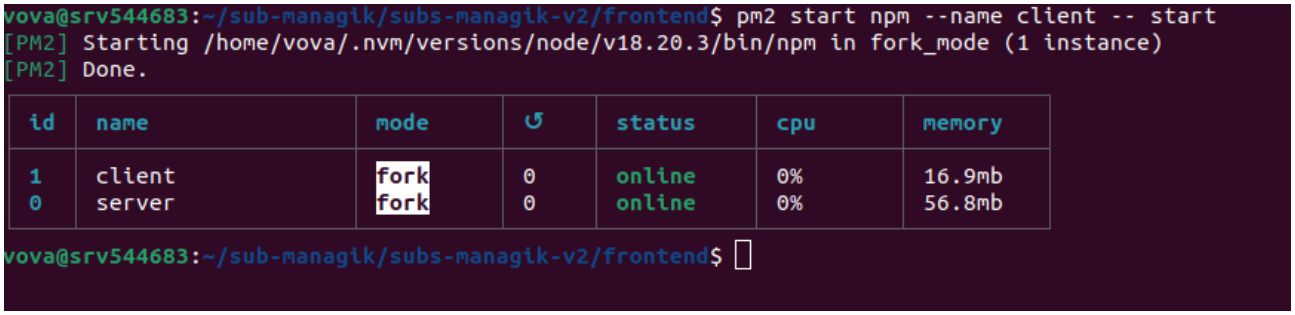

Рисунок 3.21 – Результат виконання команди pm2 start npm --name client – start

– щоб наш pm2 запускав наші процеси автоматично коли буде перезавантажуватись сервер чи коли до коду ми внесемо якісь змінні нам потрібно ввести таку команду – pm2 startup (рис. 3.22);

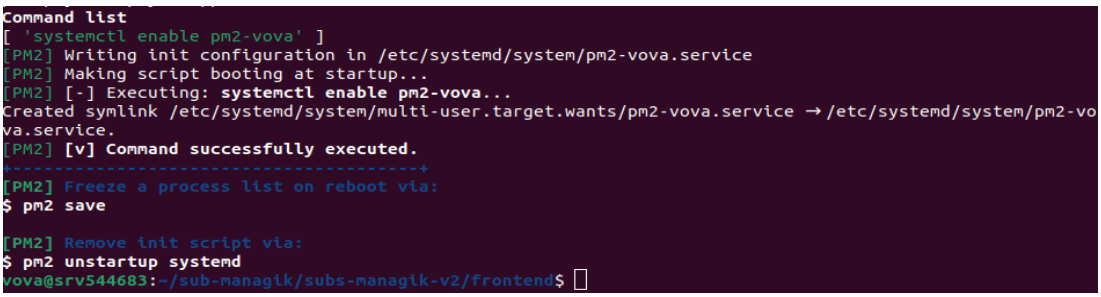

Рисунок 3.22 – Результат виконання команди pm2 startup

– тепер нам треба зберегти наші зміни в pm2, для цього потрібно ввести команду – pm2 save (рис. 3.23);

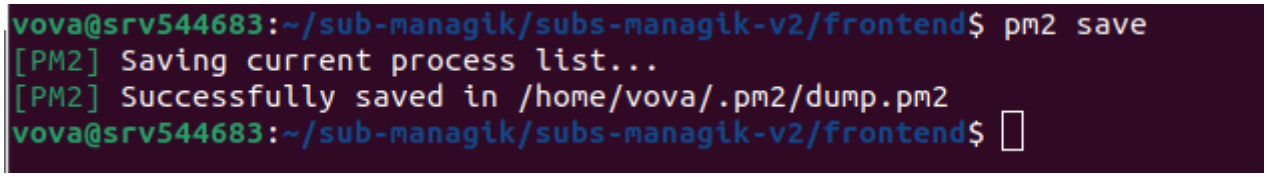

Рисунок 3.23 – Результат виконання команди pm2 save

– також ми можемо ввести список процесів запущений за допомогою команди – pm2 list (рис. 3.24);

```
vova@srv544683:~/sub-managik/subs-managik-v2/frontend$ pm2 list
```

id	name	mode	↺	status	cpu	memory
1	client	fork	0	online	0%	57.4mb
0	server	fork	0	online	0%	57.5mb

```
vova@srv544683:~/sub-managik/subs-managik-v2/frontend$
```

