МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ЛЕСІ УКРАЇНКИ
Кафедра комп'ютерних наук та кібербезпеки

САНЬКО ДАНИЛО АНДРІЙОВИЧ

РОЗРОБКА СИСТЕМИ КЕРУВАННЯ ПІДПИСКАМИ В TELEGRAM

Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки та інформаційні технології
Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр»

Науковий керівник:
БУЛАТЕЦЬКА ЛЕСЯ ВІТАЛІЇВНА,
кандитат фізико-математичних наук,
доцент кафедри комп'ютерних наук та
кібербезпеки

РЕКОМЕНДОВАНО ДО
ЗАХИСТУ
Протокол №_____
засідання кафедри_____
від_____2025 р.
Завідувач кафедри
(_____) _____
(підпис) ПІБ

ЛУЦЬК – 2025

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗГНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ ТА ТЕРМІНІВ

Git	Система управління версіями
MERN	Технологічний стек який складається з технологій MongoDB, Express.js, React.js, Node.js
Telegram	Месенджер для обміну повідомленнями, файлами та дзвінками
JS	Мова програмування, що використовується для створення інтерактивних вебсторінок, обробки подій та взаємодії з користувачем у браузері
Telegraf	Фреймворк для створення ботів у Telegram на JavaScript/Node.js, який надає простий та зручний інтерфейс для роботи з Telegram Bot API
MongoDB	Документно-орієнтована база даних, яка зберігає дані у форматі JSON-подібних документів (BSON)
Mongoose	Об’єктно-орієнтована бібліотека для роботи з MongoDB в Node.js
USDT TRC20	Токен Tether (USDT), випущений на блокчейні TRON відповідно до стандарту TRC-20
Хеш	Результат перетворення даних будь-якого розміру у фіксований рядок символів за допомогою хеш-функції. Хеші широко використовуються для перевірки цілісності даних, створення унікальних ідентифікаторів та шифрування інформації
Фреймворк	Набір готових інструментів, бібліотек і правил, що спрощують розробку програмного забезпечення, надаючи структуровану основу для побудови доатків

ВСТУП

Актуальність теми. Сучасний світ стрімко рухається в бік автоматизації процесів, що значно полегшує ведення бізнесу в різних сферах, включаючи інформаційний бізнес. У зв’язку з цим зростає попит на цифрові рішення, які спрощують управління базами користувачів, забезпечують автоматичну обробку оплат, а також надають швидкий доступ до інформаційних ресурсів. Одним із таких рішень є телеграм бот, розроблений для автоматизації дій адміністратора інформаційних ресурсів.

Метою даної роботи є дослідження розробки телеграм бота, який здатен здійснювати автоматичне управління інформаційними послугами, обробляти запити користувачів, керувати доступом до матеріалів та проводити оплату через інтеграцію з платіжними сервісами та веб-додатку, який дозволяє керувати користувачами з веб-інтерфейсу.

Для досягнення поставленої мети було поставлено наступні завдання:

- провести аналіз теоретичних знань та основних термінів, пов’язаних із розробкою ботів для месенджерів;
- здійснити порівняння подібних телеграм ботів, які використовуються для інформаційних послуг;
- визначити оптимальне середовище та інструменти для розробки бота;
- розробити телеграм бота, який дозволить автоматизувати роботу з користувачами та управління контентом;
- дослідити методи, які дозволять найбільш ефективно та швидко розробити адміністративну панель у веб-інтерфейсі;
- розробити вебдодаток для керування користувачами.

Об’єкт дослідження: технології розробки та принципи роботи з ботами для месенджерів та та вебзастосунку.

Предмет дослідження: методи автоматизації обробки підписок, реалізація функціоналу Telegram-бота та клієнт-серверна взаємодія у вебзастосунку для адміністрування користувачів.

Публікації:

Санько Д. Розробка Telegram-бота і застосунку на технологічному стеці MERN для керування підписками та користувачами інформаційних ресурсів. *Проблеми комп'ютерних наук, програмного моделювання та безпеки цифрових систем*.: матеріали II міжнар. науково-практ. конф. Луцьк, 9–11 червн. 2025 р. Луцьк, 2025. С. 94-95.

РОЗДІЛ 1.
ТЕХНОЛОГІЧНІ ПІДХОДИ СТВОРЕННЯ TELEGRAM-БОТІВ

1.1. Методи розробки Telegram-ботів

Розробка Telegram-ботів є комплексним процесом, який включає вибір інструментів, фреймворків та підходів для створення ефективної системи взаємодії. Для цього використовуються різні методи, що залежать від потреб проєкту, обсягу функціональності та доступних ресурсів. Основні методи розробки Telegram-ботів охоплюють використання Telegram Bot API, фреймворків для спрощення інтеграції та програмні патерни для забезпечення гнучкості та масштабованості.

Telegram Bot API є базовим інструментом для створення ботів. Це RESTful API, який дозволяє програмно взаємодіяти з платформою Telegram. За допомогою API можна обробляти повідомлення, керувати чатами, створювати меню, відправляти мультимедійні файли та налаштовувати інтерактивні кнопки. API забезпечує універсальність і підтримує широкий спектр мов програмування. Однак, використання API напряму може бути доволі складним, тому розробники часто обирають фреймворки, які спрощують інтеграцію та зменшують обсяг рутинного коду. [7]

Одним із популярних інструментів для створення Telegram-ботів є фреймворк Telegraf.js, який базується на мові програмування JavaScript і працює у середовищі Node.js. Цей фреймворк дозволяє організувати обробку подій через проміжний компонент (middleware), що забезпечує зрозумілу та модульну структуру коду. Наприклад, розробник може легко визначати обробники для команд, текстових повідомлень або взаємодій з кнопками. [6]

Крім Telegraf.js, для розробки ботів також використовуються інші інструменти, такі як python-telegram-bot для Python чи telebot для мови Go. Вибір конкретного фреймворку залежить від технічної бази команди розробників, мовної платформи проєкту та вимог до продуктивності системи.

Методи розробки ботів також включають інтеграцію баз даних, які необхідні для збереження інформації про користувачів, підписки чи інші важливі дані. У нашому випадку використовується база даних MongoDB у поєднанні з бібліотекою Mongoose для взаємодії з даними. Такий підхід забезпечує зручну роботу з моделями, а також підтримку складних запитів і фільтрів.

Розробка Telegram-ботів часто включає використання багатокрокових сценаріїв. Наприклад, під час додавання нового користувача або редагування підписки потрібно запитати в користувача кілька послідовних даних. Цей процес можна організувати за допомогою механізмів сцен, які доступні у фреймворку Telegraf.js. Сцени дозволяють зберігати стан діалогу, щоб користувач міг послідовно вводити дані, а бот – обробляти їх на кожному кроці.

Важливим аспектом розробки є тестування та налагодження. Для цього застосовуються як локальні сервери для моделювання взаємодії, так і сторонні інструменти, які дозволяють перевірити поведінку бота в реальних умовах. Під час тестування використовуються методи автоматизації, зокрема для перевірки обробки подій чи роботи з базою даних.

Таким чином, методи розробки Telegram-ботів включають використання Telegram Bot API, сучасних фреймворків, інтеграцію баз даних та впровадження сценаріїв для забезпечення ефективної взаємодії з користувачем. Ці методи дозволяють створити продукт, який відповідає сучасним вимогам і може бути адаптований до майбутніх змін

1.2. Блокчейн TRON і обробка платежів у системі стандарту TRC-20

TRON – це високопродуктивний публічний блокчейн, орієнтований на масове використання в сфері розваг, децентралізованих додатків (DApps), стейблкоїнів та цифрових активів.

TRON дозволяє валідувати транзакції за допомогою публічного API Tronscan, а саме:

- перевірити статус транзакції за TXID;
- отримати історію транзакцій адреси;
- фільтрувати транзакції за типом (вхідні, вихідні);
- знайти події контрактів (наприклад, Transfer токена);
- перевірити, чи транзакція була успішно підтверджена.

Тобто можна перевірити, чи існує транзакція, суму, токен і адресу отримувача.

Серед переваг TRON можна зазначити його швидкодію, оскільки блоки створюються кожні 3 секунди, мінімальні комісії, які часто або близькі до нуля, або є символічними. Tronskan не потребує зайвої ідентифікації, навідміну від, наприклад, Ethereum APIs та повертає результат у форматі JSON, що дозволяє легко обробляти дані.[15]

Використання TRON + Tronscan API для перевірки платежів – це швидкий, недорогий і технічно простий варіант для невеликих або середніх проєктів, який ідеально може підійти для подібних проєктів. Якщо потрібна більша децентралізація, події в реальному часі або кастомна логіка – краще дивитися в бік Ethereum або BSC з використанням власних нод або сервісів типу Alchemy/Infura/Moralis, але це значно збільшить час на розробку та вартість проєкту.

1.3. Підходи до розв’язання задач автоматизації в Telegram-ботах

Telegram-боти створюються для автоматизації взаємодії між користувачами та сервісами, виконання рутинних завдань, а також інтеграції з різноманітними зовнішніми платформами. Для цього розробники використовують кілька ключових підходів, які базуються на подієво-орієнтованій архітектурі, плануванні завдань та інтеграції зовнішніх API.

1.3.1. Подієво-орієнтована архітектура

Цей підхід базується на тому, що бот реагує на події, ініційовані користувачем, наприклад:

- надсилання текстового повідомлення або команди;
- взаємодія з кнопками в повідомленнях (callback-запити);
- відправка мультимедійних файлів (фото, відео, документи);
- реакція на певні ключові слова або шаблони.

Алгоритм роботи подієво-орієнтованого підходу:

- користувач відправляє запит боту;
- бот обробляє запит за допомогою обробника відповідного типу (наприклад, команда /start викликає стартову функцію);
- на основі отриманого запиту бот виконує дію: звернення до бази даних, надсилання повідомлення чи виконання зовнішнього запиту.

У цьому підході важливо коректно визначити сценарії взаємодії, щоб бот міг ефективно обробляти повідомлення. Наприклад, у бібліотеці Telegraf.js для цього використовуються "мідлвейр" (middleware) та модуль "сцени", які дозволяють створювати послідовність дій для складних сценаріїв.

1.3.2. Планування завдань

Планування завдань дозволяє автоматизувати виконання задач у певний час або з певною періодичністю. Цей підхід використовується для:

- відправки нагадувань про завершення підписки;
- перевірки статусів підписок і видалення користувачів із каналів після закінчення доступу;
- регулярного оновлення даних із зовнішніх API (наприклад, обмінних курсів або погодних даних).

Технічна реалізація:

- у Node.js використовуються пакети для планування завдань, як-от node-cron або agenda;
- розробник задає розклад виконання завдання у вигляді cron-виразів або часових інтервалів;
- у визначений час завдання виконується, наприклад, перевіряється стан підписок у базі даних.

Цей підхід дозволяє створити стабільну та передбачувану логіку виконання завдань, яка працює незалежно від активності користувачів.

1.3.3. Інтеграція із зовнішніми API

Telegram-боти можуть виконувати широкий спектр задач завдяки взаємодії з іншими сервісами через їхні API. Це відкриває можливості для автоматизації, зокрема:

- підключення платіжних систем (Stripe, LiqPay) для обробки оплат;
- інтеграція з CRM-системами для управління користувачами;
- робота з базами даних через ORM (наприклад, Sequelize для SQL-баз або Mongoose для MongoDB).

Розглянемо приклад інтеграції. Користувач надсилає команду для перевірки свого статусу підписки, після чого бот викликає відповідну функцію, яка звертається до бази даних і отримує необхідні дані. Отриманий результат повертається користувачу у вигляді повідомлення.

1.3.4. Використання сценаріїв і багатокрокової логіки

Для реалізації складних сценаріїв, таких як додавання нового користувача чи зміна підписки, Telegram-боти можуть використовувати багатокрокову логіку, яку розбивають на окремі етапи:

- користувач вводить початкові дані (наприклад, ім’я чи ID);
- бот перевіряє коректність введених даних і запитує додаткову інформацію;
- після введення всіх даних виконується підсумкове завдання (оновлення бази даних, реєстрація користувача тощо).

Бібліотека Telegraf.js дозволяє реалізувати багатокрокові сценарії через модуль "сцени", що спрощує створення послідовностей дій і забезпечує легке керування станом користувача.

Для ефективної роботи Telegram-бота важливо комбінувати описані підходи:

- подієво-орієнтована архітектура використовується для обробки повідомлень у режимі реального часу;
- планування завдань забезпечує регулярне виконання фонових задач;
- інтеграція API дозволяє розширити функціональність бота, а багатокрокова логіка спрощує складні сценарії взаємодії.

У цьому проєкті всі ці підходи були застосовані для створення багатофункціонального Telegram-бота, який автоматично керує підписками, нагадує користувачам про закінчення доступу та інтегрується з базою даних.

1.4. Вебзастосунки для керування данними користувачів

Telegram-бот, що виконує функції автоматизації керування підписками, значно спрощує взаємодію з кінцевими користувачами, проте його функціонал обмежений з точки зору адміністрування системи. У міру зростання кількості підписників, інформаційних ресурсів та варіантів підписок виникає потреба у гнучкому інструменті, який дозволяє швидко отримувати доступ до актуальної інформації про користувачів, змінювати їхній статус, переглядати аналітику, а також здійснювати ручне втручання у разі виникнення проблем. Бот як інтерфейс не є зручним для виконання складних адміністративних дій: відсутність візуального уявлення про дані, неможливість швидкого фільтрування та сортування інформації, обмеження у взаємодії з великим обсягом користувачів тощо. Саме тому виникає потреба у створенні окремого вебзастосунку для адміністратора системи, який дозволив би забезпечити повноцінне керування користувачами та підписками через зручний графічний інтерфейс, із підтримкою авторизації, фільтрації, пошуку, а також керування тарифами та ресурсами.

