МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: «**РОЗРОБКА ВЕБСЕРВІСУ ДЛЯ МОНІТОРИНГУ ПОПУЛЯРНОСТІ ВІДЕОКОНТЕНТУ**»

Виконав: студент 4 курсу, групи 6.1211-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
М.С. Крюков
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, PhD Столярова А.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Крюкову Марку Сергійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебсервісу для моніторингу популярності відеоконтенту

керівник роботи Столярова Анастасія Валеріївна, PhD
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Дослідження та аналіз предметної області.
2. Проєктування вебдодатку.
3. Реалізація вебдодатку.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 26.12.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	13.01.2025	
2.	Збір вихідних даних.	05.02.2025	
3.	Обробка методичних та теоретичних	04.03.2025	
	джерел.		
4.	Розробка першого та другого розділу.	18.04.2025	
5.	Розробка третього розділу.	20.05.2025	
6.	Оформлення та нормоконтроль	26.05.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2025	

Студент _____ М.С. Крюков
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.В. Столярова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка вебсервісу для моніторингу популярності відеоконтенту»: 65 с., 47 рис., 8 джерел, 2 додатки.

АУТЕНТИФІКАЦІЯ, БАЗА ДАНИХ, ВЕБСЕРВІС, МЕТРИКИ, СТАТИСТИКА, API, NODE.JS, REACT.

Об’єкт дослідження: процес розробки вебдодатку.

Предмет дослідження: реалізація вебдодатку з використанням бібліотеки React та зовнішнім API.

Мета роботи: створити вебсервіс для аналітики YouTube-каналів, який дозволяє відстежувати популярність відеоконтенту за допомогою періодичного оновлення даних через YouTube Data API.

Метод дослідження: методи збору і аналізу вимог до програмного забезпечення, моделювання і тестування.

У роботі продемонстровано процес проєктування та реалізації вебзастосунку для моніторингу популярності відеоконтенту на платформі YouTube. Окрему увагу приділено інтеграції з базою даних для зберігання отриманих метрик, а також реалізації механізмів аутентифікації користувачів для доступу до персональних налаштувань. Для розробки використано технології Node.js для серверної і React для клієнтської частини. В результаті отримали працюючий вебзастосунок. Система забезпечує стабільну роботу під час перезапуску і правильно обробляє помилки. В процесі розробки було проведено ручне тестування, помилок не виявлено.

SUMMARY

Bachelor’s qualifying paper “Development of a Web Service for Monitoring of the Video Content Popularity”: 65 pages, 47 figures, 8 references, 2 supplements.

AUTHENTICATION, DATABASE, WEB SERVICE, METRICS, STATISTICS, API, NODE.JS, REACT.

The object of the study is the process of web application development.

The subject of the study is the implementation of a web application using the React library and external APIs.

The aim of the study is to create a web service for YouTube channel analytics that allows monitoring the popularity of video content by periodically updating data via the YouTube Data API v3.

The research method involves collecting and analyzing software requirements, modeling, and testing.

This work demonstrates the process of designing and implementing a web application for monitoring the popularity of video content on the YouTube platform. Particular attention is given to integration with a database for storing the collected metrics, as well as the implementation of user authentication mechanisms for accessing personal settings. Node.js was used for server-side development, and React was used for the client side. As a result, a functioning web application was created. The system operates reliably during restarts and handles errors correctly. Manual testing was conducted during development, and no errors were found.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Скорочення та умовні познаки 7
Вступ..... 8
1 Дослідження та аналіз предметної області..... 10
1.1 Терміни..... 10
1.2 Опис предметної області 11
1.3 Функціональні та нефункціональні вимоги 12
1.4 Дослідження аналогів вебдодатку..... 13
1.5 Засоби розробки сайту..... 14
1.6 Висновки до розділу 1 19
2 Проєктування вебдодатку 21
2.1 Клієнт-серверна архітектура..... 21
2.2 Моделювання архітектури та структури даних системи 21
2.3 Макет сайту..... 32
2.4 Висновки до розділу 2 35
3 Реалізація вебдодатку 36
3.1 Архітектура вебдодатку 36
3.2 Проєктування бази даних..... 38
3.3 Розробка фронтенду..... 40
3.4 Розробка бекенду 49
3.5 Висновки до розділу 3 59
Висновки 60
Перелік посилань..... 61
Додаток А Інтерфейс сторінок додатку..... 62
Додаток Б Програмна реалізація 65

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ПЗ	Програмне Забезпечення
СУБД	Система управління базами даних
API	Application Programming Interface
BPMN	Business Process Modeling Notation
CCS	Cascading Style Sheets
CORS	Cross-Origin Resource Sharing
DOM	Document Object Model
FK	Foreign key
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDEF	Integrated DEFinition
JSON	JavaScript Object Notation
JWT	JSON Web Token
NPM	Node Package Manager
NoSQL	No Structured Query Language
PK	Primary Key
REST	REpresentational State Transfer
SEO	Search Engine Optimization
SQL	Structured Query Language
UI	User Interface
UML	Unified Modeling Language
UX	User eXperience

ВСТУП

Відстеження статистики переглядів каналів є важливою частиною роботи авторів контенту Youtube. Однак формат подачі цієї інформації на платформі часто незручний для авторів. Наразі контент-мейкерам доводиться самостійно відслідковувати популярність цікавих для них каналів. Процес займає багато часу, а якість спостереження залишає бажати кращого. Ця проблема ставить попит на сервіс, який зможе надавати цю статистику в зручному для аналізу вигляді. Таким чином, розробка вебсервісу моніторингу популярності відеоконтенту є актуальною задачею.

Мета роботи: створити вебсервіс для аналітики YouTube-каналів, який дозволяє відстежувати популярність відеоконтенту за допомогою періодичного оновлення даних через YouTube Data API.

Досягнення поставленої мети передбачає реалізацію наступних **завдань**:

– проаналізувати вимоги до додатку
– проаналізувати існуючі засоби розробки;
– дослідити документацію YouTube Data API;
– розробити макети сторінок;
– спроєктувати архітектуру вебсервісу;
– протестувати розроблений функціонал.

Об’єкт дослідження: процес розробки вебдодатку.

Предмет дослідження: реалізація вебдодатку з використанням бібліотеки React та зовнішнім API.

Метод дослідження: методи збору і аналізу вимог до програмного забезпечення, моделювання і тестування.

Для досягнення результату для клієнтської частини було обрано мову JavaScript з використанням бібліотеки React, в цей же час для серверної частини обрано Node.js, що дозволить нам легко реалізувати взаємодію між ними. Для зберігання інформації було обрано SQL систему управління базою

даних MySQL. Дані в ній зберігаються в таблицях зі строгими зв’язками, що запобігає дублювання та забезпечує цілісність.

В результаті процесу розробки було проаналізовано теоретичну частину та побудовано вебдодаток, що надає всю статистику у зручному вигляді. Творцям контенту більше не потрібно витрачати час на ручний збір

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку посилань та двох додатків. Загальний обсяг роботи становить 65 сторінок. Додатки викладено на 4 сторінках. У переліку посилань 8 використаних джерел.

Перший розділ присвячений аналізу вимог до вебсервісу та дослідженню засобів розробки. У другому розділі наведено проєктування вебсервісу, проілюстровано UML-діаграми та макети сторінок. В третьому розділі описано реалізацію фронтенд, бекенд частин та бази даних. В додатках наведено вигляд додатку та програмну реалізацію.

1 ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Терміни

Зазначимо загальні та технічні терміни щодо тексту кваліфікаційної роботи.

Вебдодаток – це комп'ютерний програмний додаток, доступ до якого здійснюється через веббраузер. Він написаний такими мовами, як HTML, JavaScript або Java, і для виконання додатку покладається на веббраузер [1].

YouTube-канал – особиста або організаційна сторінка на платформі YouTube, яка містить відеоконтент та інші матеріали.

Аналітика – процес збору, вимірювання та аналізу даних для прийняття рішень і поліпшення ефективності.

Перегляди – кількість разів, коли користувач натиснув на відео.

Метрики – це кількісні показники, що використовуються для вимірювання та оцінки ефективності або інших аспектів роботи системи.

Компонент – це інкапсульована одиниця програмного коду, яка має чітко визначену функціональність та інтерфейс для взаємодії з іншими компонентами та системами.