Рисунок 3.24 – Результат виконання команди pm2 list

– також можемо подивитись логи процесів що сервера що бекенду, для цього вводимо команду – pm2 log (рис. 3.25);

```
/home/vova/.pm2/logs/server-out.log last 15 lines:
0|server   | [Nest] 10344  - 06/13/2024, 3:12:17 PM     LOG [RouterExplorer] Mapped {/api/auth/login/ac
ss-token, POST} route +0ms
0|server   | [Nest] 10344  - 06/13/2024, 3:12:17 PM     LOG [RoutesResolver] UserController {/api/user}
+0ms
0|server   | [Nest] 10344  - 06/13/2024, 3:12:17 PM     LOG [RouterExplorer] Mapped {/api/user/profile,
ET} route +1ms
0|server   | [Nest] 10344  - 06/13/2024, 3:12:17 PM     LOG [RouterExplorer] Mapped {/api/user, PUT} ro
e +0ms
0|server   | [Nest] 10344  - 06/13/2024, 3:12:17 PM     LOG [RoutesResolver] SubscriptionController {/a
/user/subscription}: +0ms
0|server   | [Nest] 10344  - 06/13/2024, 3:12:17 PM     LOG [RouterExplorer] Mapped {/api/user/subscrip
on, GET} route +0ms
0|server   | [Nest] 10344  - 06/13/2024, 3:12:17 PM     LOG [RouterExplorer] Mapped {/api/user/subscrip
on, POST} route +1ms
0|server   | [Nest] 10344  - 06/13/2024, 3:12:17 PM     LOG [RouterExplorer] Mapped {/api/user/subscrip
on/:id, PUT} route +0ms
0|server   | [Nest] 10344  - 06/13/2024, 3:12:17 PM     LOG [RouterExplorer] Mapped {/api/user/subscrip
on/:id, DELETE} route +0ms
0|server   | [Nest] 10344  - 06/13/2024, 3:12:17 PM     LOG [RoutesResolver] ServiceController {/api/se
ice}: +0ms
0|server   | [Nest] 10344  - 06/13/2024, 3:12:17 PM     LOG [RouterExplorer] Mapped {/api/service, GET}
oute +1ms
0|server   | [Nest] 10344  - 06/13/2024, 3:12:17 PM     LOG [RoutesResolver] RequestedServiceController
/api/requested-service}: +0ms
0|server   | [Nest] 10344  - 06/13/2024, 3:12:17 PM     LOG [RouterExplorer] Mapped {/api/requested-ser
ce, GET} route +0ms
0|server   | [Nest] 10344  - 06/13/2024, 3:12:17 PM     LOG [RouterExplorer] Mapped {/api/requested-ser
ce, POST} route +0ms
0|server   | [Nest] 10344  - 06/13/2024, 3:12:17 PM     LOG [NestApplication] Nest application successf
ly started +188ms

/home/vova/.pm2/logs/client-out.log last 15 lines:
1|client   |
1|client   |   ▲ Next.js 14.2.3
1|client   |   - Local:        http://localhost:3000
1|client   |
1|client   |  ✓ Starting...
1|client   |  ✓ Ready in 245ms
1|client   |
1|client   | > frontend@0.1.0 start
1|client   | > next start
1|client   |
1|client   |   ▲ Next.js 14.2.3
1|client   |   - Local:        http://localhost:3000
1|client   |
1|client   |  ✓ Starting...
1|client   |  ✓ Ready in 224ms
```

Рисунок 3.25 – Результат виконання команди pm2 log

Налаштування веб сервера в нашому випадку це Nginx. Nginx буде займатись перенаправленням трафіку на наш сервер, та клієнт. Також його можна налаштувати як Load-Balancer і буде виконуватись балансування навантаження між кількома серверними додатками для забезпечення високої доступності та надійності:

– для початку оновляємо;

– скачуємо сам nginx за допомогою команди – sudo apt-get install nginx;

– потім нам треба вернутись на нашу кореневу папку нашого сервера;

– потім відкрити файл nginx конфігу та відредагувати сам файл, щоб це зробити нам треба команда – sudo nano /etc/nginx/sites-available/default;

– повністю очищуєм цей за допомогою комбінації клавіш – ctrl + k (команда буде видаляти построково, тому це треба зробити декілька разів);

– вставляємо код в цей файл (рис. 3.26);

```
  GNU nano 6.2                    /etc/nginx/sites-available/default *
server {
  listen 80;

  location /api {
      proxy_pass http://localhost:5000;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection 'upgrade';
      proxy_set_header Host $host;
      proxy_cache_bypass $http_upgrade;
  }

  location / {
      proxy_pass http://localhost:3000;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection 'upgrade';
      proxy_set_header Host $host;
      proxy_cache_bypass $http_upgrade;
  }
}
```

Рисунок 3.26 – Код який потрібно вставити в файл /etc/nginx/sites-available/default

– зберігаєм наш відредагований файл;

– можеш також перевірити чи синтаксично все в нашому файлі написано за допомогою команди «sudo nginx -t» (рис. 3.27);

```
vova@srv544683:~$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
vova@srv544683:~$
```

Рисунок 3.27 – Результат виконання команди sudo nginx -t

– тепер нам треба зупити постандарту запущений веб-сервер Apache, тому що тепер ми налаштували та будемо використовувати Nginx, це робиться за допомогою команди «udo /etc/init.d/apache2 stop» (рис. 3.28);

```
vova@srv544683:~$ sudo /etc/init.d/apache2 stop
Stopping apache2 (via systemctl): apache2.service.
```

Рисунок 3.28 – Результат виконання команди sudo /etc/init.d/apache2 stop

– тепер нам потрібно перезавантажити наш nginx за допомогою команди «sudo systemctl restart nginx».