Таким чином, створення окремого вебзастосунку дозволить забезпечити централізоване, зручне і масштабоване керування користувачами системи, покращити контроль над сервісом і розширити можливості його розвитку без обмежень, притаманних Telegram-ботам.

1.5. Порівняння розробки ботів у різних месенджерах

Розробка ботів для різних месенджерів базується на унікальних технічних характеристиках кожної платформи, що визначають вибір архітектурних рішень, функціональність, масштабованість та безпеку майбутнього застосунку. У випадку з WhatsApp, розробка здійснюється через офіційний WhatsApp Business API, який

реалізовано як REST-сервіс і вимагає реєстрації компанії, проходження процедури верифікації та отримання дозволу на відправлення повідомлень. Цей API має жорсткі обмеження щодо типів повідомлень, які можна надсилати, а також вимагає попереднього погодження шаблонів. Це ускладнює як створення гнучких діалогових сценаріїв, так і масштабування рішення – через наявність квот і регламентів на швидкість відправлення. Архітектура ботів для WhatsApp, як правило, базується на сервері, що приймає події від месенджера через webhook і обробляє відповіді відповідно до логіки бізнес-процесу.

Facebook Messenger реалізований через Graph API, який дозволяє надсилати та приймати повідомлення з різноманітним вмістом, включаючи текст, зображення, кнопки, картки та шаблони. Для підключення бота обов’язково потрібно створити Facebook-сторінку, отримати access token і налаштувати webhook. У цьому випадку взаємодія з API вимагає створення HTTPS-сервера для обробки POST-запитів, що містять події користувачів. Facebook накладає додаткові обмеження на частоту спілкування з користувачами та суворо модерує контент. Це створює додатковий технічний і бюрократичний бар’єр для розробника. З іншого боку, екосистема Facebook дає доступ до додаткових даних про користувачів, що може бути корисно в контексті персоналізованих сервісів.

Viber пропонує дещо простіший API, який також використовує webhook для надсилання подій на сторонній сервер. Розробник має змогу надсилати повідомлення різних типів, зокрема текстові, медіа та інтерфейсні елементи (наприклад, кнопки), однак платформа має менше документації, меншу активність розробницької спільноти і порівняно вужчу аудиторію. Масштабованість рішення обмежується як технічними факторами, так і популярністю месенджера у конкретних регіонах.

Slack, у свою чергу, позиціонує себе переважно як інструмент для корпоративного використання. Його API підтримує розширені інтеграції з

внутрішніми сервісами, включаючи OAuth 2.0, Events API для отримання подій і Web API для керування ботом. Розробка тут орієнтована на автоматизацію командних робочих процесів, але не є зручною для створення ботів загального призначення, через що масштабування обмежене корпоративною структурою та внутрішніми політиками безпеки компаній.

Telegram у цьому порівнянні виділяється як найвідкритіша і найгнучкіша платформа для створення ботів. Telegram Bot API реалізовано як простий у використанні REST API, який дозволяє інтегрувати бота у систему з використанням як webhook, так і polling. Розробник сам обирає архітектурну модель — webhook ідеально підходить для продакшн-розгортання, polling — для тестування або роботи без домену з SSL. Telegram дозволяє надсилати повідомлення з текстом, мультимедіа, геолокацією, контактами, кнопками, inline клавіатурами і навіть формувати складні діалогові сценарії за допомогою callback-запитів. Ботам доступна інформація про користувача, включно з ID, іменем та мовою інтерфейсу, що дає змогу налаштувати персоналізовану логіку. Telegram підтримує концепцію inline-ботів, які користувач може викликати в будь-якому чаті для вставки динамічного контенту. Це відкриває додаткові можливості для інтеграції із зовнішніми сервісами та розробки нових типів взаємодії.

З погляду безпеки Telegram використовує HTTPS для комунікації між ботом і своїм API. Хоча end-to-end шифрування не застосовується до ботів, платформа залишається однією з найнадійніших завдяки застосуванню сучасних криптографічних алгоритмів і розділенню архітектурних рівнів взаємодії. Масштабування Telegram-ботів легко досягається через асинхронну обробку повідомлень, кешування даних сесій за допомогою Redis або подібних технологій, а також побудову черг повідомлень на базі RabbitMQ чи Kafka. Telegram не накладає обмежень на кількість користувачів або частоту повідомлень, що дає

змогу обслуговувати тисячі сесій паралельно навіть без горизонтального масштабування.

З точки зору DevOps, Telegram-боти зручно розгортати як у класичному серверному середовищі, так і в контейнерах Docker або на хмарних платформах (AWS, GCP, Heroku, Railway). Telegram Bot API добре інтегрується в CI/CD-пайплайни, що дозволяє автоматизувати тестування та оновлення. Існує велика кількість бібліотек і SDK для різних мов програмування, зокрема Python (python-telegram-bot), Node.js (telegraf), PHP (php-telegram-bot), Java (TelegramBots) та інші. Це значно прискорює час розробки, а активна спільнота й наявність великої кількості open source рішень забезпечують підтримку та швидке вирішення проблем.

Таким чином, Telegram поєднує відкритість, функціональну гнучкість, простоту у використанні, високу продуктивність і широкий вибір технічних засобів. Саме ці фактори роблять його оптимальною платформою для реалізації ботів як у комерційних, так і в навчальних чи дослідницьких проектах. У рамках дипломного проекту вибір Telegram як цільової платформи для створення бота є технічно виправданим і найраціональнішим у контексті вимог до гнучкості, масштабу, стабільності та швидкості розробки.

1.5. Огляд аналогічного програмного забезпечення

Перед початком розробки власної системи управління підписками та користувачами через Telegram-бот і вебзастосунок було проведено огляд існуючих рішень на ринку, які мають подібний функціонал. Метою аналізу є визначення сильних і слабких сторін конкурентних продуктів, а також пошук рішень, які можуть бути адаптовані або покращені в рамках власного проєкту.

Існує багато аналогічного програмного забезпечення.

InviteMember – популярний сервіс для монетизації Telegram-каналів через платну підписку. Основний функціонал включає налаштування тарифних планів, інтеграцію з платіжними шлюзами (Stripe, PayPal), автоматичне додавання/видалення підписників.

Переваги:

- зручна інтеграція з Telegram;
- готовий інтерфейс для адміністрування;
- підтримка кількох платіжних систем.

Недоліки:

- обмежена кастомізація;
- платна основа з високою комісією;
- немає доступу до коду та гнучкої інтеграції з власною базою.

TelescopeBot дозволяє створювати платні групи та канали з інтеграцією підписок. Пропонує мінімалістичний функціонал для адміністраторів.

Переваги:

- простота налаштування;
- мінімальні знання потрібні для запуску.

Недоліки:

- обмежена функціональність;
- немає API або гнучкої аналітики;
- відсутність глибокої кастомізації.

BotHelp – сервіс для створення маркетингових воронок у Telegram та керування контактами. Має розширений функціонал, проте не орієнтований на підписки.

Переваги:

- зручний редактор сценаріїв;
- підтримка кількох каналів комунікації.

Недоліки:

- висока вартість;
- непридатність для платного доступу до контенту.

РОЗДІЛ 2.
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ КЕРУВАННЯ ДОСТУПОМ ДО ІНФОРМАЦІЙНИХ РЕСУРСІВ ТА УПРАВЛІННЯ КОРИСТУВАЧАМИ

2.1. Постановка задачі та визначення вимог для програмного забезпечення

У рамках даного проєкту розробляється програмне забезпечення, яке складається з двох основних компонентів:

1. Telegram-бот – інструмент автоматизованої взаємодії з користувачами через месенджер Telegram;

2. вебдодаток (адміністративна панель) – інтерфейс для адміністраторів, що забезпечує контроль за користувачами, оплатами та доступами.

Загальні функціональні вимоги:

- забезпечення взаємодії з користувачем через Telegram (бот);
- інтеграція з Telegram API для додавання/видалення користувачів з каналів/груп;
- підтримка різних тарифних планів з відповідними періодами доступу;
- зберігання інформації про користувачів, підписки, платежі;
- надсилання автоматичних сповіщень за 3 дні до закінчення підписки;
- видалення користувачів з ресурсу після завершення оплаченого періоду;
- панель адміністратора з можливістю перегляду користувачів, історії оплат, стану підписок.

Вимоги до Telegram-бота:

- реєстрація нового користувача за допомогою Telegram ID;
- надання інтерфейсу вибору тарифного плану;
- здійснення перевірки оплати;

- видача запрошення в канал/групу після підтвердження оплати;
- ведення обліку дати початку та завершення підписки;
- автоматичне надсилання нагадувань перед завершенням підписки;
- видалення користувача після завершення строку, якщо нова оплата не підтверджена.

Вимоги до вебдодатку:

- список користувачів з фільтрами та пошуком;
- перегляд історії оплат та активності користувача;
- можливість вручну додати або продовжити підписку користувачу;
- інформаційна панель з поточною статистикою (кількість користувачів, активні підписки, найближчі завершення);
- масштабованість (можливість додати підтримку нових тарифів або функцій без повного переписування коду)

2.2. Аналіз та вибір моделі розробки програмного засобу

У процесі створення програмного забезпечення важливим етапом є вибір відповідної моделі розробки, яка визначає порядок, послідовність і підходи до виконання усіх стадій життєвого циклу програмного продукту. Вибір моделі повинен базуватись на складності проєкту, його цільовому призначенні, гнучкості вимог, рівні взаємодії із замовником, а також доступних ресурсах і термінах реалізації.

У контексті розробки Telegram-бота для керування підписками та вебзастосунку для адміністрування користувачів, було проаналізовано кілька поширених моделей розробки програмного забезпечення.

Каскадна модель (Waterfall). Ця модель передбачає послідовне проходження усіх етапів: від аналізу вимог до впровадження. Її недоліком є слабка адаптивність

до змін на пізніх етапах розробки, що робить її непридатною для динамічних проєктів.

Спіральна модель. Ця модель орієнтована на проєкти з високим рівнем ризиків і включає етапи прототипування та багаторазового уточнення. Незважаючи на свою ефективність, вона вимагає великих витрат часу й ресурсів, що не є доцільним для невеликого проєкту з обмеженим бюджетом.

Ітеративна та інкрементальна модель. Цей підхід дозволяє розробляти систему частинами, поступово додаючи нові функціональні можливості. Він забезпечує раннє тестування й гнучкість у зміні вимог, але потребує добре продуманої архітектури з самого початку.

Agile-підхід (Scrum / Kanban). Agile-модель передбачає гнучке управління розробкою, розбиття роботи на короткі ітерації (спринти), постійний зворотний зв’язок і можливість швидкої адаптації до змін. Цей підхід добре підходить для команд невеликого та середнього розміру, а також для проєктів, у яких вимоги можуть змінюватися в процесі.

З огляду на обсяг і специфіку проєкту, а саме:

- наявність двох взаємопов’язаних програмних продуктів (бот і вебзастосунок);
- потребу у гнучкому додаванні функціоналу на основі відгуків користувачів;
- змінність бізнес-логіки (тарифні плани, періоди доступу, методи оплати);
- обмежений термін реалізації (3–4 місяці).

Було прийнято рішення використовувати Agile-підхід з елементами Scrum.

Розробка системи проводилась поетапно: спочатку реалізовувався базовий функціонал Telegram-бота, після чого створювався вебінтерфейс для адміністрування. На кожному етапі проводилось тестування та отримувався зворотний зв’язок, що дозволяло швидко виявляти й усувати недоліки.

Такий підхід забезпечив:

- швидкий запуск MVP (мінімально життєздатного продукту);

- поступове розширення функціоналу без переписування основи;
- ефективне планування задач через Kanban-дошку;
- мінімізацію ризиків завдяки ітеративному тестуванню.

Таким чином, Agile-підхід є найбільш доцільною моделлю розробки для реалізації подібних систем із високим рівнем інтерактивності, динамічними вимогами та необхідністю підтримки користувачів в режимі реального часу.

2.3. Опис проєкту

Архітектура системи побудована за принципом клієнт-серверної моделі з двома точками взаємодії: Telegram-ботом для користувачів та вебінтерфейсом для адміністратора (рис 2.1). Користувач звертається до Telegram-бота, який передає запити на сервер, а той, у свою чергу, взаємодіє з базою даних MongoDB. Адміністратор працює через React-інтерфейс, який також надсилає запити на сервер. Уся логіка обробки запитів, авторизації та доступу до даних зосереджена на стороні Express-сервера.[23]

2.4. Обґрунтування вибору інструментальних засобів розробки

Для реалізації вебзастосунку було обрано сучасний технологічний стек MERN (MongoDB, Express.js, React, Node.js), що дозволяє створити повноцінний клієнт-серверний застосунок з єдиною мовою програмування – JavaScript, як на фронтенді, так і на бекенді.[24]

Node.js було обрано як серверне середовище виконання JavaScript завдяки його високій продуктивності, підтримці асинхронної обробки запитів та широкій екосистемі модулів, серед яких важливе місце займає Express.js – легкий фреймворк, що спрощує побудову REST API та обробку маршрутів.