Аутентифікація – процес підтвердження особи користувача для забезпечення безпеки доступу до даних і функцій.

API – це набір інструкцій і інтерфейсів, що дозволяють програмним додаткам взаємодіяти між собою.

REST – це архітектурний стиль для надання стандартів між комп'ютерними системами в інтернеті, що спрощує взаємодію систем один з одним [2].

CORS – це механізм на основі HTTP-заголовків, який дозволяє серверу вказувати будь-які джерела (домен, схему або порт), окрім свого власного, з яких браузер повинен дозволити завантаження ресурсів [3].

JSON – це полегшений формат для зберігання та передачі даних, що часто використовується, коли дані надсилаються із сервера на вебсторінку [4].

Фронтенд – це публічна частина вебдодатку, що виконується у веббраузері. Вона відповідає за інтерфейс, а також реалізує логіку обробки користувальницького введення.

Бекенд – це серверна частина вебдодатку, яка прихована від користувача та виконується не на пристрої клієнта. Відповідає за обробку запитів з фронтенду, перевірку прав доступу, взаємодію з базою даних, підключення зовнішніх API та іншу логіку вебдодатку.

1.2 Опис предметної області

Система, що розробляється, є вебдодатком для відстеження статистики YouTube-каналів, яка дозволяє користувачам створювати персоналізовані набори каналів. Передбачається, що кожен набір включає детальну інформацію про канали та надає зведену таблицю відео з ключовими метриками, такими як загальна кількість переглядів, перегляди за останні 24 години та останню годину.

Дані для аналітики повинні збиратися та регулярно оновлюватися через YouTube Data API v3.

Системі слід забезпечувати стабільну роботу під час перезапуску та коректно обробляти помилки, пов’язані з обмеженнями лімітів API, забезпечуючи тим самим надійність і безперебійність процесу збору інформації.

Прогнозується, що користувачами цієї системи будуть лише автори контенту YouTube.

У разі масштабування до системи може бути доданий функціонал, який буде корисним дизайнерам авторів контенту, рекламним агентствам та партнерським програмам.

1.3 Функціональні та нефункціональні вимоги

Призначенням розробленого вебдодатку є забезпечення авторів контенту інструментом відстеження переглядів YouTube-каналів та об’єднання їх у набори. Додаток дозволяє збирати та відображати метрики каналів та відео з платформи YouTube, що допомагає користувачам відстежувати їхню активність, визначати тенденції та згодом приймати обґрунтовані рішення у виборі теми контенту. Основна мета додатку – автоматизація збору даних з YouTube, забезпечуючи швидкий доступ до показників за допомогою зручного інтерфейсу.

Перед початком розробки проведемо дослідження функціональних та нефункціональних вимог. Функціональні вимоги описують, які саме функції має виконувати вебдодаток. Нефункціональні відображають які властивості повинен мати функціонал. На основі цієї інформації визначимо архітектуру та стек технологій, побудуємо план проєкту згідно обсягу задачі.

Функціональні вимоги:

– реєстрація та аутентифікація користувачів;
– створення наборів та подальше управління;
– додавання та видалення YouTube-каналів з набору;
– періодичне оновлення даних через офіційне API;
– обробка помилок і лімітів API;
– обрахунок метрик;
– виведення статистичних даних;
– фільтрація та сортування даних.

Нефункціональні вимоги:

– продуктивність: підтримка великого обсягу даних та користувачів;
– масштабованість: здатність системи до швидкого розширення функціональних можливостей;
– інтерфейс: інтуїтивно зрозумілий інтерфейс у єдиному стилі;
– безпека: шифрування конфіденційних даних;

– зберігання даних: оновлення даних без критичного навантаження на сервер;
– надійність: статистична інформація має відповідати реальним даним.

Детальний аналіз функціональних вимог гарантує, що вебдодаток розв’язує проблему користувачів, в той час нефункціональні вимоги забезпечують якість програмного застосунку.

1.4 Дослідження аналогів вебдодатку

Перед створенням системи доцільним є дослідити наявні загальнодоступні сайти-аналоги. Серед таких можемо відокремити популярний серед авторів контенту сайт vidiq.com. VidIQ – інструмент для керування каналами YouTube, включаючи SEO та аналітику YouTube в реальному часі [5].

Перевагою vidiq.com є відображення загальної статистики Youtube-каналів, а саме:

– відображення загальних ключових метрик каналів за обраним проміжком часу у вигляді додаткової таблиці;
– відображення загальних ключових метрик каналів у вигляді стовпчастій діаграми.

Хоча це дуже цікава статистика, вона не відповідає вимогам розроблюваного додатку. Цей функціонал може бути доданий у процесі масштабування сайту.

Недоліки vidiq.com:

– надає застарілу статистику;
– неможливо створити декілька наборів;
– обмеження кількості Youtube-каналів;
– для розблокування всього функціоналу потрібно купувати платну передплату;
– функціонал працює із численними багами.

З наведених причин розглянутий сайт не вирішує поставлену проблему. Платна передплата знімає обмеження на кількість Youtube-каналів, але не впливає на інші недоліки. Відсутність актуальної статистики та наявність багів відображення робить цей функціонал сервісу не можливим до використання. Абсолютна більшість авторів контенту не використовують цей функціонал.

1.5 Засоби розробки сайту

Проведемо огляд стеку-технологій, який планується використати при розробці даного вебдодатку. Почнемо з огляду фундаменту, а саме з HTML.

HTML – це стандартизована мова розмітки для створення вебсторінок та вебзастосунків, що дозволяє задавати, як контент відображатиметься у браузері. Вона буде використовуватись для каркасу інтерфейсу, логічних блоків, форм введення, таблиць статистики та семантичної розмітки, до якої додадуться стилістичні інструкції. Незважаючи на те, що ця мова буде використана при розробці, створення власного HTML-файлу не передбачається, натомість уривки семантики стануть частиною файлів JavaScript.

CSS – це мова стилів, яка використовується для оформлення вебсторінок. Вона дозволяє визначити, як елементи HTML повинні відображатися на сторінці. Стиль можна застосувати до подібних елементів за допомогою класів, ідентифікаторів або тегів. CSS підтримує медіазапити, що дає змогу змінювати стиль сторінки залежно від пристрою або розмірів вікна. Також вона дозволяє створювати анімації та плавні переходи між станами елементів. Мова CSS не має повноцінних популярних аналогів. Існують кілька підходів, які є надбудовою CSS синтаксису, але зрештою браузер отримує та інтерпретує саме CSS-файли.

JavaScript – це високорівнева, інтерпретована мова програмування, яка використовується для створення динамічних елементів на вебсторінках.

Головним плюсом є можливість змінювати вміст вебдодатку та поведінку в реальному часі без перезавантаження сторінки. Так як передбачається створення динамічного сайту, це саме те, що потрібно.

Іншою важливою перевагою є підтримка асинхронного виконання завдань, що дозволяє, завантажувати дані з сервера, не блокуючи решту роботи сторінки. Функціональні вимоги мають на увазі роботу з великою кількістю даних, тому асинхронність критично важлива. Користувач зможе взаємодіяти з вебсторінкою, навіть коли попередній запит ще не виконано. На практиці це буде дуже корисно, наприклад, для додавання каналів без очікування на завантаження даних. У тому числі буде можливість використовувати axios – клієнт HTTP на основі промісів для браузера та Node.js, який спрощує виконання запитів та обробку відповідей у форматі JSON.

Варто зазначити, що написані JavaScript-файли за допомогою Webpack та Babel перед попаданням на клієнтський пристрій оптимізуються, щоб охопити якнайбільше клієнтів. Це гарантує, що додаток завантажуватиметься швидко та працюватиме у всіх потрібних браузерах, використовуючи лише необхідний обсяг коду.

Отже, у вебдодатку JavaScript стане головним інструментом: він забезпечить інтерактивність інтерфейсу, асинхронні запити до API, а також дозволить писати серверну логіку на тій самій мові, зберігаючи цілісність і спрощуючи розробку.