Тепер ми все налаштувати і можемо тестувати наш сервер. Для цього ми берем наш API адрес які нас присвоїв Hostinger, в моєму випадку це – 89.116.22.179 і заходим в нашому браузері по посиланню – http://89.116.22.179/ і у нас має з’явитись наший повністю функціональний веб-сервіс (рис. 3.29).

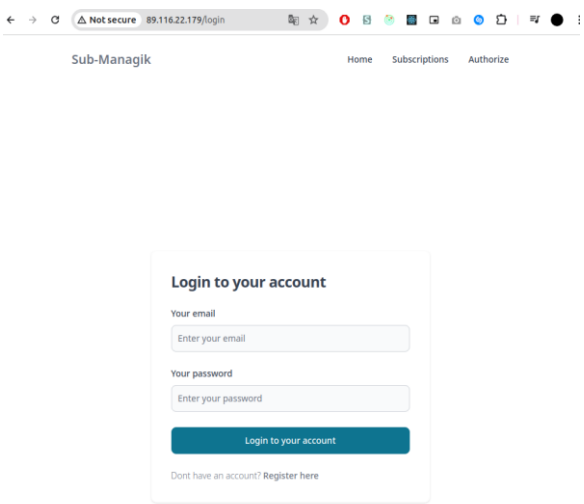

Рисунок 3.29 – Результат успішного публікування нашого веб-сервісу, без домену та SSL сертифікату

Встановлюємо домен:

– для початку треба його купити. Я придбав sub-managik.com;

– потім нам треба правильно налаштувати DNS / Nameservers;

– створюємо новий запис типу: A, Name: @, Points to: тут має бути наш IP адрес нашого серверу (рис. 3.30);

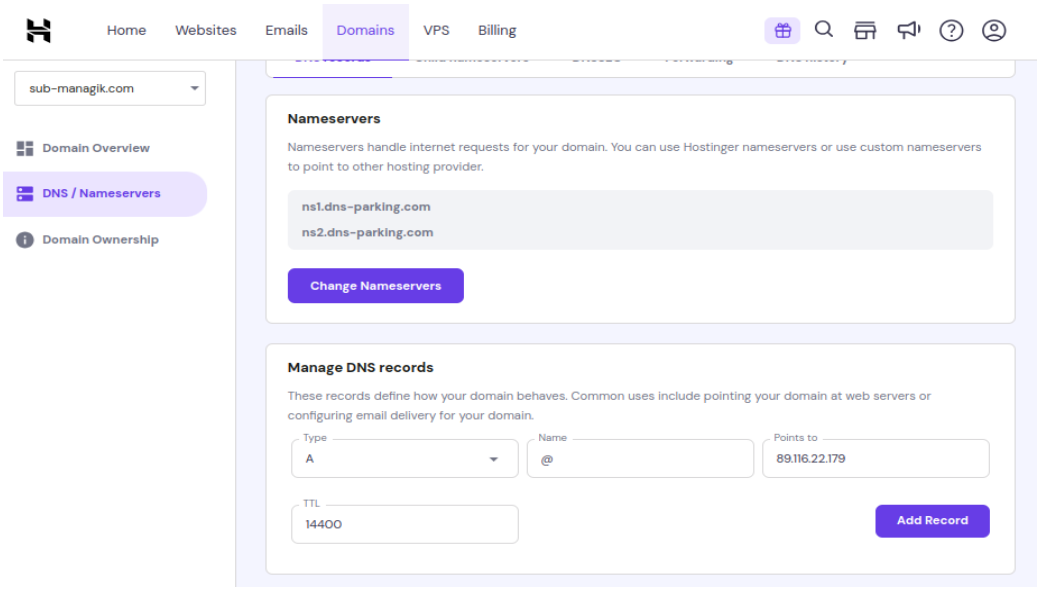

Рисунок 3.30 – Екран налаштування домену для запису Name: @ з IP адресом нашого серверу

– знову робимо те саме але тепер Name буде www (рис. 3.31);