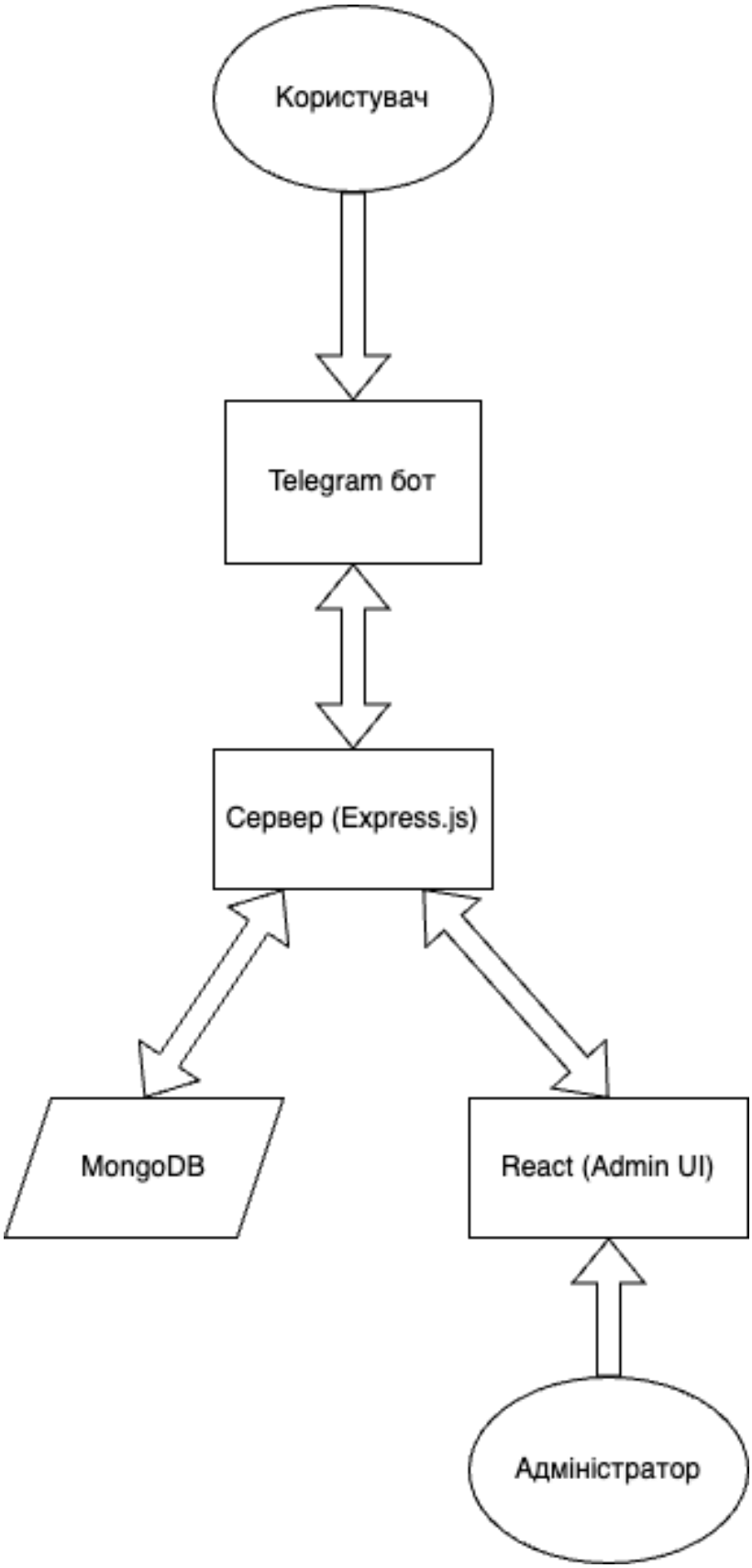


Рисунок 2.1 – Схема архітектури проєкту

Для побудови інтерфейсу адміністратора було використано React – бібліотеку для створення динамічних застосунків. React забезпечує компонентну структуру, що полегшує розробку, повторне використання коду та масштабування проєкту. У парі з ним використовується Tailwind CSS, що дозволяє швидко реалізовувати адаптивний інтерфейс завдяки утилітарному підходу до стилізації.[13]

Базу даних реалізовано на основі MongoDB – документно-орієнтованої NoSQL СУБД, яка забезпечує гнучкість структури даних, простоту масштабування та зручну інтеграцію з Node.js через бібліотеку Mongoose.[5]

У процесі розробки Telegram-бота було прийнято рішення використовувати бібліотеку Telegraf.js, оскільки вона є однією з найпопулярніших та найзручніших для створення ботів на Node.js. Ця бібліотека надає зрозумілий синтаксис, підтримує middleware-архітектуру, сцени, контекстну обробку повідомлень і легко інтегрується з асинхронними функціями. Telegraf дозволяє швидко реалізовувати логіку взаємодії з користувачами, обробляти команди, надсилати повідомлення та виконувати дії залежно від статусу підписки.

Окрім Telegraf.js, існують й інші бібліотеки, які могли бути використані для створення Telegram-бота. Наприклад, node-telegram-bot-api – це ще один потужний інструмент для Node.js, який має широку документацію та підтримує більшість можливостей Bot API, проте він менш гнучкий у побудові складних сценаріїв взаємодії, ніж Telegraf. Для розробки ботів на Python однією з найпоширеніших бібліотек є python-telegram-bot, яка забезпечує повноцінну підтримку Telegram API й добре підходить для синхронних задач. Більш сучасним рішенням для Python є aiogram – асинхронна бібліотека з підтримкою asyncio, що дозволяє будувати масштабовані системи, але потребує глибшого знання Python та роботи з асинхронністю.

З урахуванням того, що основна частина проєкту реалізована на Node.js, вибір Telegraf.js був оптимальним рішенням. Він забезпечив єдине середовище розробки,

зручну інтеграцію з Express.js і MongoDB, а також дозволив підтримувати логіку всього проєкту на одній мові програмування – JavaScript.[22]

У цьому проєкті Telegram-бота використовуються декілька ключових бібліотек, які суттєво полегшують розробку та додають корисну функціональність. Ось огляд найцікавіших і найважливіших із них:

Telegraf – це основна бібліотека для створення Telegram-бота на Node.js. Вона забезпечує зручний інтерфейс для обробки повідомлень, команд, callback-запитів, inline-кнопок тощо. Telegraf підтримує middleware-підхід, що дозволяє чітко структурувати логіку взаємодії з користувачем. Також вона добре інтегрується з асинхронною логікою, сценами та локалізацією.

telegraf-i18n – бібліотека додає підтримку інтернаціоналізації (i18n) для Telegraf-бота. З її допомогою можна легко реалізувати мультимовний інтерфейс, наприклад, англійську та українську мови, використовуючи прості YAML/JSON-файли з перекладами. Це особливо корисно, якщо бот орієнтований на широку аудиторію.

telegraf-ratelimit – забезпечує захист від спаму та надмірної кількості запитів. Цей модуль обмежує частоту виконання команд користувачами, щоб зменшити навантаження на сервер і не допустити зловживань (наприклад, надсилання десятків команд поспіль).

Mongoose – одна з найпопулярніших бібліотек для взаємодії з MongoDB. Вона дозволяє описувати структуру документів у вигляді схем (schemas), створювати моделі (models) та виконувати CRUD-операції зручним способом. У контексті бота вона використовується для збереження користувачів, термінів підписки, статусів, прав доступу тощо. [16]

telegraf-session-local – проста сесійна бібліотека, яка дозволяє зберігати тимчасовий стан користувача локально (у файлі або в пам’яті). Наприклад, якщо

бот працює зі сценами, можна зберігати проміжні відповіді користувача, вибрані опції або мову.

axios та node-fetch – бібліотеки використовуються для виконання HTTP-запитів до сторонніх API. Наприклад, axios часто використовується для перевірки транзакцій у мережі USDT, а node-fetch — як легковаговий інструмент для отримання JSON-даних. [10][18]

У процесі розробки програмного забезпечення було обрано середовище розробки Visual Studio Code (VS Code) як основний інструмент для написання коду та організації роботи над проєктом.

VS Code є легким, швидким та потужним редактором коду, розробленим компанією Microsoft. Він має відкритий вихідний код, підтримує велику кількість мов програмування та технологій, зокрема ті, що використовуються в проєкті: JavaScript, Node.js, React, MongoDB, а також YAML, JSON та Markdown для конфігураційних файлів і документації. [17]

Наведемо основні переваги VS Code, які вплинули на вибір.

Широкий вибір розширень, зокрема:

- ESLint – для автоматичної перевірки стилю коду [19];
- Prettier – для форматування[20];
- REST Client – для тестування API без потреби в сторонніх інструментах.

Інтеграція з Git – дозволяє легко керувати версіями коду, зокрема через GitHub або GitLab, що є критично важливим у командній розробці або при впровадженні CI/CD. [14]

Термінал всередині редактора – забезпечує зручність запуску локального сервера, MongoDB, тестів або команд npm без перемикання між вікнами.

Використання VS Code як основного інструменту дозволило ефективно організувати розробку, пришвидшити процес написання коду, спростити

відлагодження та забезпечити зручність у тестуванні всіх компонентів програмного продукту – як Telegram-бота, так і вебзастосунку на MERN-стеку.

Обрані інструменти забезпечують швидку розробку, зручну підтримку, гнучку архітектуру та відповідають вимогам сучасного повнофункціонального вебзастосунку.

2.5. Особливості програмної реалізації Телеграм бота

Програмна реалізація Telegram-бота базується на Node.js із використанням фреймворку Telegraf.js, що забезпечує модульну архітектуру з поділом на сервіси, сцени та динамічну обробку станів користувача. Бот інтегрується з криптовалютною мережею USDT TRC20 для прийому оплат, використовуючи валідацію хешів транзакцій та збереження даних у базі. Ключові функції включають управління підписками, автоматичне додавання користувачів до чатів і нагадування про завершення підписок. Для безпеки реалізовано перевірку вводу, захист токенів і чутливих даних, а обробка повідомлень здійснюється через чергу для уникнення перевантаження API Telegram.

2.5.1. Реєстрація бота в телеграм

Для початку роботи нам потрібно отримати токен нашого бота, який ми маємо використати для ідентифікації його у Telegram. Для цього нам потрібно виконати наступні кроки (рис.2.2):

- відкрити діалог із BotFather у Telegram[25];
- виконати команду /newbot для створення нового бота;
- казати назву та унікальний юзернейм для бота.

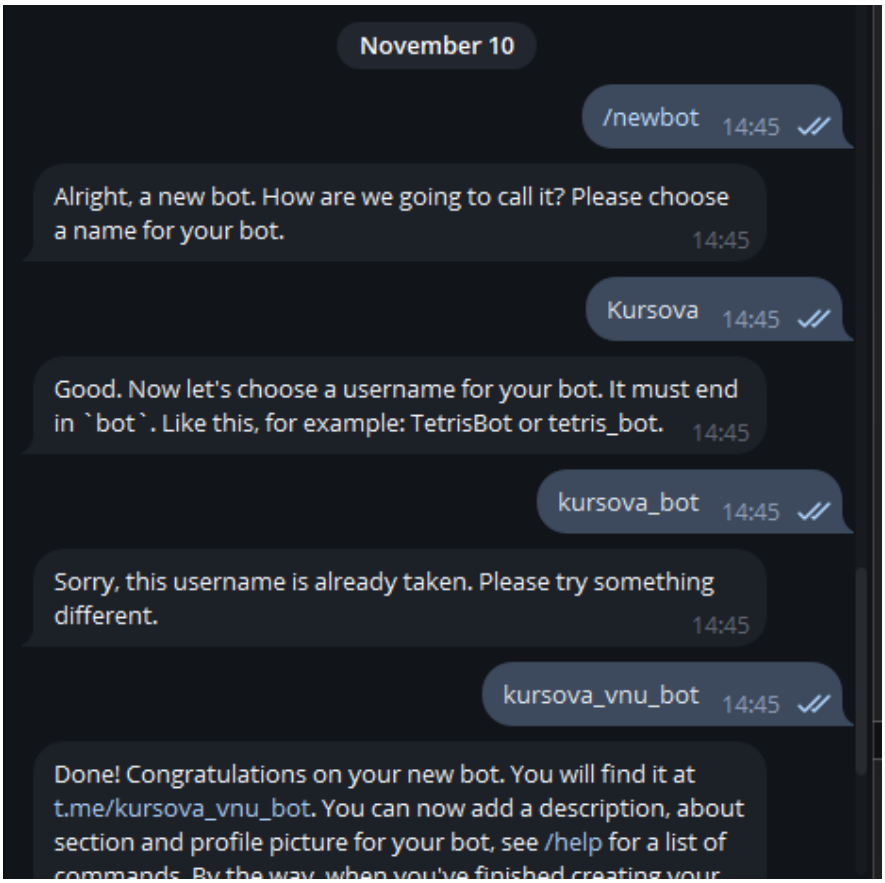


Рисунок 2.2 – Приклад чату з ботом BotFather

2.5.2. Конфігураційний файл

На рис. 2.3. подано стартовий конфігураційний файл. Цей конфігураційний файл використовується для налаштування бота. Він включає адресу гаманця для прийому платежів у мережі TRC20 (USDT), який використовується для перевірки транзакцій. Також задається кількість днів, за які бот нагадає користувачам про завершення їхньої підписки.

У файлі передбачені параметри для збереження інформації про груповий чат і Telegram-канал. Це включає ідентифікатор, назву і посилання-запрошення до чату або каналу. Крім цього, передбачено масив для опису тарифних планів, що міститиме деталі про вартість, тривалість і назву підписок. Зараз цей масив порожній, оскільки він є лише прикладом і буде заповнений при створенні тарифів.[9]

```json
{
  "USDT_TRC20_WALLET_ADDRESS": "TGXfRhjz5tz6WXRRAEcwRuR7anDpnbELWr",
  "REMINDER_DAYS": 3,
  "CHAT": {
    "id": "",
    "title": "",
    "invite_link": ""
  },
  "CHANNEL": {
    "id": "",
    "title": "",
    "invite_link": ""
  },
  "TARIFFS": [

  ]
}
```

Рисунок 2.3 – Стартовий конфігураційний файл

2.5.3. Сцени

Сцени в Telegram-ботах використовуються для створення багатокрокових взаємодій із користувачами. Вони дозволяють структурувати логіку бота так, щоб кожен крок взаємодії залежав від попередніх даних. Найчастіше сцени реалізуються за допомогою бібліотеки Telegraf.js і її модуля telegraf/scenes.