Node.js – це середовище виконання JavaScript на сервері, яке дозволяє використовувати JavaScript для розробки серверних додатків. Використовує систему npm для керування залежностями, яка надає доступ до величезної кількості бібліотек і інструментів, що дозволяють значно спростити розробку. З цих причин Node.js є популярним вибором для побудови RESTful API, вебсервісів і мікросервісів. Він прийматиме HTTP-запити з фронтенду: реєстрація користувача, авторизація, створення та керування набором каналів, фільтрація та сортування статистики. Express.js задасть маршрути та

середовища для обробки цих запитів, а також поверне відповіді у форматі JSON. Також Node.js може працювати з різними базами даних, через різноманітні бібліотеки. Таким чином, у системі Node.js забезпечить швидкий, гнучкий і масштабований бекенд, здатний забезпечити реалізацію функціональних вимог розроблюваного вебзастосунку.

Коли було визначено основний стек технологій, варто перейти до вибору фреймворка або бібліотеки. React, Vue.js і Angular – це три найбільш популярні варіанти для створення UI на основі JavaScript.

React зосереджений на створенні компонентів, що можна повторно використовувати, і працює за принципом віртуального DOM, що дозволяє швидко оновлювати лише частини сторінки. Його популярність зросла завдяки простоті, масштабованості та великій спільноті.

Vue.js дозволяє будувати інтерфейси, комбінуючи декларативний синтаксис для оновлення DOM з можливістю використання компонентів. Vue також підтримує одностороннє та двостороннє зв’язування даних, що робить його відмінним вибором для створення інтерактивних додатків з мінімумом налаштувань.

Angular забезпечує комплексне рішення для розробки вебдодатків з усіма необхідними інструментами: від маршрутизації до управління станом. Цей фреймворк базується на TypeScript, що надає переваги в типізації та підтримці складних проєктів. З цих причин Angular вибирають для великих додатків через його потужність. Він також має вбудовані інструменти для тестування.

В межах виконання даної кваліфікаційної роботи обрано бібліотеку React. Основною причиною є те, що React дозволяє будувати додатки, розділяючи їх на невеликі, самодостатні компоненти, що значно спрощує процес розробки та підтримки. Віртуальний DOM підвищує продуктивність додатків, оскільки оновлюються лише ті частини інтерфейсу, які змінилися. Це буде доречно на сторінці, в якій буде об’єднаний набір каналів, фільтрація та таблиця зі статистикою. Велика спільнота React публікує навчальні

посібники та численну кількість навчального відеоматеріалу, що робить ознайомлення з бібліотекою швидкою. Інтуїтивно зрозумілий підхід до обробки стану та компонентів у React дозволяє легко масштабувати проєкти, зберігаючи при цьому контроль над їхньою логікою та виглядом. З метою масштабування можна буде використовувати React Native для поширення програми позабраузера.

Невід'ємною частиною застосунку є база даних. Вона відіграє ключову роль у забезпеченні постійності даних, які не втрачаються після оновлення сторінки чи перезапуску сервера. В межах виконання даної кваліфікаційної роботи обрано базу даних **MySQL** завдяки своїй швидкості, надійності та легкості у використанні. Це одна з найпопулярніших реляційних баз даних з відкритим вихідним кодом, що використовує SQL для управління та маніпуляцій з даними. Ще однією важливою причиною вибору MySQL є простота інтеграції з Node.js через бібліотеку mysql2, що спрощує реалізацію серверної логіки.

Переваги MySQL:

- зберігає дані у таблицях з певною структурою, що дозволяє визначати зв'язки між даними та забезпечує виконання складних запитів;
- здатна працювати з великими обсягами даних;
- включає механізми аутентифікації, авторизації, шифрування даних та резервне копіювання;
- підтримує різні операційні системи, такі як Linux, Windows, macOS.

До бази даних MySQL було обрано надбудову **Adminer**. Це легкий інструмент для керування базами даних з відкритим кодом. Він надає простий вебінтерфейс для керування MySQL та іншими базами даних. Adminer допомагає адміністратору системи виконувати завдання, такі як управління користувачами, моніторинг даних та інше.

У процесі розробки вебзастосунку надзвичайно важливо забезпечити стабільне середовище для розробки та розгортання. Щоб досягти цього обрано платформу для контейнеризації застосунків – **Docker**. Він дозволяє створити

ізольоване середовище, де всі компоненти системи (база даних, бекенд, фронтенд) працюють у контейнерах з чітко визначеними налаштуваннями. Це дає змогу уникнути конфліктів залежностей між різними середовищами.

RESTful API використовують для доступу до даних, контенту, алгоритмів, мультимедійних файлів та інших цифрових ресурсів за URL-адресами в інтернеті. Це API, що використовує HTTP-запити для доступу до ресурсів та їхнього використання. Такими HTTP-запитами є GET, PUT, POST і DELETE [7]. У вебзастосунку буде використовуватися для організації взаємодії між клієнтською і серверною частинами за певними стандартами, наприклад щоб:

– відправити дані реєстрації користувача на сервер (POST /register);
– отримати перелік наборів (GET /sets).

У розробці вебсервісу важливим етапом є публікація та контроль версій програмного коду. **Git** – це розподілена система контролю версій, що дозволяє мати повну копію проєкту на своєму комп’ютері. На відміну від централізованих рішень всі гілки, комміти та історія зберігаються локально. Практично Git цінний своєю екосистемою. Система надає вбудовані інструменти для вирішення конфліктів та позначок версій, що спрощує випуск релізів. На платформі GitHub буде викладено програмну реалізацію готового вебдодатку.

Передостанньою складовою, яка буде використана в розробці вебдодатку є інструмент розробки дизайну інтерфейсу. У процесі роботи над кваліфікаційною роботою було виділено 2 додатки.

Adobe XD – простий та багатофункціональний інструмент для UI/UX-дизайну. У ньому зручно створювати макети, банери, прототипи та вайрфрейми. Також Adobe XD дає змогу:

– малювати векторну графіку;
– забезпечити взаємодію з інтерактивними елементами;
– адаптувати картинки під необхідні розміри;
– швидко задавати стиль.

Figma – популярний софт для прототипування і розроблення інтерфейсів із можливістю спільного редагування. Групова робота полегшує комунікацію і прискорює процес. Figma – хмарний сервіс, тому проєкти будуть автоматично зберігатися, а чернетки не загубляться. Також в Figma можна:

- малювати у високій якості;
- автоматично налаштовувати дизайн під будь-який екран;
- тестувати створені проєкти;
- одночасно працювати з безліччю файлів без шкоди для продуктивності;
- переносити проєкти з інших сервісів без втрати шрифтів;
- шрифти в Figma автоматично завантажуються з Google Fonts [6].

Фінальним інструментом є редактор коду. Редактори типу **Visual Studio Code** базуються на мінімальному ядрі з розширеннями, що додають перевірку стилю, інтеграцію з Git та інше. Модульний підхід дає гнучкість налаштування. Для розробки вебзастосунку буде використано програмний застосунок Visual Studio Code.

Також будуть використані доповнення:

- підсвітка синтаксису та автодоповнення;
- підтримка фреймворків;
- інструменти для Docker, Git, ESLint тощо.

1.6 Висновки до розділу 1

В результаті аналізу вимог та існуючих аналогів для розробки вебдодатку аналітики YouTube-каналів було сформовано наступний мінімально необхідний стек технологій та інструментів:

- фронтенд: JavaScript з залученням бібліотеки React;
- бекенд: Node.js, RESTful API, аутентифікація через JWT;

- база даних: MySQL, адміністрування через Adminer;
- контейнеризація: Docker Desktop та Docker Compose для ізоляції сервісів;
- автоматизація: node-cron для періодичного оновлення даних;
- контроль версій: Git із GitHub;
- дизайн та прототипування: Figma для макетів інтерфейсу;
- розробка та налагодження: Visual Studio Code із розширеннями;
- джерело первинної інформації: інтеграція з YouTube Data API v3.

Отже, сформований стек технологій повністю відповідає функціональним та нефункціональним вимогам до системи. Обрані інструменти дозволяють ефективно реалізувати клієнтську частину з динамічним інтерфейсом, надійний серверний функціонал з захищеною аутентифікацією, стабільну роботу з базою даних, а також автоматизоване оновлення даних. Інтеграція з офіційним YouTube API гарантує актуальність отримуваної інформації, а контейнеризація спрощує розгортання та підтримку додатку в різних середовищах.

2 ПРОЄКТУВАННЯ ВЕБДОДАТКУ

2.1 Клієнт-серверна архітектура

Клієнт-серверна архітектура реалізує розподілену модель, де клієнт надсилає запит до сервера, а сервер обробляє його відповідно до вбудованої логіки та повертає результат. Клієнт відповідає за інтерфейс і формування запитів, сервер – за обробку.