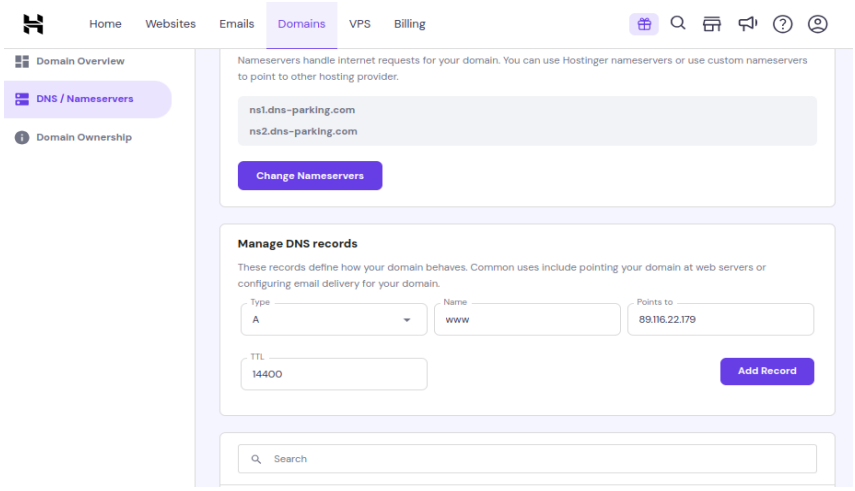

Рисунок 3.31 – Екран налаштування домену для запису Name: www з IP адресом нашого серверу

– тепер нам треба додати наш IPv6 (рис. 3.32);

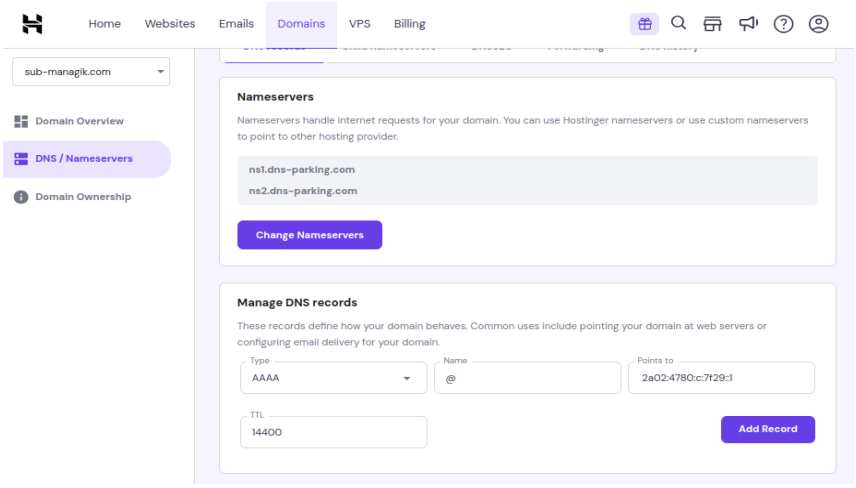

Рисунок 3.32 – Екран налаштування домену де ми додаєм IPv6 адрес нашого серверу

– тепер наш веб-сервіс має доменне ім’я (рис. 3.33).

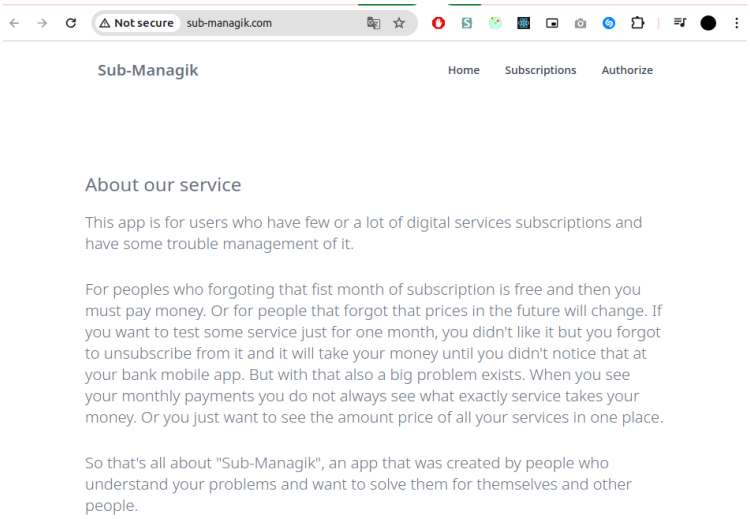

Рисунок 3.33 – Результат успішного налаштування нашого домену

Тепер нам треба додати SSL сертифікат:

– для початку нам потрібно встановити Certbot за допомогою команди «sudo apt install certbot python3-certbot-nginx»;