Сцена – це контекст взаємодії, в якому бот очікує конкретні повідомлення або дії від користувача. Користувач перебуває у сцені, доки не виконає всі кроки або не завершить сцену вручну.

Сцени працюють наступним чином:

- користувач ініціює дію, яка активує сцену (наприклад, натискає кнопку або надсилає команду);
- бот переводить користувача в контекст сцени.
- сцена може складатися з декількох послідовних кроків. На кожному кроці бот чекає вводу певних даних від користувача;

- після завершення всіх кроків сцена завершується, і користувач виходить із її контексту.

У нашому боті реалізовані три глобальні сцени (рис.2.4):

- adminka – сцена для адміністратора;
- pay – сцена для оплати;
- users – сцена для ручного керування користувачами.

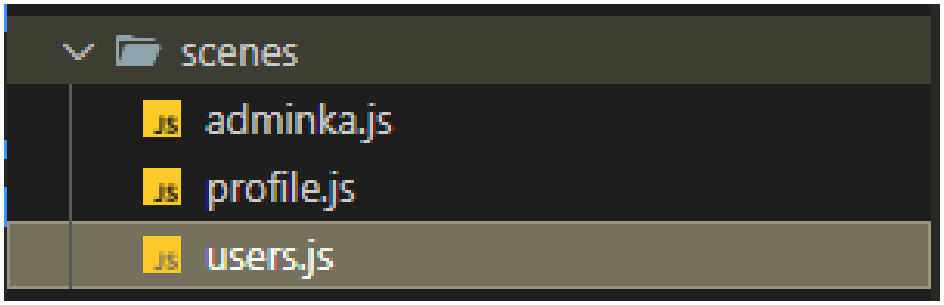


Рисунок 2.4 – Сцени

2.5.4. Сцена адміністративної панелі

Адміністративна панель (рис. 2.5.) дозволяє адміністратору:

- змінити адресу гаманця для оплати;
- змінити посилання на телеграм ресурси;
- редагувати/додавати користувачів, тарифи.

2.5.5. Зміна даних інформаційного ресурсу

Функція editChatLink створює сцену для Telegram-бота, яка дозволяє редагувати параметри чату (ID, назву та посилання-запрошення) у конфігурації. Вона реалізує покрокову взаємодію з користувачем, де кожен крок відповідає за введення одного з параметрів.

Сцена _chatlink створюється з використанням модуля Scene (рис.2.6). Вона буде відповідати за всю взаємодію, пов’язану з редагуванням даних чату.

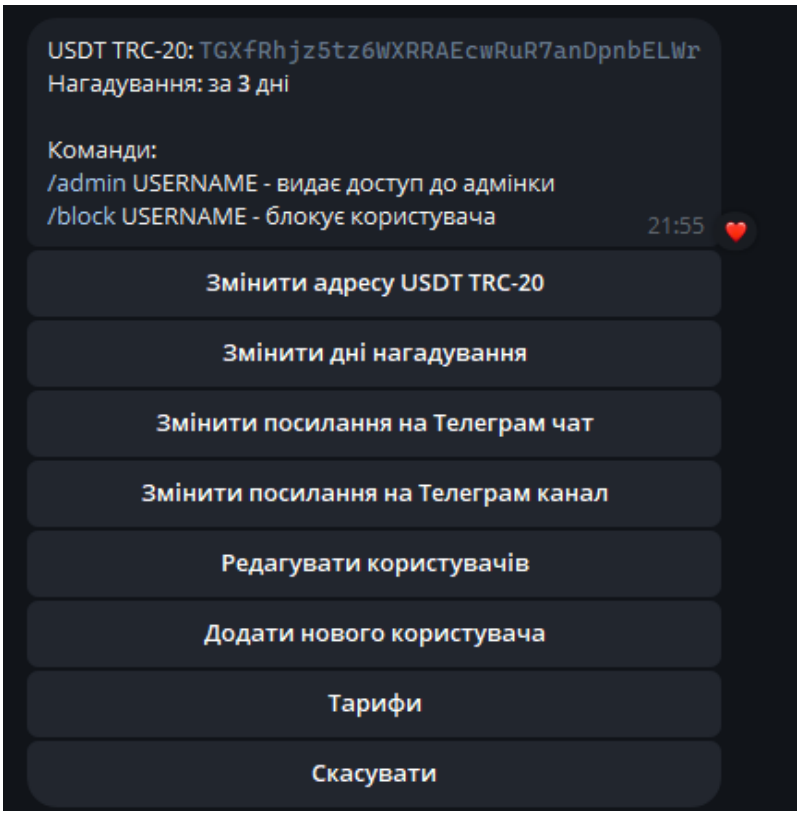


Рисунок 2.5 – Вигляд адміністративної панелі

```
1  const _chatlink = new Scene('_chatlink');
```

Рисунок 2.6 – Створення сцени

Блок, поданий на рис 2.7 виконується, коли користувач входить у сцену _chatlink. Спочатку ініціалізуються змінні step (номер кроку) і data (поточна конфігурація чату), після чого завантажуються дані конфігурації чату з сервісу balanceService['CONFIG']['CHAT']. Потім користувачеві надсилається повідомлення з першим кроком (введення ID чату) через функцію messages.editChatLink.

Блок коду, що подано на рис.2.8, обробляє текстові повідомлення, які користувач надсилає у відповідь на запит бота. Поточний крок і дані зберігаються у ctx.scene.state.

На кожному кроці перевіряється, який параметр потрібно відредагувати. Спочатку зберігається id чату, потім title (назва чату), а після цього – invite_link (посилання-запрошення). Якщо дані введено успішно, сцена переходить до наступного кроку, а користувач отримує нове повідомлення із запитом на введення наступного параметра. На останньому кроці оновлюється конфігурація за допомогою balanceService.updateConfig('CHAT', data), після чого сцена завершується викликом ctx.scene.leave().

```
_chatlink.enter(async (ctx) => {
        const { user } = ctx.state;
        const { message_id } = ctx.update.callback_query.message;

        ctx.scene.state.step = 0;

        const data =  balanceService['CONFIG']['CHAT'];
        ctx.scene.state.data = balanceService['CONFIG']['CHAT']; // Load data from config
        console.log('Scene entered. Data:', ctx.scene.state.data);
        const message = messages.editChatLink(user.lang, ctx.scene.state.step, ctx.scene.state.data);

        sender.enqueue({
            chat_id: ctx.from.id,
            message
        });
    });
```

Рисунок 2.7 – Вхід у сцену

2.5.6. Функція tarrifs

Функція tariffs створює сцену для управління тарифами в Telegram-боті (рис.2.9). Вона дозволяє адміністратору додавати, редагувати, переглядати, видаляти тарифи та управляти їх атрибутами через покрокову взаємодію з ботом.

```
_chatlink.on('text', async (ctx) => {
        const { user } = ctx.state;
        const { step, data } = ctx.scene.state; // Get current step and data
        const { text } = ctx.message;

        console.log('Current step:', step);
        console.log('Data before processing:', data);
        console.log('User input:', text);



        if (!data) {
            console.error('Data is undefined. Cannot proceed.');
            return;
        }

        let isCorrect = false,
                message = null;

        if (step === 0) {
            isCorrect = true;
            ctx.scene.state.step++;
            data.id = text;
            message = messages.editChatLink(user.lang, ctx.scene.state.step, ctx.scene.state.data);
        } else if (step === 1) {
            isCorrect = true;
            ctx.scene.state.step++;
            data.title = text;
            message = messages.editChatLink(user.lang, ctx.scene.state.step, ctx.scene.state.data);
        } else if (step === 2) {
            isCorrect = true;
            ctx.scene.state.step++;
            data.invite_link = text;

            message = messages.editChatLink(user.lang, ctx.scene.state.step, ctx.scene.state.data);
        } else if (step === 3) {
            balanceService.updateConfig('CHAT', data);
            message = messages.editChatLink(user.lang, ctx.scene.state.step, ctx.scene.state.data);
            await ctx.scene.leave();
        }


        _chatlink.action('confirm', async (ctx) => {


            balanceService.updateConfig('CHAT', ctx.scene.state.data);

            await ctx.deleteMessage();
            await ctx.scene.leave();
        });

        if (message) {
            sender.enqueue({
                chat_id: ctx.from.id,
                message
            });
        }
        console.log('Data after processing:', data);
    });
```

Рисунок 2.8 – Код обробника

```
_tariffs.action('add', async (ctx) => {
        const { user } = ctx.state;
        const { message_id } = ctx.update.callback_query.message;

        ctx.scene.state.step++;
        ctx.scene.state.data = {
            chats: []
        };

        const message = messages.addTariff(user.lang, ctx.scene.state.step, ctx.scene.state.data, message_id);

        sender.enqueue({
            chat_id: ctx.from.id,
            message
        });
    });
```

Рисунок 2.9 - Код методу add

```
_tariffs.action('confirm', async (ctx) => {
        const { CHAT, CHANNEL, TARIFFS } = balanceService.CONFIG;

        const id = (TARIFFS[TARIFFS.length - 1]) ? Number(TARIFFS[TARIFFS.length - 1].id) : 0;
        ctx.scene.state.data.id = (id) ? id + 1 : 1;

        ctx.scene.state.data.chats = [
            {
                id: CHAT.id,
                title: CHAT.title,
                invite_link: CHAT.invite_link
            },
            {
                id: CHANNEL.id,
                title: CHANNEL.title,
                invite_link: CHANNEL.invite_link
            }
        ];

        balanceService.updateConfig('TARIFFS', ctx.scene.state.data);

        await ctx.deleteMessage();
        await ctx.scene.reenter();
    });
```

Рисунок 2.10 – Код методу confirm

Додавання нового тарифу відбувається таким чином: збільшується значення step і створюється новий об’єкт тарифу (data), після чого користувачу надсилається повідомлення для введення назви тарифу через функцію messages.addTariff.

Підтвердження створення тарифу відбувається у кілька кроків (рис.2.10). Спочатку новому тарифу присвоюється ID, після чого автоматично додаються дані чату та каналу тарифу. Потім тариф додається до конфігурації через balanceService.updateConfig(‘TARIFFS’, data). Після цього користувач повертається до списку тарифів. Вибір тарифу для редагування відбувається за ID тарифу, отриманим після натискання кнопки. За цим ID з конфігурації вибирається відповідний тариф, після чого користувачу надсилається повідомлення для його редагування (рис 2.11).

```
_tariffs.action(/tariff-([0-9]+)/, async (ctx) => {
        const { user } = ctx.state;
        const { message_id } = ctx.update.callback_query.message;

        const id = ctx.match[1];
        const tariff = balanceService['CONFIG']['TARIFFS'].find(el => el.id == id);

        ctx.scene.state.data = tariff;

        const message = messages.editTariff(user.lang, tariff, message_id);

        sender.enqueue({
            chat_id: ctx.from.id,
            message
        });
    });
```

Рисунок 2.11 – Код методу tariff

Для редагування атрибутів тарифу, атрибут (наприклад,ціна, назва чи тривалість підписки) обирається через ключ у кнопці. Далі сцена переходить на відповідний крок для редагування цього атрибуту (рис. 2.12).

```
_tariffs.action(/edit-([a-z]+)/, async (ctx) => {
        const { user } = ctx.state;
        const { data } = ctx.scene.state;
        const { message_id } = ctx.update.callback_query.message;

        const key = ctx.match[1];

        ctx.scene.state.step = STEPS[key];

        const message = messages.addTariff(user.lang, ctx.scene.state.step, data, message_id);

        sender.enqueue({
            chat_id: ctx.from.id,
            message
        });
    });
```

Рисунок 2.12 – Код методу edit

2.5.7. Сцена оплати

Сцена pay обробляє оплату підписки за допомогою криптовалюти USDT у мережі TRC20. Вона взаємодіє з користувачем, отримує від нього дані (вибір тарифу, хеш транзакції) і зберігає їх для подальшої перевірки (рис.2.13).

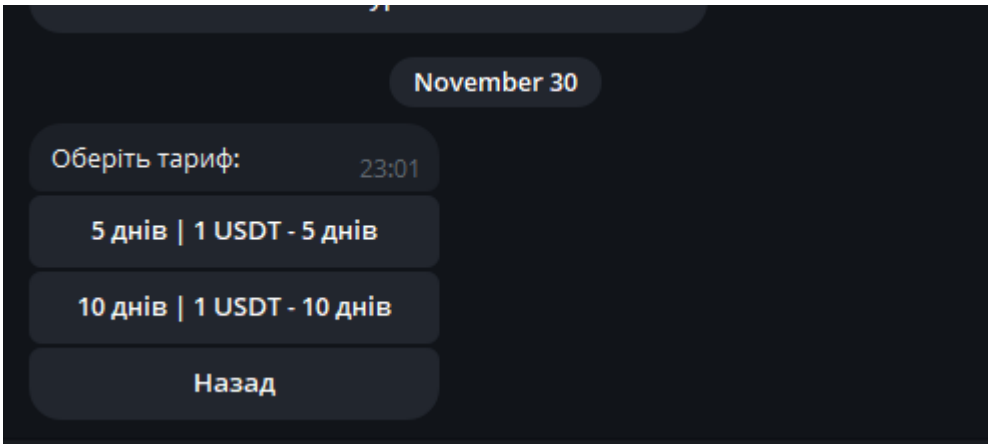


Рисунок 2.13 – Сцена оплати

Після вибору тарифу, користувачу буде запропоновано відправити певну суму на гаманець вказаний у повідомленні (рис.2.14).