Серверна частина поділена на три рівні:

– мережевий: приймає й відсилає повідомлення через HTTP;
– прикладний: реалізується основна логіка й алгоритми;
– рівень зберігання: інтеграція із базою даних.

Поділ на рівні дозволяє оновлювати один із шарів без впливу на інші. Отже, клієнт-серверна архітектура забезпечує розподіл обов’язків між вузлами системи, спрощує підтримку коду та дає змогу гнучко масштабувати кожен компонент окремо. Чітке розмежування ролей спрощує розробку і підтримку коду, адже кожен компонент сфокусований на своїй задачі. Окреме виділення рівнів забезпечує гнучкість. Кожен шар можна змінювати без залучення інших. Дана архітектура реалізує функціональні та нефункціональні вимоги до вебдодатку.

2.2 Моделювання архітектури та структури даних системи

UML-діаграми – це важливий інструмент візуального проєктування, який дозволяє створювати добре задокументоване програмне забезпечення. UML-діаграми потрібні для того, щоб візуально моделювати структуру, поведінку та взаємозв’язки компонентів програмного забезпечення. Основні причини, чому UML-діаграми потрібні у ході розробки вебзастосунку:

– формалізація вимог;
– проєктування архітектури;
– аналіз логіки та сценаріїв роботи;
– полегшення тестування та підтримки.

Функціональна схема системи – це графічне зображення основних функцій і потоків інформації між ними, що відображає послідовність обробки даних, взаємодію компонентів та зв'язки типу «вхід-обробка-вихід» для аналізу й проєктування архітектури. На рисунку 2.1 зображено функціональну схему системи.

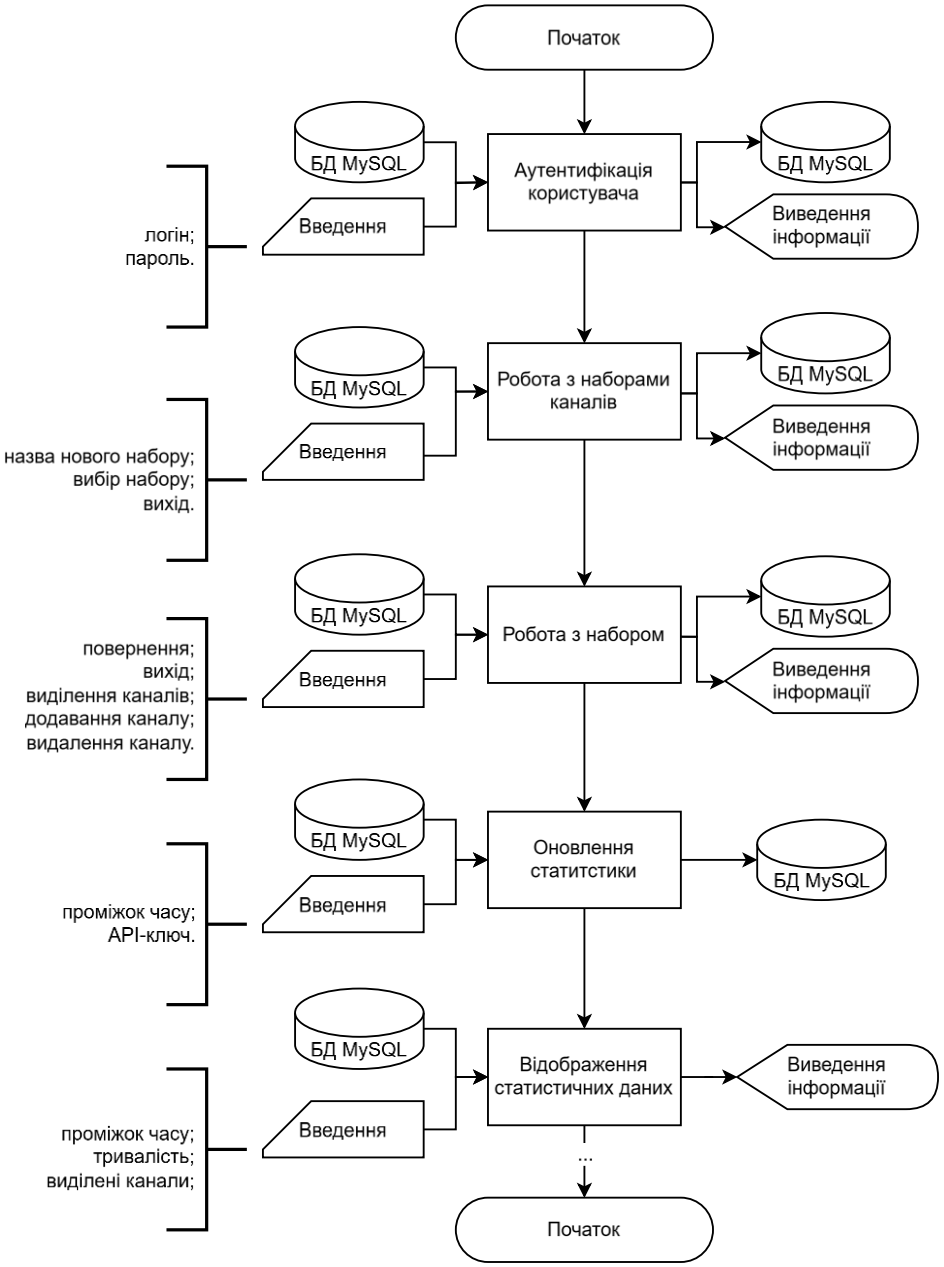


Рисунок 2.1 – Функціональна схема системи

З наведеної функціональної схеми видно, що вебзастосунок розбивається на 5 основних блоків:

– аутентифікація користувача;
– робота з наборами каналів;
– робота з набором;
– оновлення статистики;
– відображення статистики.

Структурна схема модулів – це графічна модель, яка відображає розбиття системи на модулі, їх ієрархічні зв’язки та точки взаємодії. Вона демонструє, які інтерфейси використовують модулі для обміну даними й викликів функцій, що спрощує аналіз залежностей і забезпечує впорядковану архітектуру проєкту. На рисунку 2.2 зображено структурну схему модулів.

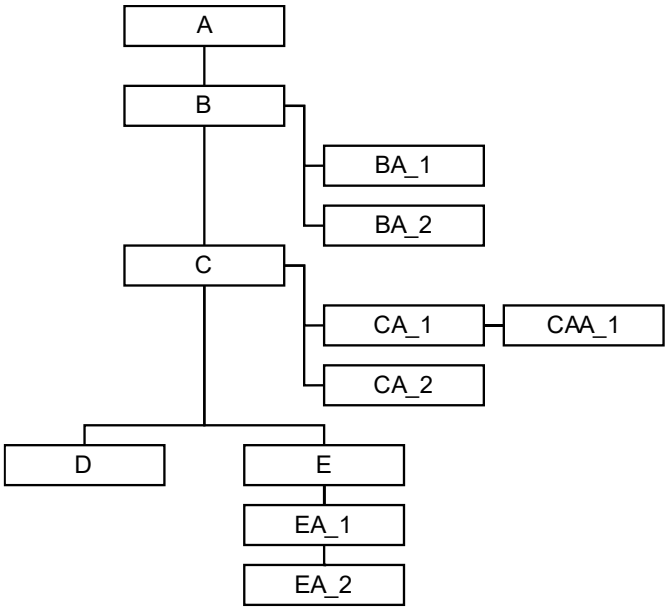


Рисунок 2.2 – Структурна схема модулів

Модуль A – у цьому модулі відбувається аутентифікація користувача. Запускає модуль B, коли користувач авторизований.

Модуль B – викликається з модуля A для роботи з наборами. Запускає модуль C, коли користувач обирає набір.

Модуль BA_1 – викликається з модуля B для створення нового набору каналів.

Модуль BA_2 – викликається з модуля B для видалення існуючого набору каналів.

Модуль C – викликається з модуля B для роботи з обраним набором.

Модуль CA_1 – викликається з модуля C, відповідає за додавання нового YouTube-каналу до обраного набору.

Модуль CAA_1 – викликається з модуля C, реалізує перевірку правильності юзернейму каналу перед додаванням у систему.

Модуль CA_2 – викликається з модуля C, відповідає за видалення YouTube-каналу до обраного набору.