– потім нам треба оновити нашу конфігурацію Nginx і додати наш домен: server_name sub-managik.com www.sub-managik.com (рис. 3.34);

```
server {
  server_name sub-managik.com www.sub-managik.com;

  location /api {
      proxy_pass http://localhost:5000;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection 'upgrade';
      proxy_set_header Host $host;
      proxy_cache_bypass $http_upgrade;
  }

  location / {
      proxy_pass http://localhost:3000;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection 'upgrade';
      proxy_set_header Host $host;
      proxy_cache_bypass $http_upgrade;
  }
```

Рисунок 3.34 – Вміст файлу з конфігурацією Nginx

– потім нам треба встановити сам сертифікат для нашого домену за допомогою команди «sudo certbot --nginx -d sub-managik.com -d www.sub-managik.com»:

1) після цього треба запустити цю команду для перевірки синтаксису нашої нової конфугурації «sudo nginx -t»;
2) потім перевіряємо за допомогою цієї команди помилки в нашому SSL сертифікаті «sudo certbot renew --dry-run»;
3) після цього наш веб-сервіс буде мати SSL сертифікат (рис. 3.35).

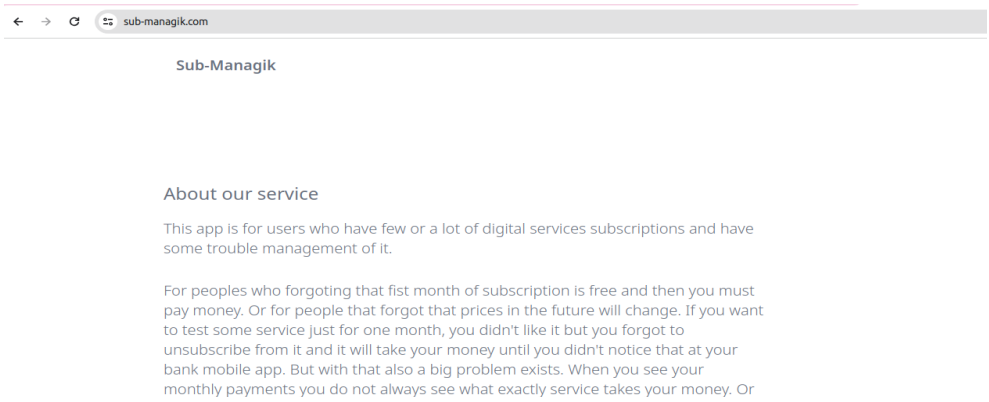

Рисунок 3.35 – Результат успішного публікування нашого веб-сервісу, з налаштованим доменом та SSL сертифікатом

Ось і все, ми успішно задеплоїли наш веб-сервіс, додати йому доменне ім’я, та також додали йому SSL-сертифікат.

Висновки до розділу 3

Підсумовуючи третій розділ, слід зазначити, що практична реалізація веб-сервісу є важливим етапом. Правильне проектування веб-сервісу, допоможе в майбутньому легко розширяти його функціонал або заміняти певний функціонал та підтримувати веб-сервіс. Також використання сучасних технологій забезпечує його надійність, та зручність під час реалізації та користуванні.

Одним із ключових аспектів у розробці є створення бази даних. Це завдання включає аналіз та моделювання даних, розробку схеми бази даних і взаємозв’язків між даними. Правильна організація даних, таблиць та зв’язків між ними даних дозволяє уникнути надлишковості даних та забезпечити їх швидке та коректне оброблення. В майбутньому таку базу даних буде легко підтримувати, та розширяти новими таблицями та даними, що є дуже важливим аспектом.

Не менш важливою складовою є розробка документації для програмного продукту. В нашому випадку це налаштування VPS сервера, налаштування середовища, правильне розгортання веб-сервісу на VPS сервері, підняття бази даних та її налаштування, правильне налаштування доменного імені, та встановлення SSL сертифікату для безпеки передачі даних між клієнтом та сервером і кращого SEO.

ВИСНОВКИ

Основною метою даної роботи було успішне виконання усіх поставлених завдань, якість та своєчасність виконаної роботи.

Головних цілей було досягнуто, в процесі розробки було виконано великий об’єм робіт.

Під час виконання роботи було проведено аналіз предметної області, аби чітко розуміти які переваги та недоліки присутні в застосунках з менеджменту онлайн підписок у конкурентів.

Наступним кроком було проаналізовано сайти конкурентів(Subscriptions – Track Expenses, Subsee, Billbot, Rocket Money – Bills & Budgets). На них виявлено проблеми, яких було уникнено при розробці веб-сервісу з менеджменту онлайн підписок. Серед найбільш поширених проблем важко не відмітити:

– недоступність в Україні;

– не сучасний або ж застарілий дизайн;

– забагато зайвого функціоналу;

– не вчасно надсилаються сповіщення з нагадування про оплату підписки;

– не зручний користувацький досвід;

– погана оптимізація;

Наступним етапом було обрано необхідні для розробки інструменти:

– Next.Js для клієнтської частини;

– Nest.Js для серверної частини;

– Postgresql база даних;

– Prisma для спрощення роботи з базою даних;

– Twilio для надсилання SMS сповіщеннь користувачам;

– Telegram Bot API для надсилання повідомлень адміністратору в телеграм бот.