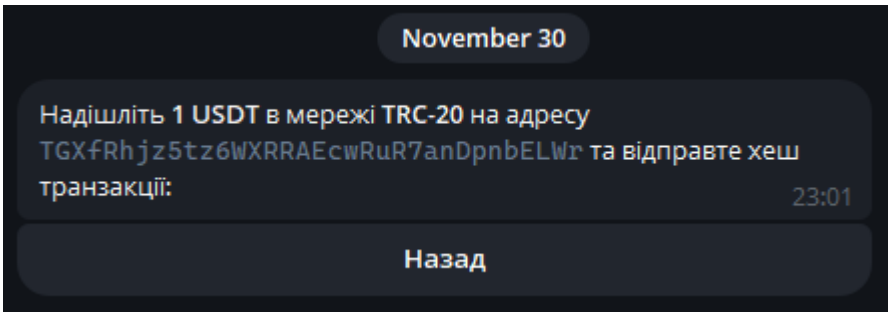


Рисунок 2.14 – Повідомлення від бота про адресу гаманця і суму, необхідну для отримання послуги.

Якщо користувач відправляє текст, що відповідає хешу транзакції (64-символьний рядок у форматі hex), перевіряється унікальність хешу в базі даних через paymentDBService.get.

Якщо хеш є новим, створюється запис про оплату в базі даних (paymentDBService.create), оплата додається до черги на перевірку (balanceService.enqueue), користувач отримує повідомлення з подякою, після чого бот виходить зі сцени. Повідомлення про помилку буде відправлено у випадку, якщо хеш є невалідним, сума в транзакції недостатня або отримувач вказаний неправильно (рис.2.15).

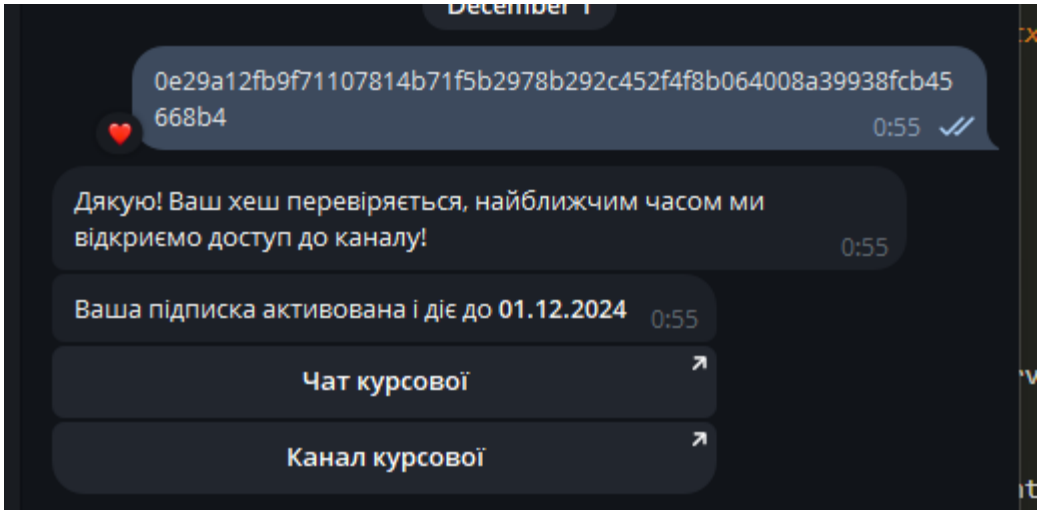


Рисунок 2.15 – Приклад успішного платежу

Якщо користувач не вніс вчасно платіж, він буде сповіщений та видалений з ресурсів. Також користувачу буде заблоковано доступ до наступної оплати. При оплаті відбувається розблокування користувача в інформаційних ресурсах.

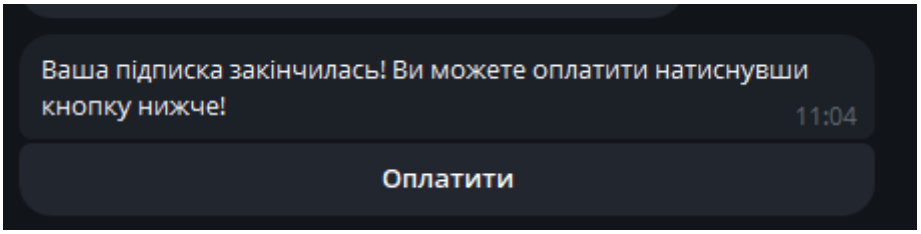


Рисунок 2.16 – Приклад сповіщення про анулювання підписки

```
pay.hears(USDT_TRC20_HASH_REG, async (ctx) => {
        const { user } = ctx.state;
        const {
            step,
            data
        } = ctx.scene.state;
        const { text } = ctx.message;

        if (step === 2) {
            const check = await paymentDBService.get({ hash: text });

            if (!check) {
                const payment = await paymentDBService.create({
                    tg_id: ctx.from.id,
                    amount: data.amount,
                    currency: 'USDT',
                    wallet: data.wallet,
                    hash: text,
                    data
                });

                balanceService.enqueue(payment);

                await ctx.replyWithHTML(ctx.i18n.t('thxForPayment_message'));
                await ctx.scene.leave();
            } else {
                const message = messages.paymentError(user.lang, { key: 'hashIsAlreadyUse_message' });

                sender.enqueue({
                    chat_id: ctx.from.id,
                    message
                });
            }
        }
    });
```

Рисунок 2.17 – Код обробник платежу

2.5.8. Сцена для редагування користувачів

Адміністратор може вручну редагувати дату закінчення підписки, а також давати іншим користувачам права адміністратора у боті, блокувати, розблоковувати користувачів. Також адміністратор може додавати нового користувача до бази даних, без взаємодії попереднього з самим ботом, але що важливо, бот не може першим ініціювати діалог, що запобігає спаму.

Адміністратор може вручну відредагувати дату закінчення підписки, наприклад, у випадку, коли користувач провів оплату в інший спосіб. Якщо вказана дата підписки є майбутньою відносно сьогодні, то відбувається процес подібний до того, що відбувається при оплаті, користувач розблоковується у інформаційних джерелах та має доступ за посиланням до них (рис. 2.18).

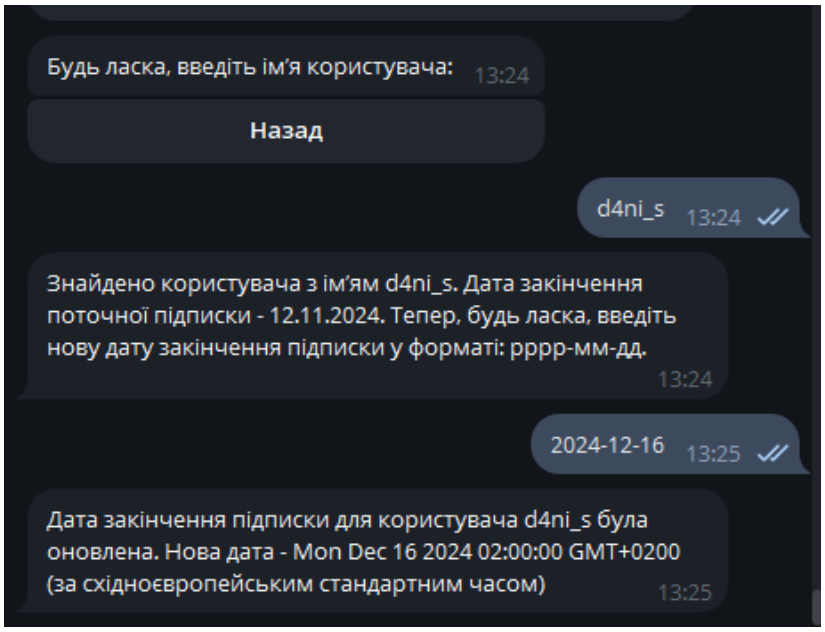


Рисунок 2.18 – Приклад редагування дати закінчення підписки

Адміністратор може додати нового користувача у разі потреби. Наприклад, якщо до цього адміністрування велося вручну, через таблицю в Excel, а зараз потрібно внести нових користувачів до бази даних. Для цього адміністратору

потрібно знати tg_username та tg_id. Після того, як адміністратор ввів дані йому потрібно буде ввести дату закінчення підписки.

2.5.9. Моделі MongoDB

У MongoDB моделі це концепція, яка використовується в бібліотеках, таких як Mongoose, щоб полегшити роботу з даними в MongoDB. Модель визначає структуру документів, які зберігаються в колекції, а також надає методи для взаємодії з базою даних (записи, пошук, оновлення, видалення тощо). В нашому проекті представлено дві моделі: User та Payment.

Об'єкти в моделях є умовними записами, колекціями даних про певну сутність, наприклад, користувача.

User модель відповідає за створення об'єкту користувача та містить дані про нього, необхідні для роботи з ним (рис.2.19):

- tg_id – унікальний ідентифікатор Telegram-користувача, обов'язковий для заповнення;
- tg_username – ім'я користувача у Telegram. Необов'язкове поле, може бути відсутнє;
- lang – мова користувача, обмежена переліченими значеннями (uk, en, ru);
- isActive – вказує, чи є користувач активним. За замовчуванням – true;
- isAdmin – позначає, чи має користувач права адміністратора;
- isBlocked – чи заблокований користувач;
- from – джерело, звідки прийшов користувач;
- chats – масив, який зберігає ID або назви чатів, пов'язаних із користувачем;
- start_date – дата, коли користувач розпочав роботу з ботом або системою;
- sub_end_date – дата завершення підписки (може бути null, якщо підписки немає).

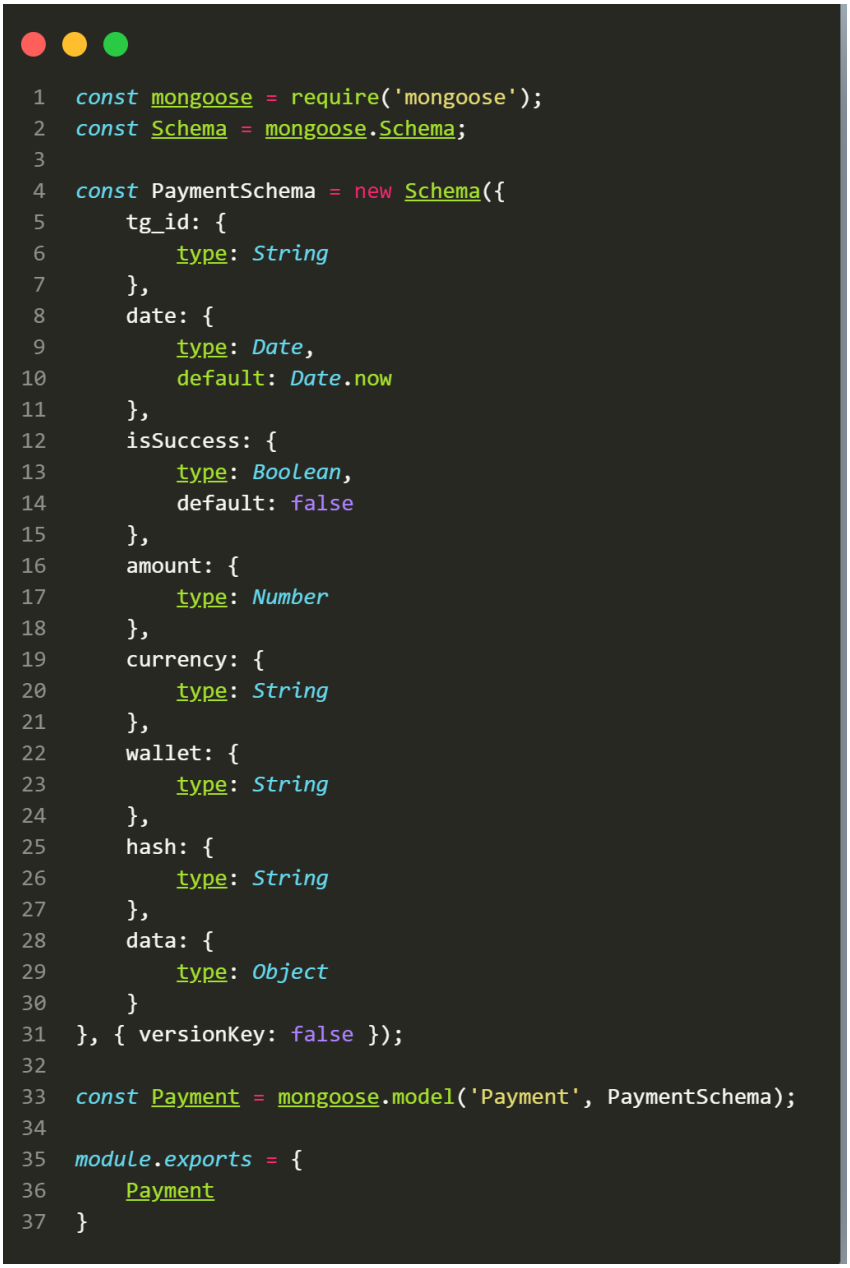
```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const UserSchema = new Schema({
    tg_id: {
        type: String,
        required: true,
        unique: true,
        index: true,
    },
    tg_username: {
        type: String,
        required: true
    },
    lang: {
        type: String,
        required: true,
    },
    isActive: {
        type: Boolean,
        default: true
    },
    isAdmin: {
        type: Boolean,
        default: false
    },
    isBlocked: {
        type: Boolean,
        default: false
    },
    start_date: {
        type: Date,
        default: Date.now
    },
    sub_end_date: {
        type: Date,
        default: Date.now
    },
    from: {
        type: String,
        default: 'organic'
    },
    chats: {
        type: Array,
        default: []
    }
}, { versionKey: false });

const User = mongoose.model('User', UserSchema);

module.exports = {
    User
}
```

Рисунок 2.19 – Створення моделі користувача у MongoDB



```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const PaymentSchema = new Schema({
    tg_id: {
        type: String
    },
    date: {
        type: Date,
        default: Date.now
    },
    isSuccess: {
        type: Boolean,
        default: false
    },
    amount: {
        type: Number
    },
    currency: {
        type: String
    },
    wallet: {
        type: String
    },
    hash: {
        type: String
    },
    data: {
        type: Object
    }
}, { versionKey: false });

const Payment = mongoose.model('Payment', PaymentSchema);

module.exports = {
    Payment
}
```

Рисунок 2.20 – Створення моделі Payment

Payment модель (рис.2.20) відповідає за створення об’єкту транзакції та містить дані які, необхідні для роботи:

- tg_id – Telegram ID користувача, який здійснив платіж;
- isSuccess – позначає, чи був платіж успішним;
- amount – сума, яку сплатив користувач;
- currency – валюта, в якій був здійснений платіж (наприклад, USDT);

- wallet – гаманець, на який був здійснений платіж;
- hash – унікальний хеш транзакції для ідентифікації платежу;
- data – додаткові дані про тариф за який проходила оплата;
- date – дата обробки платежу.