Модуль D – здійснює періодичне оновлення статистики.

Модуль E – викликається з модуля B для роботи з відображенням.

Модуль EA_1 – формує статистичні дані.

Модуль EA_2 – реалізує фільтрування та сортування статистичних даних.

Нотація BPMN – це стандарт графічного моделювання бізнес-процесів, що дозволяє чітко формалізувати алгоритми виконання та взаємодію між учасниками. Переваги нотації BPMN:

– зрозумілість для всіх учасників процесу;
– стандартизація та уніфікація;
– можливість моделювання складних процесів;
– підтримка ролей і відповідальностей.

Діаграма BPMN чітко відображає послідовність і взаємодію між користувачем та системними модулями: від аутентифікації до роботи з наборами каналів і подальшого відображення статистики. Використання розгалужень підкреслює розмежування відповідальностей та дозволяє моделювати як ручні дії користувача, так і автоматизоване періодичне оновлення статистики. Такий підхід сприяє прозорості процесів, полегшує аналіз і оптимізацію, а також дозволяє легко інтегрувати модель у BPM-систему для автоматизації.

На рисунку 2.3 зображено нотацію BPMN.

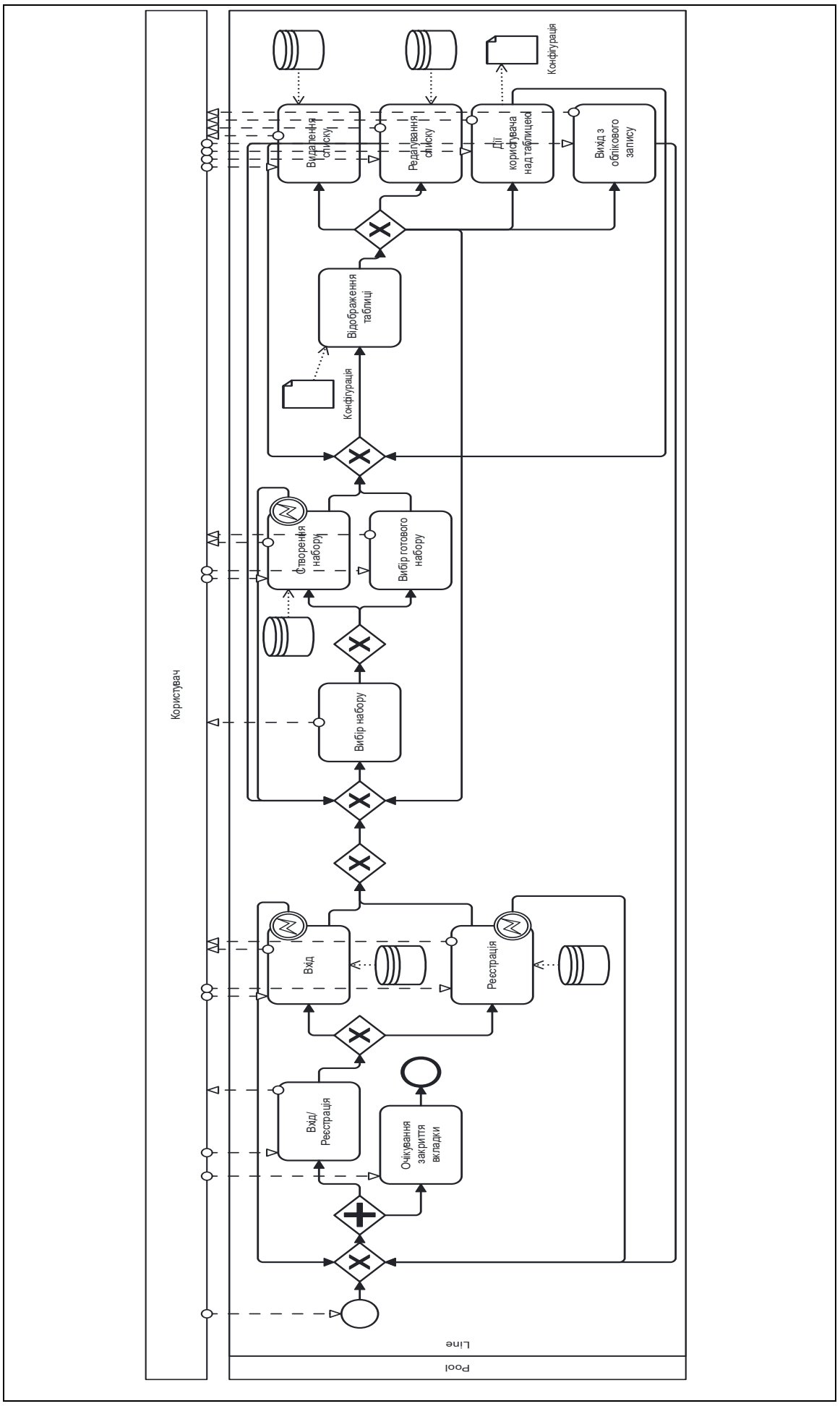


Рисунок 2.3 – Нотація BPMN

Діаграма прецедентів – це модель, яка відображає функціональні вимоги системи через набір сценаріїв та їх взаємодію із зовнішніми суб'єктами (акторами). На рисунку 2.4 зображено діаграму прецедентів.



Рисунок 2.4 – Діаграма прецедентів

Ключові елементи:

– актори – ролі, що взаємодіють із програмою;
– прецеденти – конкретні варіанти використання системи, завдання чи серії дій;
– асоціації – зв'язки між акторами і прецедентами, що показують участь.

Діаграма прецедентів допомагає формалізувати сценарії використання, узгодити очікування замовника й розробників та виявити основні функції ще на початковому етапі аналізу.

Діаграма послідовності – це UML-модель, яка показує хронологічний порядок обміну повідомленнями між об’єктами в межах сценарію: лінії життя позначають об’єкти, стрілки – виклики методів, а прямокутники на лініях життя – виконання операцій. На рисунку 2.5 зображено прецедент «Додавання каналу до списку».

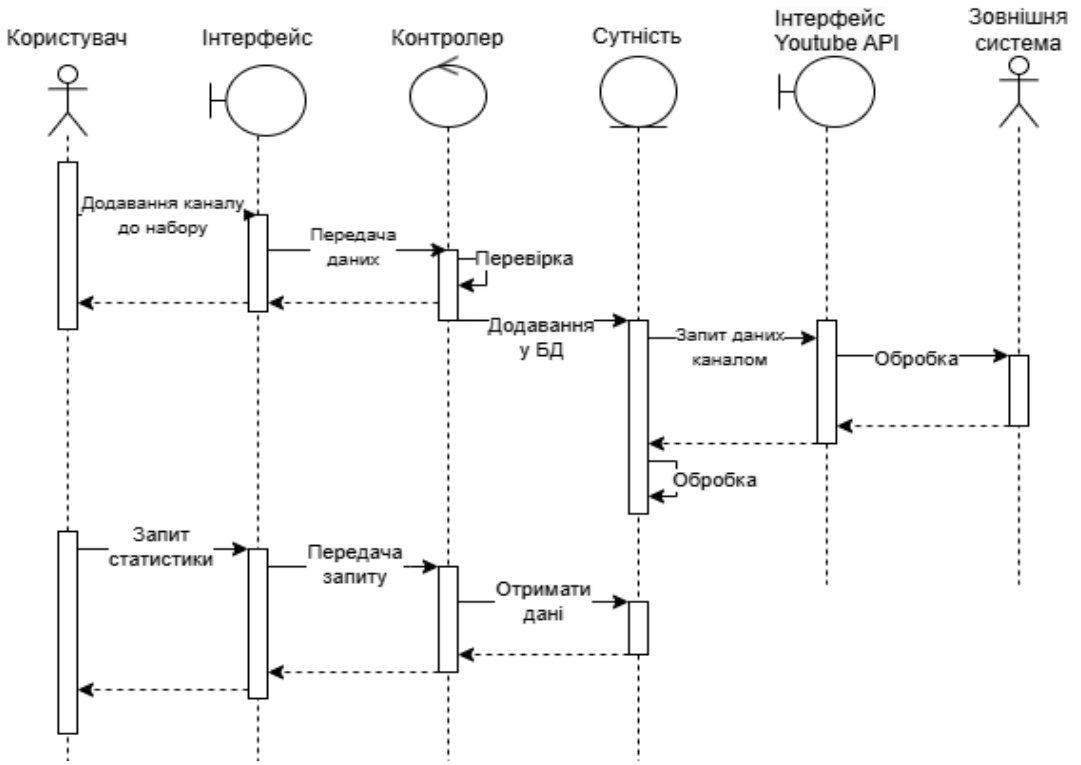


Рисунок 2.5 – Діаграма послідовності для прецеденту «Додавання каналу до списку»

Діаграми кооперації – це UML-модель, що показує об’єкти та їх зв’язки, якими пронумеровано передаються повідомлення, щоб відобразити порядок і структуру взаємодії.