Обрані інструменти дуже добре працюють в поєднанні між собою, проблем чи підводних каменів при розробці не виникало.

Подальшими діями було налаштовано локально середовище для розробки веб-сервісу. Встановлення NodeJs, та пакетного менеджера npm. Встановлення бази даних Postgresql та встановлення інструменту для спрощення роботи з самою базою даних Prisma.

Наступним кроком була спроектована база даних, з всіма необідними таблицями і взаємозв'язками між таблицями.

Після завершення проектування бази даних, було розроблено серверну частину зі всіма необхідними модулями, такі як: авторизація, автентифікація, додавання/редагування/видалення/читання підписок користувачі, створення користувачем заявок на додавання його бажаного сервісу який відсутній в веб-сервісі, надсилання SMS сповіщень та надсилання повідомлень в телеграм бот.

Після розробки серверної частини, був створений дизайн веб-сервісу та розроблену клієнтської частини за допомогою Next.Js з використанням Tailwindcss для швидкого і простого написання стилів та Flowbite-React для використання базовий готових UI компонентів. За допомогою axois та бібліотеки tanstack/react-query виконуються запити на наш сервер для отримання даних.

Наступним крок був деплой. В ході цього кроку було вибрало VPS провайдер. Підготовлене середовище на вибраному та придбаному VPS сервері для подальшого публікування коду. Всталовлено та налаштовано базу даних. Встановлення правильних env зміних. Скопільовано клієнтську та серверну частину веб-сервісу. Налаштовано веб-сервер Nginx для переадресації трафіку. Придбано домене ім'я та правильно налаштовене. Встановленно SSL сертифікат.

Розробка велась легко, гармонічно та без будь яких проблем. В процесі розробки глухих кутів не виникало.

В результаті виконання кваліфікаційної роботи бакалавра, було створено повноцінний веб-сервіс. Було докладено максимальних зусиль для його валідного функціонування та легкості в експлуатації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wardana K. UX Case Study: Subscription Management App. 2022. URL: https://medium.com/@krisnawrdn/ux-case-study-subscription-management-app-9b352e596c7 (дата звернення: 15.02.2024).

2. Мобільний додаток Subscriptions – Track Expenses. URL: https://apps.apple.com/ua/app/subscriptions-track-expenses/id1577082754?l=en (дата звернення: 20.02.2024).

3. Мобільний додаток Subsee. URL: http://surl.li/uoxhx (дата звернення: 20.02.2024).

4. Мобільний додаток Billbot. URL: https://apps.apple.com/ru/app/billbot-subscription-manager/id1523543874?l=en (дата звернення: 20.02.2024).

5. Мобільний додаток Rocket Money – Bills & Budgets. URL: https://apps.apple.com/us/app/rocket-money-bills-budgets/id1130616675 (дата звернення: 20.02.2024).

6. Документація React. URL: https://react.dev/learn (дата звернення: 10.03.2024).

7. Документація Next.js. URL: https://nextjs.org/docs (дата звернення: 10.03.2024).

8. Документація React Hook Form. URL: https://react-hook-form.com/get-started (дата звернення: 20.03.2024).

9. Документація Yup. URL: https://github.com/jquense/yup (дата звернення: 20.03.2024).

10. Документація Tailwind CSS. URL: https://tailwindcss.com/docs (дата звернення: 20.03.2024).

11. Документація TanStack Query. URL: https://tanstack.com/query/latest/docs/framework/react/overview (дата звернення: 05.04.2024).

12. Документація Flowbite. URL: https://flowbite-react.com/docs/getting-started/introduction (дата звернення: 05.04.2024).

13. Документація date-fns. URL: https://date-fns.org/docs/Getting-Started (дата звернення: 05.04.2024).

14. Документація Axios. URL: https://axios-http.com/docs/intro (дата звернення: 10.04.2024).

15. Документація Nest.JS. URL: https://docs.nestjs.com (дата звернення: 20.04.2024).

16. Документація Prisma. URL: https://www.prisma.io/docs/getting-started (дата звернення: 21.04.2024).

17. Документація Twilio Messaging. URL: https://www.twilio.com/docs/messaging (дата звернення: 10.05.2024).

18. Документація Node Telegram Bot API. URL: https://www.npmjs.com/package/node-telegram-bot-api (дата звернення: 15.05.2024).