2.6. Програмна реалізація вебдодатку на базі MERN

Програмна реалізація вебзастосунку базується на технологічному стеку MERN (MongoDB**,** Express.js, React, Node.js)**.** Застосунок має модульну архітектуру, що поділяє функціональність на окремі компоненти: маршрути, контролери, сервіси та інтерфейсні модулі. Серверна частина, реалізована на Node.js із використанням Express.js, відповідає за REST API, автентифікацію, обробку запитів до бази даних і взаємодію з Telegram-ботом через HTTP-запити або Webhook. [1]

Клієнтська частина реалізована за допомогою React з використанням Tailwind CSS для стилізації, що забезпечує адаптивний, швидкий і зручний інтерфейс. Компонентна структура React дозволяє легко масштабувати систему та оновлювати окремі функціональні блоки без впливу на загальну архітектуру.

2.6.1. Архітектура MERN-додатку

Клієнтська частина (Frontend). Клієнтська частина додатку реалізована з використанням бібліотеки React, яка дозволяє створювати динамічний, реактивний інтерфейс на основі компонентної структури. Основні елементи:

- компоненти – окремі блоки інтерфейсу (таблиці, форми, панелі), що повторно використовуються;
- React Router – маршрутизація між сторінками без перезавантаження[21];
- Fetch API – для надсилання запитів до серверного API;
- Tailwind CSS – для стилізації інтерфейсу з використанням утилітарних класів.

Серверна частина (Backend) – Node.js + Express.js. Серверна логіка реалізована на платформі Node.js з фреймворком Express.js, який спрощує створення REST API. Основні компоненти серверної архітектури:

- роутери (routes) – визначають кінцеві точки API для взаємодії з клієнтом;
- контролери (controllers) – обробляють запити, викликають сервіси та повертають відповіді;
- сервіси (services) – містять бізнес-логіку (керування підписками, перевірка прав доступу). [2]

Вся комунікація між React-інтерфейсом і Node-сервером здійснюється через REST API, тобто сервер повертає дані у форматі JSON через HTTP запити, які клієнт обробляє та відображає.[12]

2.6.2. Організація серверної частини

Серверна частина відповідає за встановлення з'єднання з базою даних і обробку всіх запитів, що надходять з клієнтської сторони, зокрема – автентифікацію адміністратора, створення, редагування та видалення користувачів.

Наприклад роутер record.js обробляє маршрути, пов'язані з користувачами (рис. 2.21). Код реалізує набір REST-операцій для колекції users у MongoDB: отримання всіх користувачів, отримання одного користувача за ID, створення нового користувача, оновлення даних, видалення, а також розширення підписки. Додатково є маршрут для отримання статистики. Запити обробляються асинхронно, з використанням MongoDB через драйвер і ObjectId для роботи з ідентифікаторами.

Завдяки цьому роутеру сервер може приймати запити типу GET, POST, PATCH, DELETE, відповідно обробляти їх, взаємодіяти з базою даних, а також повертати клієнту відповідні відповіді або повідомлення про помилки.

Також додатково було створено модель бази даних Admin, щоб зберігати дані аутентифікації адміністратора. Ця модель реалізована за допомогою бібліотеки

Mongoose, яка дозволяє створювати схеми для MongoDB у вигляді об’єктно-орієнтованих структур.

```
router.get("/:id", async (req, res) => {
  try {
    const collection = await db.collection("users");
    const query = { _id: new ObjectId(req.params.id) };
    const result = await collection.findOne(query);

    if (!result) {
      res.status(404).json({ message: "Not found" });
      return;
    }

    res.json(result);
  } catch (error) {
    console.error("Error fetching record:", error);
    res.status(400).json({ message: error.message });
  }
});
```

Рисунок 2.21 – Приклад обробки маршруту для отримання даних користувача з певним ідентифікатором

Модель Admin містить два основні поля – username та password, де обидва є обов’язковими, а ім’я користувача має бути унікальним (рис.2.22). Для підвищення безпеки паролі не зберігаються у відкритому вигляді. Завдяки функції pre(‘save’) перед збереженням документа у базу паролі автоматично хешуються з використанням алгоритму bcrypt. Це означає, що навіть у разі витоку бази даних зловмисник не отримає реальні паролі, а лише їхні захищені хеші.

Окрім цього, в моделі реалізований метод comparePassword, який дозволяє порівняти введений користувачем пароль із тим, що збережений у базі. Це використовується під час входу адміністратора для перевірки правильності облікових даних.

```
import mongoose from 'mongoose';
import bcrypt from 'bcrypt';

const adminSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  password: { type: String, required: true }
});

adminSchema.pre('save', async function(next) {
  if (!this.isModified('password')) return next();
  try {
    const salt = await bcrypt.genSalt(10);
    this.password = await bcrypt.hash(this.password, salt);
    next();
  } catch (err) {
    next(err);
  }
});

adminSchema.methods.comparePassword = async function(candidatePassword) {
  return bcrypt.compare(candidatePassword, this.password);
};

export default mongoose.model('Admin', adminSchema);
```

Рисунок 2.22 – Реалізація моделі Admin

Таким чином, дана модель забезпечує надійну систему зберігання й перевірки паролів, що є критично важливою складовою безпечної аутентифікації адміністратора в адміністративному вебзастосунку.

Після успішного входу користувача (наприклад, адміністратора), сервер створює JWT і відправляє його клієнту. Клієнт зберігає цей токен (найчастіше в localStorage або sessionStorage) і додає його до заголовка Authorization кожного наступного запиту.

JWT (JSON Web Token) – це відкритий стандарт (RFC 7519), який дозволяє безпечно передавати інформацію між клієнтом і сервером у вигляді зашифрованого

токена. Він часто використовується для аутентифікації користувачів у вебзастосунках, зокрема в тих, що реалізовані на MERN-стеку (рис. 2.23). [8]

Рисунок 2.23 – Приклад JWT токена та його збереження в Local storage браузеру

Серверна частина має чітку модульну структуру, де кожен функціональний блок винесено в окрему директорію або файл. Наприклад:

- routes/ – містить усі Express-маршрути;
- models/ – Mongoose-схеми для роботи з MongoDB;
- middleware/ – проміжні функції, зокрема перевірка JWT;
- db/ – підключення бази даних;
- server.js - основний файл запуску застосунку, де створюється екземпляр Express-сервера, підключається файл імпорту бази даних, реєструються маршрути, мідлвари та запускається сервер на визначеному порту.

2.6.3. Структура та реалізація інтерфейсу користувача

Клієнтська частина застосунку реалізована за допомогою бібліотеки React, що дозволяє створювати інтерактивні інтерфейси з компонентною архітектурою. Основне завдання цієї частини – забезпечити адміністратора зручним вебінтерфейсом для керування користувачами, перегляду їх даних, пошуку, та редагування даних (рис.2.24).

Інтерфейс розділений на окремі функціональні компоненти. Наприклад, компонент RecordList відповідає за виведення таблиці з користувачами, підтримує пагінацію, пошук за Telegram ID або ім’ям користувача. Для стилізації

використовується утилітарний CSS-фреймворк Tailwind CSS, що дозволяє швидко створювати адаптивний і сучасний інтерфейс без перевантаження стилями.

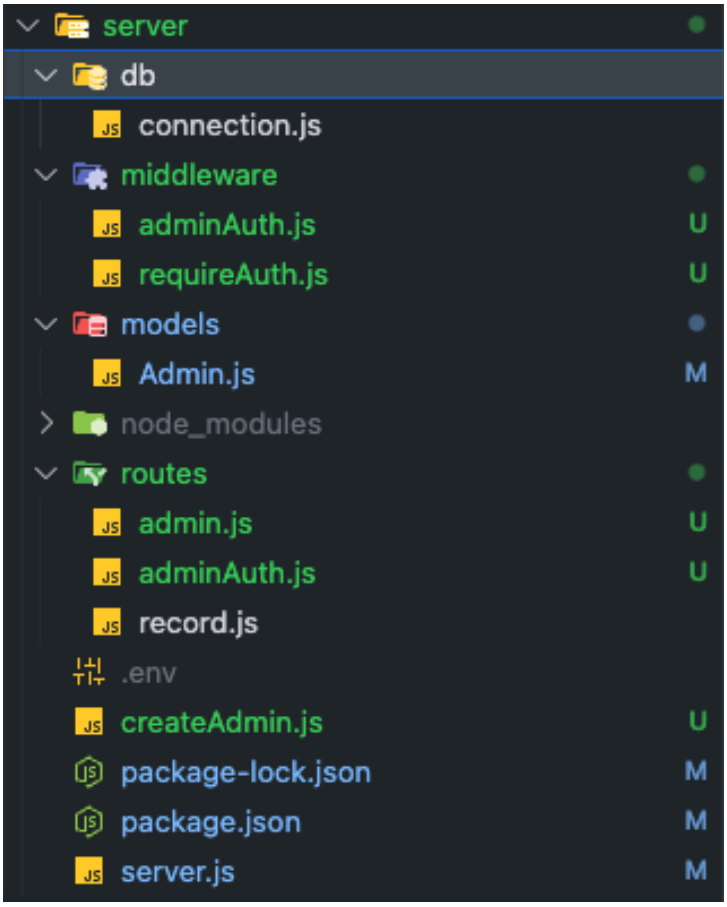


Рисунок 2.24 – Структура серверної частини

З клієнта надсилаються запити до серверної частини за допомогою fetch API. При отриманні списку користувачів, видаленні чи оновленні записів React-інтерфейс динамічно оновлюється без перезавантаження сторінки, що забезпечує комфортне користування навіть при великій кількості даних.

Розпочнемо поетапне знайомство з інтерейсом, адміністратора зустрічає сторінка логінізації у якій потрібно вказати логін та пароль для входу (рис.2.25).

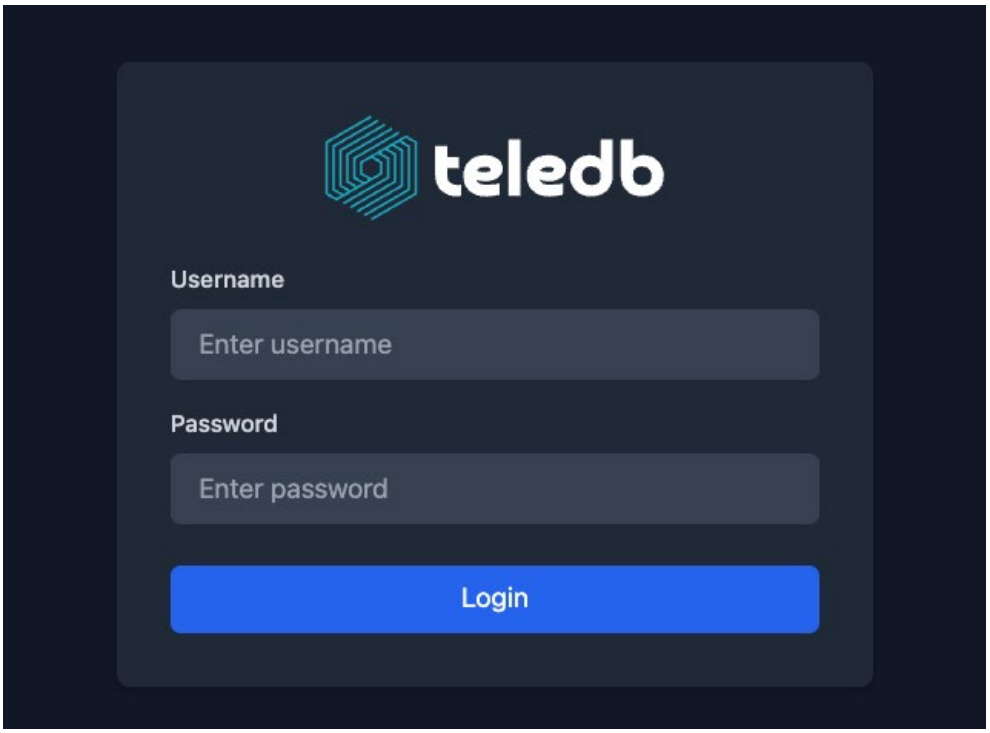


Рисунок 2.25 – Сторінка логіну

У вебзастосунку реалізована базова система автентифікації адміністратора, яка складається з двох основних механізмів: хешування пароля за допомогою bcrypt і використання JWT-токена для збереження сеансу.