На рисунку 2.6 зображено прецедент «Додавання каналу до списку». Зв’язки, що представлені на рисунку:

– 1: додати канал(): користувач вводить юзернейм каналу;
– 2: отримати дані(): відправляє запит до YouTube API по юзернейму каналу;
– 3: зберегти дані(): інформація отримана від YouTube зберігається в базу даних;

– 4: помилка(): якщо під час звернення до API сталася помилка, YouTube API повертає код помилки у модуль;
– 5: передати статус(): повертає користувачу підтвердження або помилку;
– 6: запитати таблицю(): користувач виділив канал;
– 7: сформулювати запит(): створення SQL-запиту;
– 8: запит даних(): реалізація SQL-запиту;
– 9: надіслати дані(): отримання SQL-звіту;
– 10: надіслати дані(): сформувати JSON-відповідь;
– 11: відобразити таблицю(): відобразити таблицю на фронтенді.

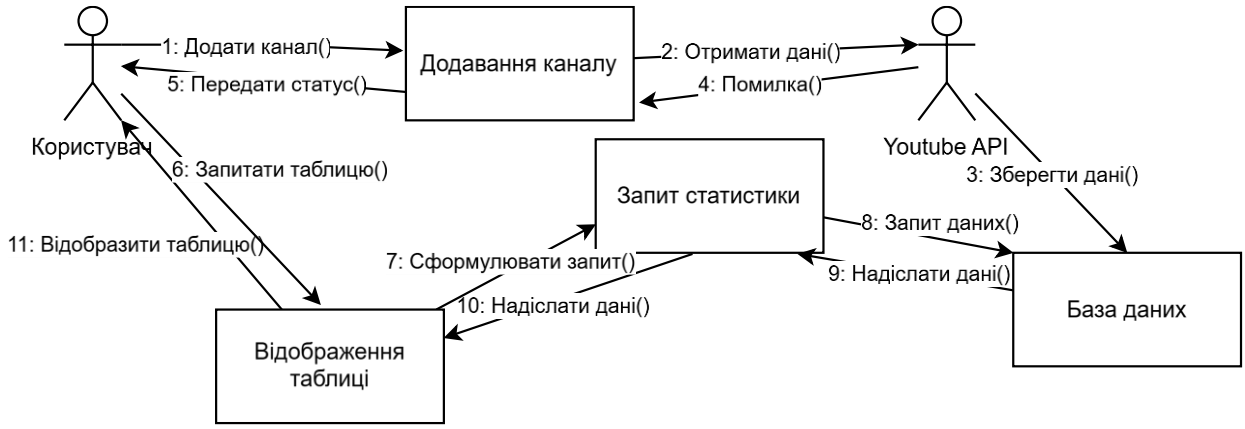


Рисунок 2.6 – Діаграми кооперації

Діаграма станів – це UML-модель, яка описує множину станів об’єкта та переходи між ними під впливом подій, відображаючи внутрішню поведінку системних елементів у різні моменти часу. На рисунку 2.7 зображено діаграму станів.

Дана діаграма детально описує перші три модулі, а саме аутентифікацію, роботу з списком наборів та визначеним набором. Робота з таблицею дуже спрощена, щоб не перевантажувати малюнок банальною фільтрацією, сортуванням. У наступних діаграмах ця частина теж буде скорочена. Модуль із оновленням статистики не залежить від стану об’єкта, а працює автономно із заздалегідь визначеним проміжком часу.

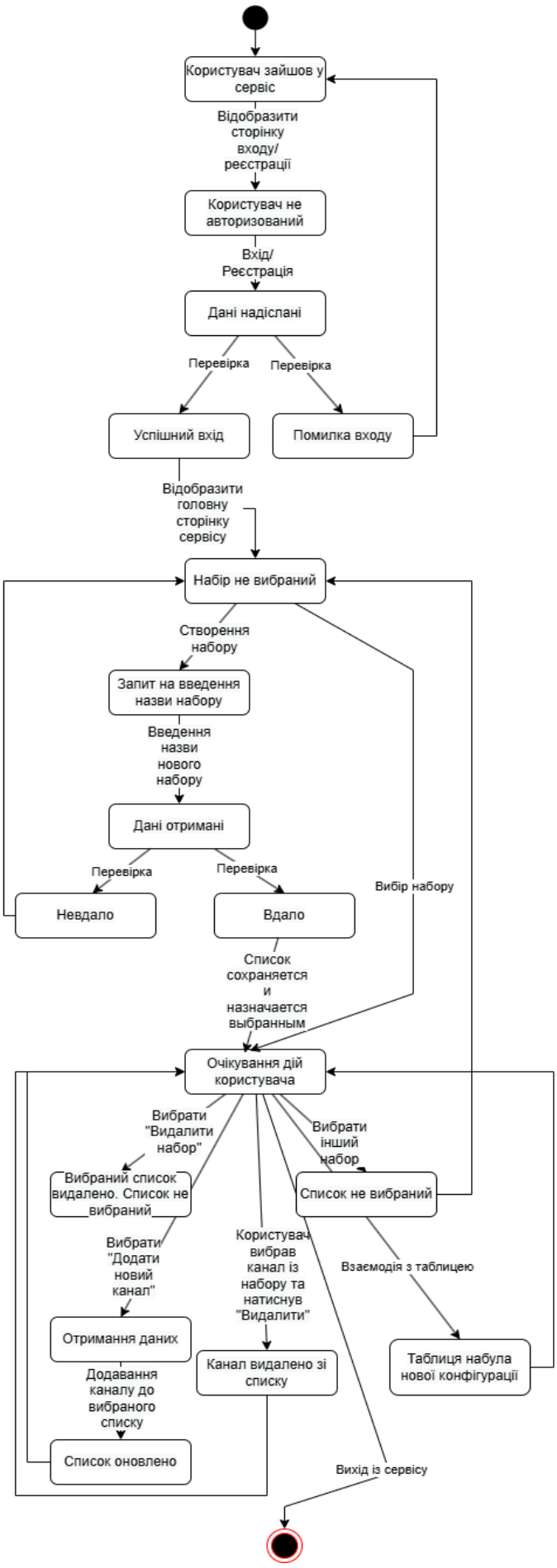


Рисунок 2.7 – Діаграма компонентів

Діаграми діяльності – це UML-моделі, що ілюструють послідовність та паралельність дій у процесі виконання сценаріїв, відображаючи потоки керування й обміну ресурсами між активностями. На рисунку 2.8 зображено діаграму діяльності.