ДОДАТКИ

Додаток А

Структура файлів веб-сервісу

```
├── backend/                    # Root folder for backend
│   ├── prisma                  # Folder with prisma schema of DB
│   ├── seeders                 # Folder with seeders for DB
│   ├── src/
│   │   ├── auth/               # Authentication module
│   │   │   ├── ....            # Other authentication module files and folders
│   │   ├── notifications
│   │   │   ├── ....            # Other notifications module files and folders
│   │   ├── requested-service
│   │   │   ├── ....            # Other requested-service module files and folders
│   │   ├── service
│   │   │   ├── ....            # Other service module files and folders
│   │   ├── subscription
│   │   │   ├── ....            # Other subscription module files and folders
│   │   ├── telegram
│   │   │   ├── ....            # Other telegram module files and folders
│   │   ├── user
│   │   │   ├── ....            # Other user module files and folders
│   │   ├── app.module.ts       # Root module of the application
│   │   ├── main.ts             # Entry point of the application
└── frontend
    ├── src
    │   ├── api
    │   │   ├── ....            # Files required for making API requests
    │   ├── app
    │   │   ├── login
    │   │   │   ├── ....        # Other login page files and folders
    │   │   ├── profile
    │   │   │   ├── ....        # Other profile page files and folders
    │   │   ├── register
    │   │   │   ├── ....        # Other register page files and folders
    │   │   ├── request-service
    │   │   │   ├── ....        # Other request Service page files and folders
    │   │   └── subscriptions
    │   │   │   ├── ....        # Other subscriptions page files and folders
    │   │   ├── not-found.tsx   # Not found page in App
    │   │   ├── page.tsx        # Entry page in App
    │   ├── components
    │   │   ├── ....            # Other UI components files and folders
```

```
│   ├── config
│   │   ├── ....                # Other congif files used in app
│   ├── constants
│   │   ├── ....                # Other constants files used in app
│   ├── hooks
│   │   ├── ....                # Other hook files
│   ├── services
│   │   ├── ....                # Other services files used in app
│   └── types
│   │   ├── ....                # Other types files used in app
│   ├── middleware.ts           # Main middleware of App
├── ....                        # Other frontend file
```

Додаток Б

Демонстрація коду логін, реєстрація

```
import {
 BadRequestException,
 Injectable,
 NotFoundException,
 UnauthorizedException
} from "@nestjs/common"
import { JwtService } from "@nestjs/jwt"
import { verify } from "argon2"
import { Response } from "express"
import { UserService } from "src/user/user.service"
import { AuthDto } from "./dto/auth.dto"
import { RegisterDto } from "./dto/register.dto"
import { ConfigService } from "@nestjs/config"

@Injectable()
export class AuthService {
 EXPIRE_DAY_REFRESH_TOKEN = 10
 REFRESH_TOKEN_NAME = "refreshToken"

 constructor(
   private jwt: JwtService,
   private userService: UserService,
   private configService: ConfigService
 ) { }

 async login(dto: AuthDto) {
   // eslint-disable-next-line @typescript-eslint/no-unused-vars
   const { password, ...user } = await this.validateUser(dto)

   const tokens = this.generateTokens(user.id)

   return {
     user,
     ...tokens
   }
 }

 async register(dto: RegisterDto) {
   const foundUserByEmail = await
this.userService.getByEmail(dto.email)
   if (foundUserByEmail) throw new BadRequestException({
     type: "email",
```

```
          message: "User with such email already exists"
        })

        const foundUserByPhone = await this.userService.getByPhone(dto.phone)
        if (foundUserByPhone) throw new BadRequestException({
          type: "phone",
          message: "User with such phone already exists"
        })

        if (dto.password !== dto.repeatPassword) throw new BadRequestException({
          type: "repeatPassword",
          message: "Passwords must match"
        })

        // eslint-disable-next-line @typescript-eslint/no-unused-vars
        const { password, ...user } = await this.userService.create(dto)

        const tokens = this.generateTokens(user.id)

        return {
          user,
          ...tokens
        }
      }

      async getNewTokens(refreshToken: string) {
        const result = this.jwt.verify(refreshToken)
        if (!result) throw new UnauthorizedException("Invalid refresh token")

        // eslint-disable-next-line @typescript-eslint/no-unused-vars
        const { password, ...user } = await this.userService.getById(result.id)
        if (!user) throw new NotFoundException("User not found")

        const tokens = this.generateTokens(user.id)

        return {
          user,
          ...tokens
        }
      }

      addRefreshTokenToResponse(res: Response, refreshToken: string) {
```

```
        const expiresIn = new Date()
        expiresIn.setDate(expiresIn.getDate()                                    +
this.EXPIRE_DAY_REFRESH_TOKEN)

        res.cookie(this.REFRESH_TOKEN_NAME, refreshToken, {
          httpOnly: true,
          domain: this.configService.get<string>("DOMAIN_NAME"),
          expires: expiresIn,
          secure: true,
          sameSite:      this.configService.get<string>("NODE_ENV")      ===
"production" ? "lax": "none"
        })
      }

      removeRefreshTokenFromResponse(res: Response) {
        res.cookie(this.REFRESH_TOKEN_NAME, "", {
          httpOnly: true,
          domain: this.configService.get<string>("DOMAIN_NAME"),
          expires: new Date(0),
          secure: true,
          sameSite:      this.configService.get<string>("NODE_ENV")      ===
"production" ? "lax": "none"
        })
      }

      private generateTokens(userId: string) {
        const data = { id: userId }

        const accessToken = this.jwt.sign(data, {
          expiresIn: "1h"
        })

        const refreshToken = this.jwt.sign(data, {
          expiresIn: "14d"
        })

        return { accessToken, refreshToken }
      }

      private async validateUser(dto: AuthDto) {
        const user = await this.userService.getByEmail(dto.email)

        if (!user) throw new NotFoundException({
          type: "email",
          message: "User not found"
```

```
      })

      const isValid = await verify(user.password, dto.password)

      if (!isValid) throw new UnauthorizedException({
        type: "password",
        message: "Invalid password",
      })

      return user
    }
  }
```