Під час створення адміністратора його пароль не зберігається у відкритому вигляді – замість цього він хешується за допомогою алгоритму bcrypt, що забезпечує надійний захист навіть у разі витоку бази даних. Хеш зберігається в базі, а під час входу введений пароль порівнюється з хешем.[11]

Після успішної автентифікації сервер створює JWT (JSON Web Token), який містить зашифровану інформацію про користувача (наприклад, його ID). Цей токен зберігається на клієнті (зазвичай у localStorage) і додається до кожного запиту в заголовках, щоб підтвердити авторизований доступ до захищених маршрутів.

Після успішної аутентифікації користувач переходить на головну сторінку, яка містить таблицю зі списком усіх користувачів та основною інформацією про них (рис. 2.26). У таблиці передбачено можливість вибору кількості записів, що

відображаються на одній сторінці (ітерації пагінації), а також пошук конкретного користувача за Telegram ID або ім’ям користувача (username).

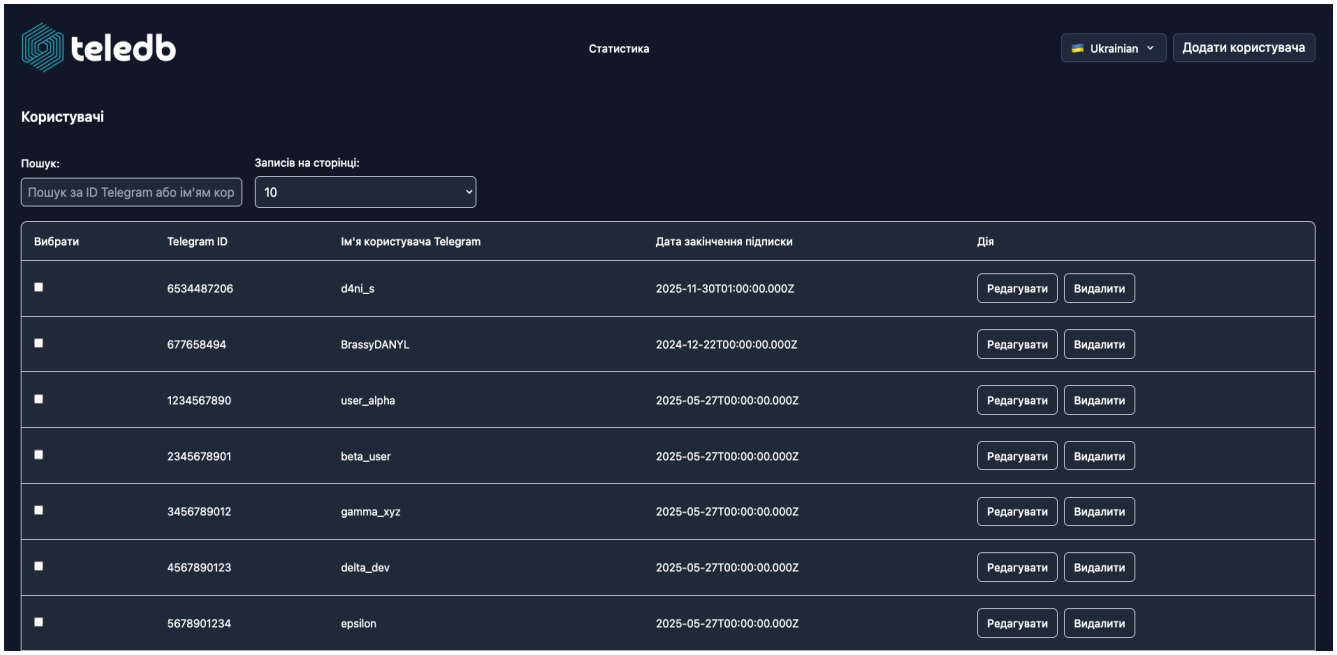


Рисунок 2.26 – Головна сторінка додатку з таблицею користувачів

В хедері користувачу доступна навігація, для переходу на інші сторінки, які можуть бути додані в подальшому, перемикач мови та кнопка яка дозволяє додати нового користувача.

В самій таблиці адміністратор може додати вибрати користувача для подальшого редагування, вибрати декількох користувачів для продовження періоду підписки на певний термін, видалити користувача, або відредагувати дані певного користувача.

При натисканні кнопки «Додати користувача» адміністратор побачить форму для створення користувача (рис.2.27).

Також створений компонент на якому відображається статистика доходів, використовуючи бібліотеку Recharts для візуалізації даних у вигляді стовпчастої діаграми. Він отримує дані про користувачів з API, обчислює потенційний

щомісячний дохід на основі дати закінчення підписки кожного користувача, і представляє ці дані у вигляді діаграми. Дані про доходи обчислюються на основі фіксованої ціни підписки та дати закінчення підписки кожного користувача, а потім перетворюються у формат, придатний для Recharts.

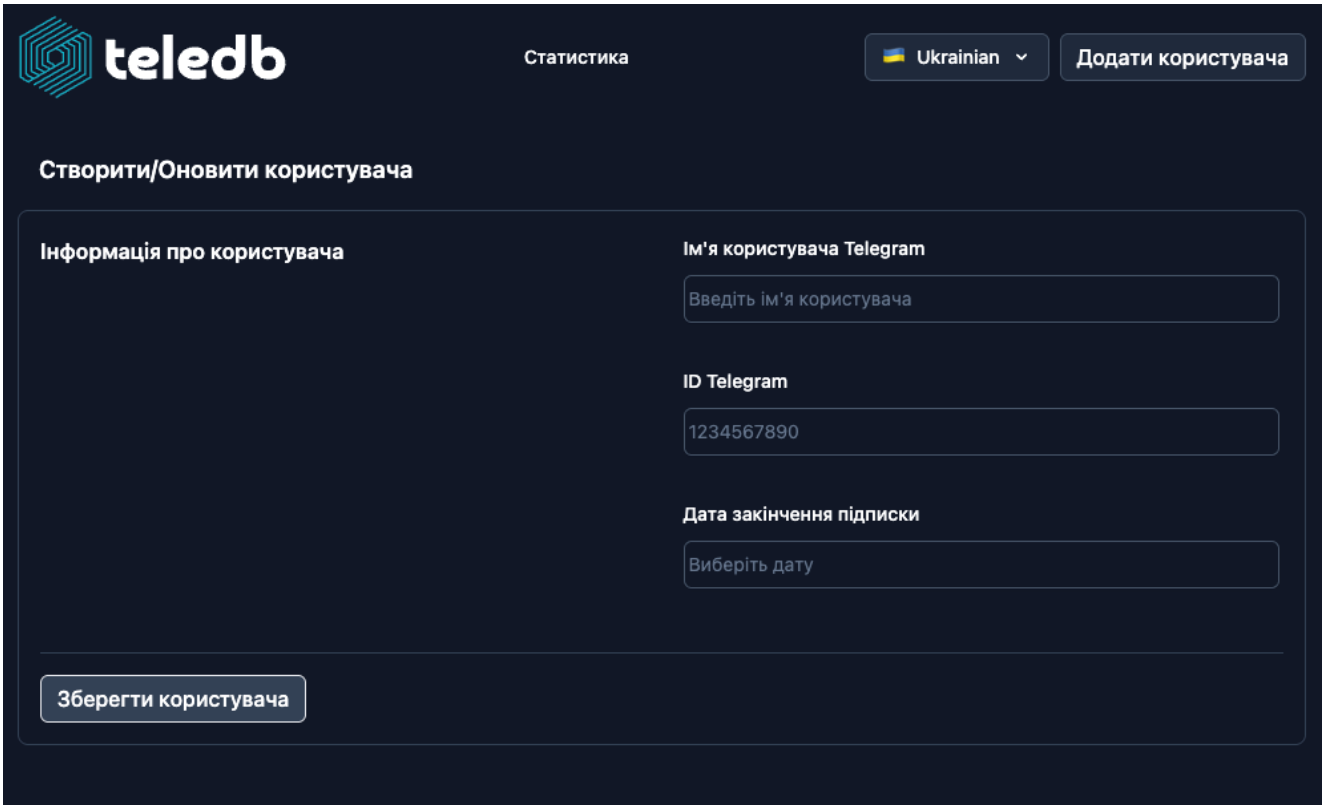


Рисунок 2.27 – Форма створення користувача

Вебдодаток використовує react-i18next для реалізації багатомовності, що дозволяє адаптувати інтерфейс до різних мов. За допомогою хука useTranslation компоненти отримують доступ до функції t, яка використовується для отримання перекладів для текстових елементів інтерфейсу, таких як мітки, заголовки та повідомлення (рис. 2.28). Це забезпечує локалізацію контенту для різномовних користувачів.

```json
{
  "users": "Користувачі",
  "search": "Пошук:",
  "recordsPerPage": "Записів на сторінці:",
  "extendSubscription": "Продовжити підписку",
  "edit": "Редагувати",
  "delete": "Видалити",
  "extendByDays": "Продовжити на дні:",
  "extend": "Продовжити",
  "createUser": "Створити/Оновити користувача",
  "userInfo": "Інформація про користувача",
  "tgUsername": "Ім'я користувача Telegram",
  "tgId": "ID Telegram",
  "subscriptionEndDate": "Дата закінчення підписки",
  "selectDate": "Виберіть дату",
  "saveUser": "Зберегти користувача",
  "statistics": "Статистика",
  "addUser": "Додати користувача",
  "searchByTelegramIdOrUsername": "Пошук за ID Telegram або ім'ям користувача",
  "select": "Вибрати",
  "telegramId": "Telegram ID",
  "telegramUsername": "Ім'я користувача Telegram",
  "subscriptionExpirationDate": "Дата закінчення підписки",
  "action": "Дія",
  "enterusername": "Введіть ім'я користувача",
  "monthlyPotentialRevenue": "Потенційний щомісячний дохід",
  "revenue": "Дохід"
}
```

Рисунок 2.28 – Приклад файлу з перекладами у форматі JSON

2.7. Організація тестування та налагодження програмного засобу

Тестування програмного забезпечення здійснювалося вручну на всіх етапах розробки, як клієнтської, так і серверної частини. У процесі створення Telegram-бота та вебзастосунку було проведено серію перевірок функціональності, стабільності та взаємодії з базою даних.

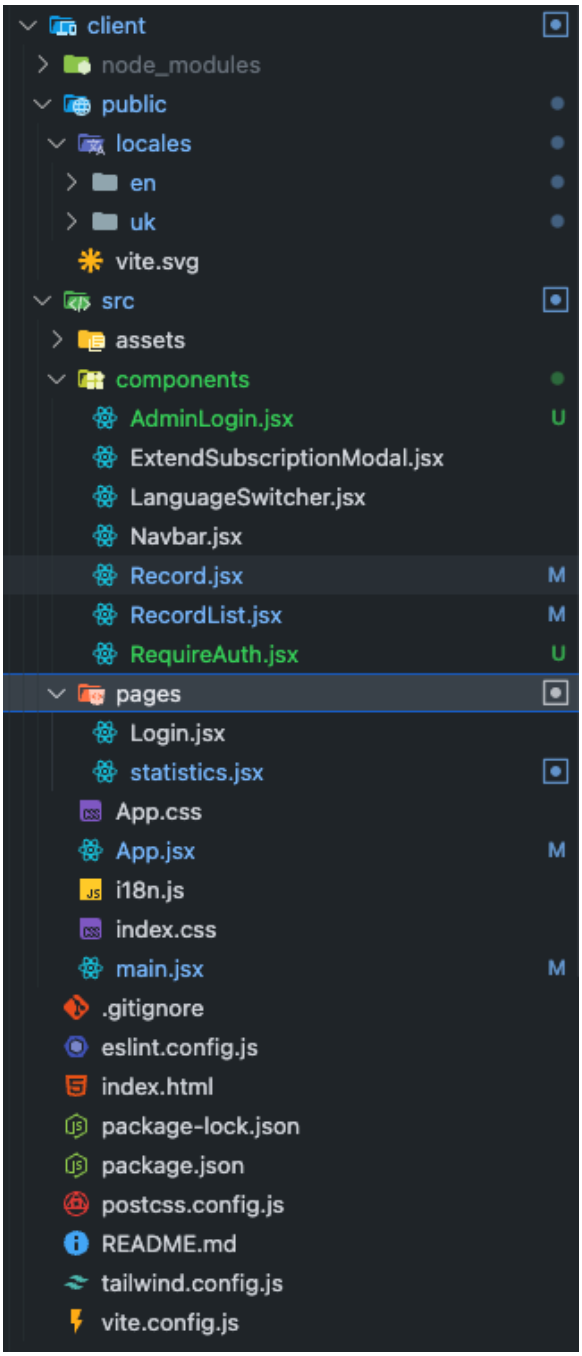


Рисунок 2.29 – Структура клієнтської частини вебдодатку

Для тестування серверної частини застосовувались інструменти на кшталт Postman і що дозволяло надсилати HTTP-запити до API, перевіряти правильність обробки даних, відповідність статус-кодів та вмісту відповіді.

Клієнтська частина тестувалася через браузер у реальному часі, з особливою увагою до:

- коректності відображення таблиці користувачів;
- взаємодії з модальним вікном продовження підписки;
- автоматичного оновлення інтерфейсу після дій над записами.

Особливу увагу було приділено перевірці авторизації через JWT. Тестувалися сценарії з відсутнім, недійсним або простроченим токеном. У всіх випадках застосунок правильно обробляв запити – повертав помилку доступу або перенаправляв користувача на сторінку входу.

Проєкт розміщено у віддаленому репозиторії GitHub, що дозволяло зручно вести контроль версій, створювати коміти при додаванні нової функціональності або виправленні помилок. У разі необхідності було легко відстежити зміни в коді та повернутися до стабільної версії.