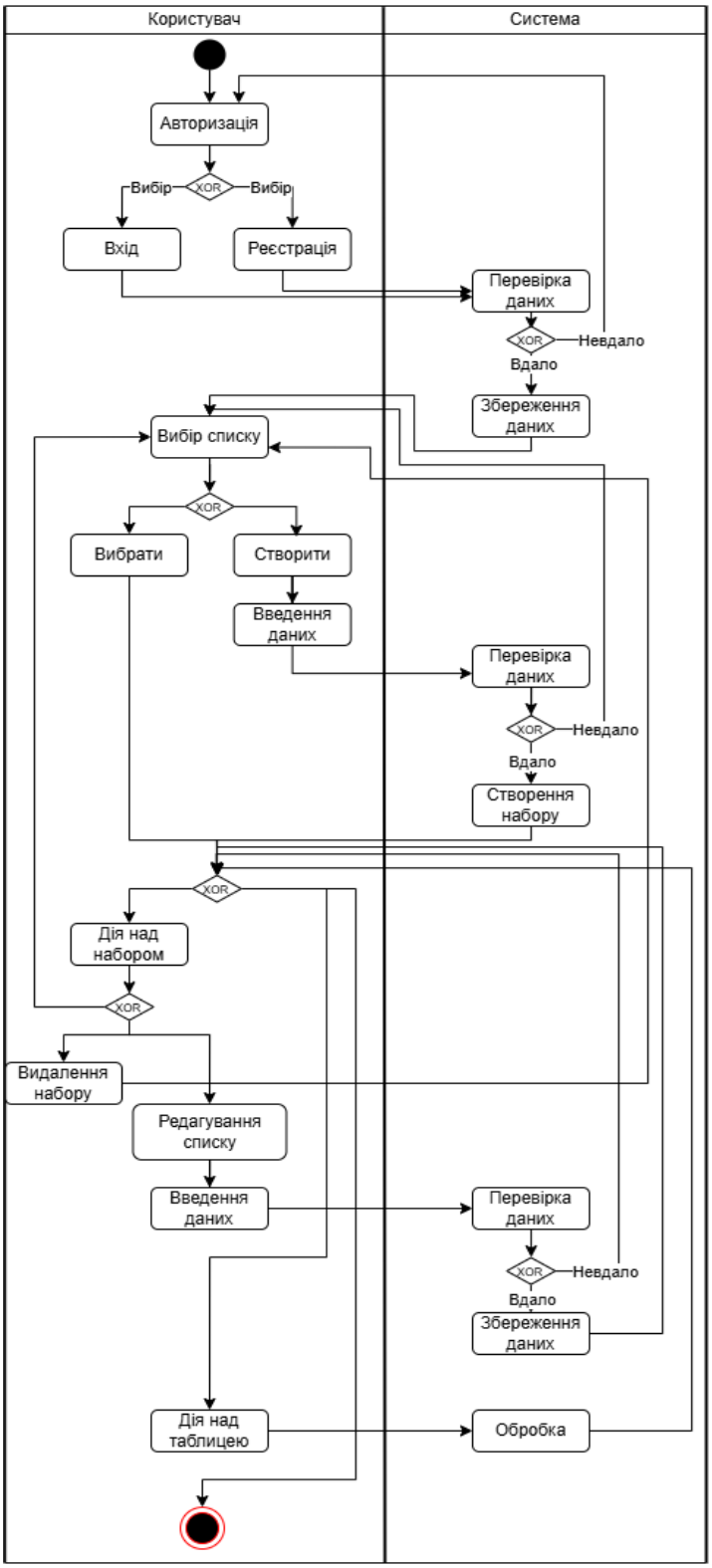


Рисунок 2.8 – Діаграми діяльності

Після входу користувач переходить до вибору списку каналів – може обрати існуючий набір або створити новий. У випадку створення він вводить дані, система перевіряє їхню коректність та створює новий набір у базі.

Далі над обраним набором здійснюються операції: видалення, редагування списку або додавання нових каналів. Кожного разу після введення змін система проводить валідацію та зберігає оновлену інформацію.

У завершальному етапі користувач може виконувати дії над таблицею даних, наприклад, сортування чи фільтрація.

В рамках роботи над базою даних також слід розробити **ER-діаграму**. IDEF1 – це методологія сімейства IDEF, призначений для формалізації структур даних і їх взаємозв'язків у рамках проєктування інформаційних систем. Він використовує графічні нотації для відображення сутностей, атрибутів та зв'язків. На рисунку 2.9 зображено діаграму IDEF1.

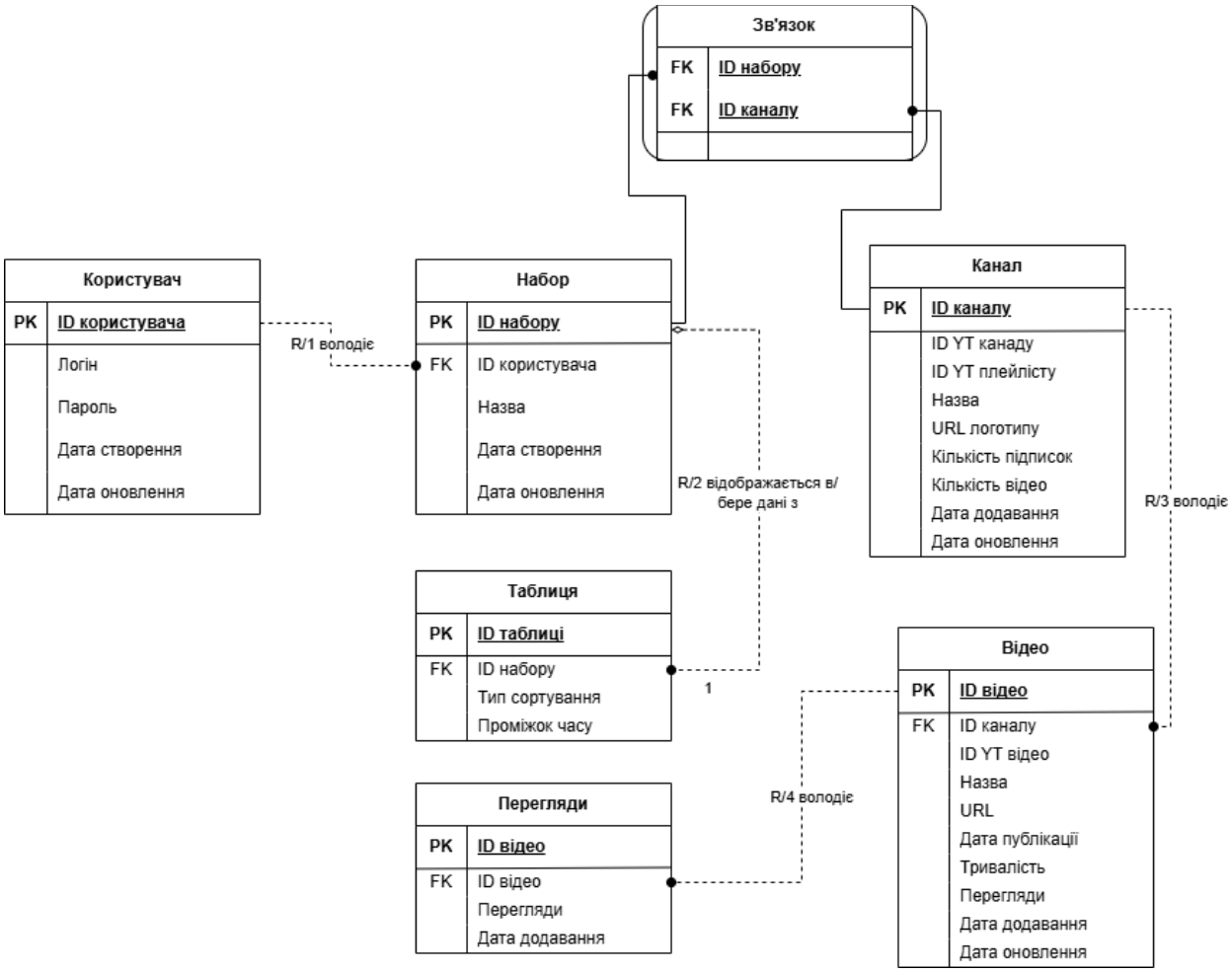


Рисунок 2.9 – IDEF1

Загалом, дана ER-модель покликана гарантувати коректне зберігання та оновлення даних про користувачів, набори каналів, канали та їхню статистику в межах функціональних вимог.

2.3 Макет сайту

Макет вебсайту – це статичне візуальне представлення макета, дизайну та інтерфейсу користувача. Він служить схемою для структури вебсайту та допомагає дизайнерам і клієнтам візуалізувати кінцевий продукт. Макет зазвичай включає такі елементи, як заголовок, меню навігації, розділи вмісту, бічна панель, нижній колонтитул та будь-які інші важливі функції вебсайту [8].

На основі макету легко перевірити зручність користування (UX): протестувати навігацію, послідовність кроків у формі, розташування кнопок та важливих елементів. Якщо користувачі стикаються з труднощами ще на рівні прототипу, це значно пришвидшує виправлення недоліків.

На рисунку 2.10 зображено макет головної сторінки.