Додаток В

Повний код який надсилає SMS сповіщення всім користувачам

```
import { Injectable, Logger } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { Cron, CronExpression } from '@nestjs/schedule';
import { addMonths, format, isBefore, isToday, parseISO, setMonth, setYear } from 'date-fns';
import { UserService } from 'src/user/user.service';
import { Twilio } from 'twilio';

@Injectable()
export class NotificationsService {
 private readonly logger = new Logger(NotificationsService.name);
 private twilioClient: Twilio;

 constructor(
   private userService: UserService,
   private configService: ConfigService,
 ) {
   const accountSid = this.configService.get<string>('TWILIO_ACCOUNT_SID');
   const authToken = this.configService.get<string>('TWILIO_AUTH_TOKEN');
   this.twilioClient = new Twilio(accountSid, authToken);
 }

 @Cron(CronExpression.EVERY_DAY_AT_9PM)
 async handleEveryDayAt21Cron() {
   this.logger.log('Cron job every day at 21:00 started...');

   const users = await this.userService.getAllWithSubscriptionsAndService();

   let totalUsers = 0;
   let totalSubscriptions = 0;
   let successCount = 0;
   let failureCount = 0;

   for (const user of users) {
     if (user.subscriptions.length > 0) {
       totalUsers++;
     }
```

```
          for (const subscription of user.subscriptions) {
            const                        adjustedDate                        =
this.adjustNextPaymentDate(subscription.nextPaymentAt.toISOString());user

            if (isToday(adjustedDate) && subscription.isNotifying) {
              this.logger.log(`Sending SMS notification to user ${user.name}
to  ${user.phone}  for  service  ${subscription.service.fullName}  with  price
${subscription.price}$...`);
              totalSubscriptions++;

              try {
                await this.sendSubscriptionNotificationSMS(
                  user.phone,
                  subscription.service.fullName,
                  adjustedDate
                );
                successCount++;
              } catch (error) {
                failureCount++;
              }
            }
          }
        }
        this.logger.log(`Cron job every day at 21:00 finished. Total users
notified: ${totalUsers}, total subscriptions: ${totalSubscriptions}. Success:
${successCount}, Failures: ${failureCount}`);
      }
      /**
       * This function is used to adjust the next payment date to the current
month and year.
       * @param nextPaymentAt string - the next payment date
       * @returns Date object
       */
      private adjustNextPaymentDate(nextPaymentAt: string): Date {
        const now = new Date();
        let nextPaymentDate = parseISO(nextPaymentAt);
        nextPaymentDate = setYear(nextPaymentDate, now.getFullYear());
        nextPaymentDate = setMonth(nextPaymentDate, now.getMonth());
        if (isBefore(nextPaymentDate, now) && !isToday(nextPaymentDate)) {
          nextPaymentDate = addMonths(nextPaymentDate, 1);
        }
        return nextPaymentDate;
      }
```

```
      private async sendSubscriptionNotificationSMS(phone: string,
serviceName: string, nextPaymentDate: Date) {
        const formattedDate = format(nextPaymentDate, 'dd.MM.yyyy');
        const message = `Reminder: Your subscription to ${serviceName} is due
on ${formattedDate}. So today will be payment day.`;

        try {
          await this.twilioClient.messages.create({
            body: message,
            from: this.configService.get<string>('TWILIO_PHONE_NUMBER'),
            to: phone,
          });
          this.logger.log(`Succesfully sent SMS to ${phone}: ${message}`);
        } catch (error) {
          this.logger.error(`Failed to send SMS to ${phone}:
${error.message}`);
          throw error;
        }
      }
}
```