Загалом програмний засіб продемонстрував стабільну роботу в умовах реального використання, забезпечуючи взаємодію між ботом, базою даних та інтерфейсом адміністратора без суттєвих збоїв.

2.8 Рекомендації по використанню та впровадженню програмного засобу

Програмний засіб, що складається з Telegram-бота та вебзастосунку для адміністрування користувачів і підписок, рекомендовано впроваджувати в організаціях, які надають платний доступ до інформаційних ресурсів через Telegram. Він буде особливо корисним для аналітичних каналів, освітніх спільнот, медіапроєктів, маркетингових агентств та інших сервісів, що працюють за моделлю передплати. Його використання дозволяє автоматизувати значну частину рутинної роботи: додавання та видалення підписників, перевірку платежів, сповіщення про

завершення доступу, а також зручне адміністрування всієї системи через інтуїтивно зрозумілий вебінтерфейс.

Для ефективного впровадження програмного комплексу рекомендується попередньо налаштувати Telegram-бота через сервіс @BotFather, отримати токен доступу та створити окремий Telegram-канал або групу з обмеженим доступом. Далі необхідно розгорнути серверну частину (Node.js + MongoDB) на хостингу або VPS, встановити всі залежності та налаштувати змінні середовища для безпечного зберігання ключів і даних. Вебзастосунок запускається в середовищі браузера та не потребує складного встановлення — достатньо мати стабільне з’єднання з API-сервером.

У процесі експлуатації адміністратор може керувати підписками через панель керування, фільтрувати користувачів, редагувати інформацію, переглядати дати завершення підписки та вручну продовжувати доступ. Для більшої зручності також передбачено мультимовний інтерфейс, який може бути доповнений іншими мовами відповідно до аудиторії.

Важливо забезпечити регулярне резервне копіювання бази даних і захищене зберігання токенів доступу. У разі інтеграції з платіжними системами слід додатково перевірити коректність обробки транзакцій і використання безпечних каналів зв’язку. Рекомендується також протестувати систему з невеликою групою користувачів перед масштабним запуском, щоб виявити можливі логічні або технічні помилки.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було успішно реалізовано автоматизовану систему для керування підписками на інформаційні ресурси в месенджері Telegram, що включає два ключові компоненти: Telegram-бот і адміністративний вебзастосунок на базі MERN-стеку.

У ході роботи було досягнуто наступних результатів:

- розроблено Telegram-бота з використанням Node.js і Telegraf.js, який автоматично додає користувачів до каналів, перевіряє факт оплати, надсилає нагадування та видаляє користувачів після завершення терміну підписки;
- реалізовано вебінтерфейс адміністратора на базі React і Tailwind CSS;
- застосовано технологію JWT для безпечної автентифікації адміністратора, а також хешування паролів з використанням алгоритму bcrypt;
- впроваджено REST API на базі Express.js для обробки запитів між клієнтом і сервером, з гнучкою взаємодією з базою даних MongoDB;
- проведено комплексне тестування функціоналу системи, як вручну, так і за участі сторонніх користувачів, що дозволило виявити й усунути помилки, пов’язані з авторизацією, пошуком, розширенням підписки та взаємодією з Telegram API;
- забезпечено стабільну роботу застосунку в режимі реального часу, а також реалізовано адаптивність до змін тарифів і структури даних.

Загалом, розроблений програмний засіб є прикладом ефективного поєднання сучасних вебтехнологій, серверної логіки, баз даних та інтеграції з зовнішнім API. Проєкт демонструє здатність вирішувати реальні завдання автоматизації, має практичну цінність для малого та середнього інформаційного бізнесу і може бути основою для подальшого розвитку або комерціалізації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Express - Node.js web application framework. Express - Node.js web application framework. URL: https://expressjs.com (date of access: 05.06.2025).
2. Index | node.js v24.1.0 documentation. Node.js – Run JavaScript Everywhere. URL: https://nodejs.org/en/docs (date of access: 05.06.2025).
3. Mongoose ODM v8.15.1. Mongoose ODM v8.15.1. URL: https://mongoosejs.com (date of access: 05.06.2025).
4. React. React. URL: https://react.dev (date of access: 05.06.2025).
5. Team M. D. MongoDB documentation - homepage. MongoDB: The World's Leading Modern Database | MongoDB. URL: https://www.mongodb.com/docs/ (date of access: 05.06.2025).
6. Telegraf.js - v4.16.3. telegraf.js - v4.16.3. URL: https://telegraf.js.org (date of access: 05.06.2025).
7. Telegram bot API. Telegram APIs. URL: https://core.telegram.org/bots/api (date of access: 05.06.2025).
8. JWT.IO Introduction to JSON Web Tokens. Auth0. URL: https://jwt.io/introduction (date of access: 05.06.2025).
9. dotenv package – npm. URL: https://www.npmjs.com/package/dotenv (date of access: 05.06.2025).
10. axios - Promise based HTTP client for the browser and node.js. URL: https://axios-http.com (date of access: 05.06.2025).
11. bcryptjs – npm. A bcrypt library for Node.js. URL: https://www.npmjs.com/package/bcryptjs (date of access: 05.06.2025).
12. REST API Design Guidelines – Microsoft. URL: https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design (date of access: 05.06.2025).
13. Tailwind CSS Documentation. URL: https://tailwindcss.com/docs (date of access: 05.06.2025).

14. GitHub Docs – Introduction to GitHub. URL: https://docs.github.com (date of access: 05.06.2025).
15. JSON – JavaScript Object Notation. URL: https://www.json.org (date of access: 05.06.2025).
16. Introduction to MongoDB CRUD Operations. URL: https://www.mongodb.com/docs/manual/crud/ (date of access: 05.06.2025).
17. Visual Studio Code – Code Editing Redefined. URL: https://code.visualstudio.com (date of access: 05.06.2025).
18. node-fetch – npm package. URL: https://www.npmjs.com/package/node-fetch (date of access: 05.06.2025).
19. ESLint – Pluggable JavaScript Linter. URL: https://eslint.org (date of access: 05.06.2025).
20. Prettier – Opinionated Code Formatter. URL: https://prettier.io (date of access: 05.06.2025).
21. React Router – Declarative Routing for React.js. URL: https://reactrouter.com (date of access: 05.06.2025).
22. JavaScript documentation – MDN Web Docs. Mozilla Developer Network. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (date of access: 05.06.2025).
23. Introduction to Express Middleware. URL: https://expressjs.com/en/guide/using-middleware.html (date of access: 05.06.2025).
24. MERN Stack Guide – MongoDB Developer Hub. URL: https://www.mongodb.com/mern-stack (date of access: 05.06.2025).
25. BotFather - Telegram Bot Management Tool. Telegram Official. URL: https://core.telegram.org/bots#botfather (date of access: 05.06.2025).

ДОДАТКИ

Додаток А

Технічне завдання

Призначення застосування – застосунок призначений для адміністраторів Telegram-каналів і власників інформаційних ресурсів, які реалізують доступ до контенту на умовах платної підписки. Система дозволяє автоматизувати перевірку оплати, контроль терміну дії підписки та керування доступом, а також надає зручний вебінтерфейс для управління користувачами, тарифами та підключеними каналами.

ПРИЗНАЧЕННЯ І ПІДСТАВИ ДЛЯ РОЗРОБКИ

Призначення цієї роботи полягає у створенні програмного забезпечення, що забезпечує автоматизоване управління підписками в Telegram, інтегроване з платіжними системами, а також інтерфейс адміністратора, що дає змогу повноцінно контролювати користувачів, перевіряти статус підписки та оперативно вносити зміни.

В умовах зростання попиту на інформаційні ресурси, що працюють через месенджери, виникає потреба в гнучких, надійних і захищених системах монетизації доступу. Telegram, як один із найпопулярніших месенджерів, дозволяє ефективно поширювати контент, але не має вбудованих механізмів для управління платним доступом. Це зумовлює необхідність розробки окремого інструменту, який поєднує в собі автоматизацію дій бота та зручність вебінтерфейсу для адміністрування.

Запропоноване рішення є актуальним для малого та середнього бізнесу, освітніх платформ, аналітичних спільнот, які використовують Telegram як канал комунікації та поширення контенту.

ВИМОГИ ДО ПРОГРАМИ

Вимоги до функціональних характеристик:

- система повинна забезпечувати автоматичне додавання користувача до приватного каналу/групи після підтвердження оплати;
- Telegram-бот повинен надсилати сповіщення про завершення підписки та видаляти користувача після її завершення;
- вебінтерфейс повинен дозволяти адміністратору переглядати список користувачів, шукати їх за Telegram ID або username, змінювати терміни підписки, видаляти чи додавати користувачів вручну;
- реалізована підтримка багатомовного інтерфейсу (через i18n);
- система повинна взаємодіяти з базою даних MongoDB через REST API.

Вимоги до надійності:

- система повинна стабільно працювати у режимі 24/7;
- Telegram-бот має коректно обробляти випадки недоступності API Telegram або помилки транзакцій;
- після технічного збою сервер має відновлювати функціональність протягом 5 хвилин;
- всі ключові дії (створення користувача, оновлення підписки тощо) мають логуватися.

Умови експлуатації:

- вебзастосунок повинен коректно працювати в сучасних браузерах (Chrome, Firefox, Edge);
- серверна частина має розгортатися на сервері з підтримкою Node.js і MongoDB;

- Telegram-бот має бути зареєстрований через @BotFather з активним токеном.

Вимоги до складу і параметрів технічних засобів:

- мінімальні вимоги до сервера: CPU 2 vCore, RAM 2 ГБ, SSD 10 ГБ;
- для розробки: ПК із Node.js (v18+), MongoDB, VS Code;
- клієнтська частина протестована на екранах від 360px до 1920px.

Додаток Б

Інструкція для користувача

Програмний комплекс складається з Telegram-бота та вебзастосунку для адміністрування користувачів і підписок. Щоб скористатися ботом, користувач повинен перейти за відповідним посиланням у Telegram та натиснути кнопку "Start". Після запуску бот запропонує оплатити підписку. Після успішного підтвердження транзакції користувач автоматично додається до приватного каналу або групи, на які поширюється підписка. У будь-який момент користувач може надіслати команду /status, щоб перевірити залишок часу до завершення дії підписки. Бот автоматично надсилає нагадування за кілька днів до завершення доступу. Після завершення оплаченої підписки бот самостійно видаляє користувача з каналу. Адміністратор також може виконувати адміністративні функції через інтерфейс бота, а саме: надавати права адміністратора бота іншим користувачам, змінювати термін підписки для певного користувача, змінювати дані для оплати (адресу криптовалютного гаманця), редагувати/додавати тарифи (ціна, назва, тривалість), змінити дані інформаційних ресурсів (назва та посилання для запрошення) та блокувати певних користувачів. Для кожної функції доступна своя кнопка в адміністративній панелі, яка доступна за командою /adminka.

Адміністратор працює через вебінтерфейс, який доступний за певною адресою. Перед початком використання потрібно ввести login і пароль. Після авторизації в системі відкривається таблиця з усіма користувачами: відображаються Telegram ID, імена користувачів, а також дати завершення підписки. Через панель можна шукати користувачів за ID або username, змінювати термін їх підписки, видаляти або додавати вручну. Є можливість масового продовження підписки для обраних користувачів — достатньо відмітити їх чекбоксами та натиснути відповідну кнопку. Також, на сторінці «Статистика»

адміністратор може переглянути прогнозований дохід від підписників. Прогноз здійснюється на основі кількості користувачів, у яких закінчується підписка протягом певного місяця. Для розрахунку потенційного доходу ця кількість множиться на базову вартість місячної підписки. Водночас слід враховувати, що ці підрахунки є орієнтовними, оскільки частина користувачів може не продовжити підписку, натомість можуть з’явитися нові підписники. Крім того, окремі користувачі можуть обрати довгострокові тарифи, наприклад, річну підписку, яка має нижчу вартість у перерахунку на місяць, що також впливає на точність прогнозу. Всі дані зберігаються в MongoDB, а обробка запитів здійснюється через REST API. Адміністративна частина захищена авторизацією з використанням JWT-токенів, а паролі зберігаються в базі в зашифрованому вигляді за допомогою bcrypt. Система автоматично відновлює роботу після збоїв, а всі помилки логуються на сервері. Для роботи з ботом потрібен акаунт Telegram і доступ до Інтернету, для роботи з вебінтерфейсом — сучасний браузер.

АНОТАЦІЯ

Санько Д.А. Розробка системи керування підписками в Telegram. Рукопис

Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 122 Комп'ютерні науки. Волинський національний університет імені Лесі Українки, Луцьк, 2025р.

У роботі представлено розробку комплексної системи управління підпискою на інформаційні ресурси, що функціонують у середовищі месенджера Telegram. Запропоноване рішення спрямоване на автоматизацію процесів обробки підписок, моніторингу платежів та адміністрування користувачів. Архітектура системи побудована на двох взаємопов'язаних програмних компонентах. Перший компонент — Telegram-бот, створений за допомогою Node.js і MongoDB, який відповідає за взаємодію з кінцевими користувачами, обробку транзакцій, контроль доступу до каналів, надсилання сповіщень та перевірку терміну дії підписки. Другий компонент — вебзастосунок, розроблений на основі сучасного MERN-стеку (MongoDB, Express.js, React, Node.js), що надає адміністративний інтерфейс для керування користувачами, перегляду та зміни підписок, фільтрації даних і аналітики. Система підтримує базові сценарії керування платним доступом, а також забезпечує безпечну обробку даних користувачів і можливість масштабування при збільшенні кількості підписників. Крім того, реалізовано багатомовний інтерфейс, адаптивну верстку та використання токенів JWT для автентифікації адміністратора.

Ключові слова: Telegram, бот, підписка, автоматизація, JavaScript, Node.js, MongoDB, MERN, React, Express.