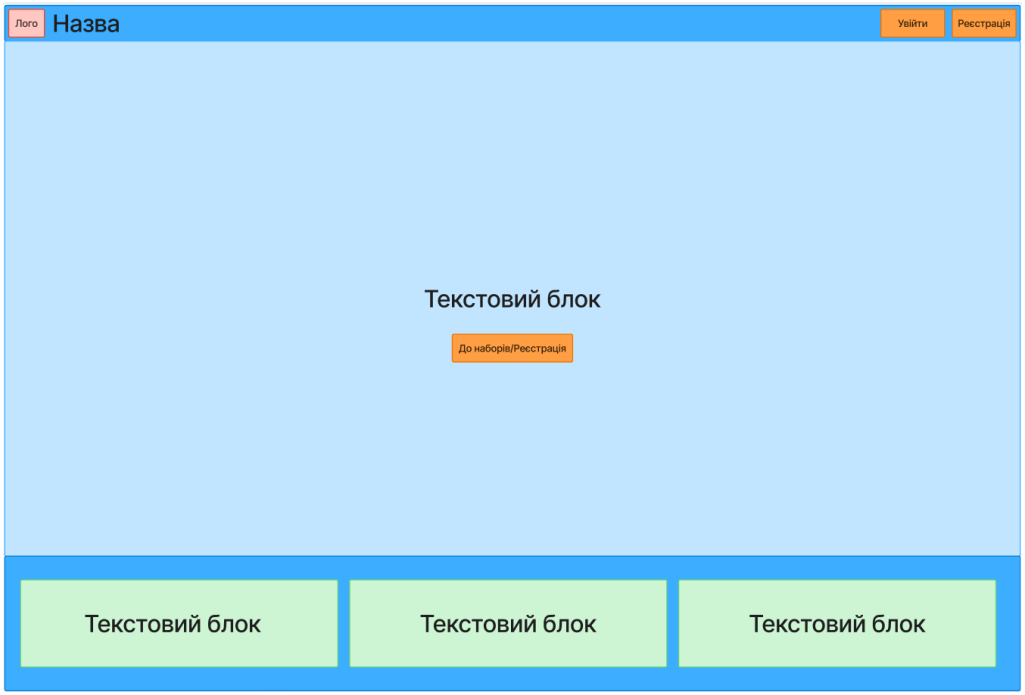


Рисунок 2.10 – Макет головної сторінки

Шапка завжди фіксована та одразу має навігацію до ключових кнопок авторизації та реєстрації (див рис. 2.10). Секція посередині із текстовою частиною наголошує на кнопку «До наборів/Реєстрація». Список трьох блоків унизу відповідає за швидке ознайомлення з перевагами сайту.

На рисунку 2.11 зображено макет сторінки реєстрації та авторизації. Центрування форми мінімізує відволікання і робить фокус користувача безпосередньо на полях вводу. Однорідність полів та кнопок створює зрозумілу візуальну ієрархію.

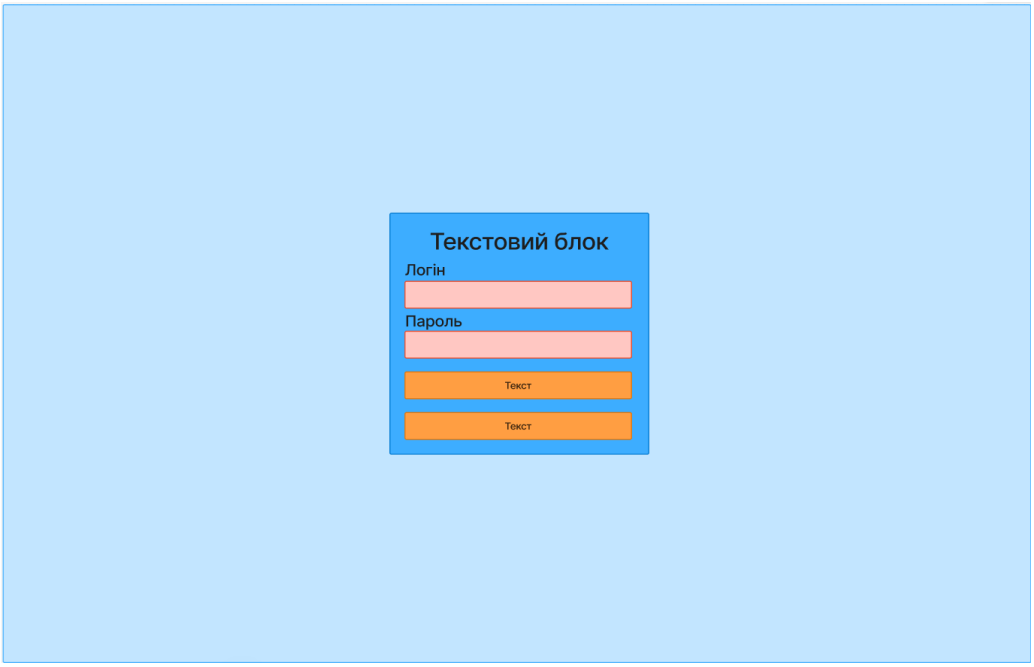


Рисунок 2.11 – Макет сторінки реєстрації та авторизації

На рисунку 2.12 зображено макет сторінки з списком наборів. Вертикальний список наборів впорядковує інформацію у вигляді: назва + дія.

На рисунку 2.13 зображено макет сторінки з статистичної інформацією. Ліва панель із заголовком «Назва набору» та списком каналів виконує роль пульта керування. Користувач обирає або додає/видаляє канал. Верхні кнопки «Повернення» та «Вихід» завжди доступні – вони легко помітні над списком і дозволяють швидко змінити набір або вийти з облікового запису. Права панель утримує детальну інформацію про відео, відгороджуючи дані від навігаційної частини. Зверху розташовані фільтри таблиці. Колонки з метриками вирівняні

в табличному вигляді для полегшеного порівняння чисел. Назви стовпців є кнопками, які перемикають сортування стовпців за зменшенням/зростанням.

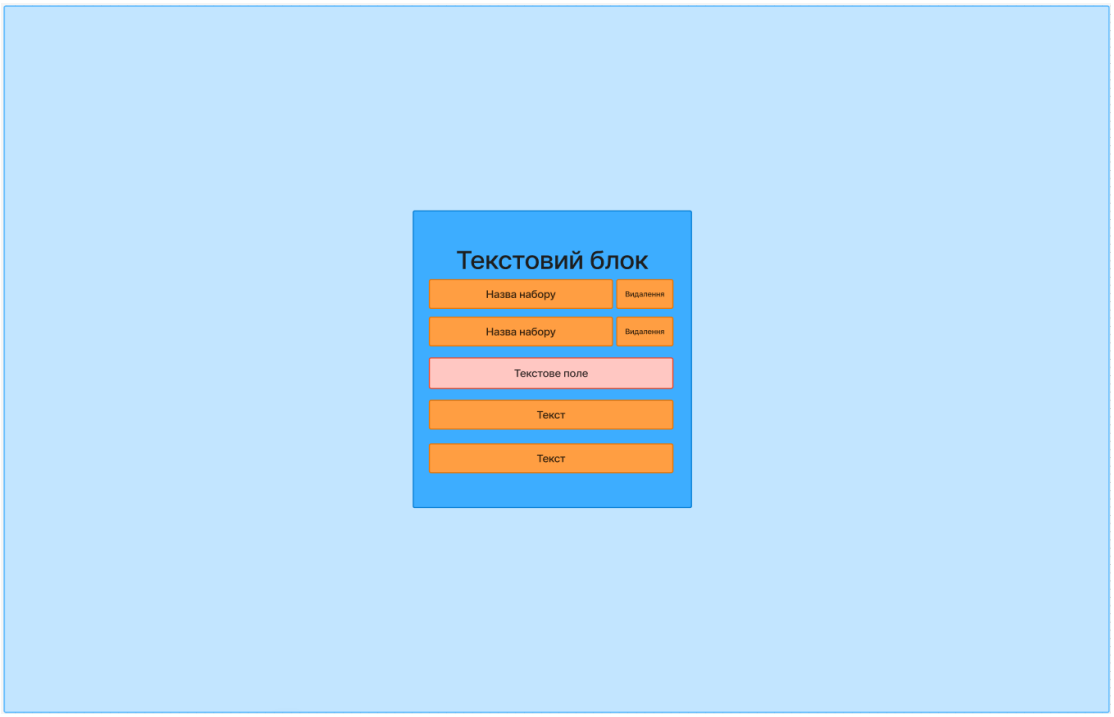


Рисунок 2.12 – Макет сторінки з списком наборів

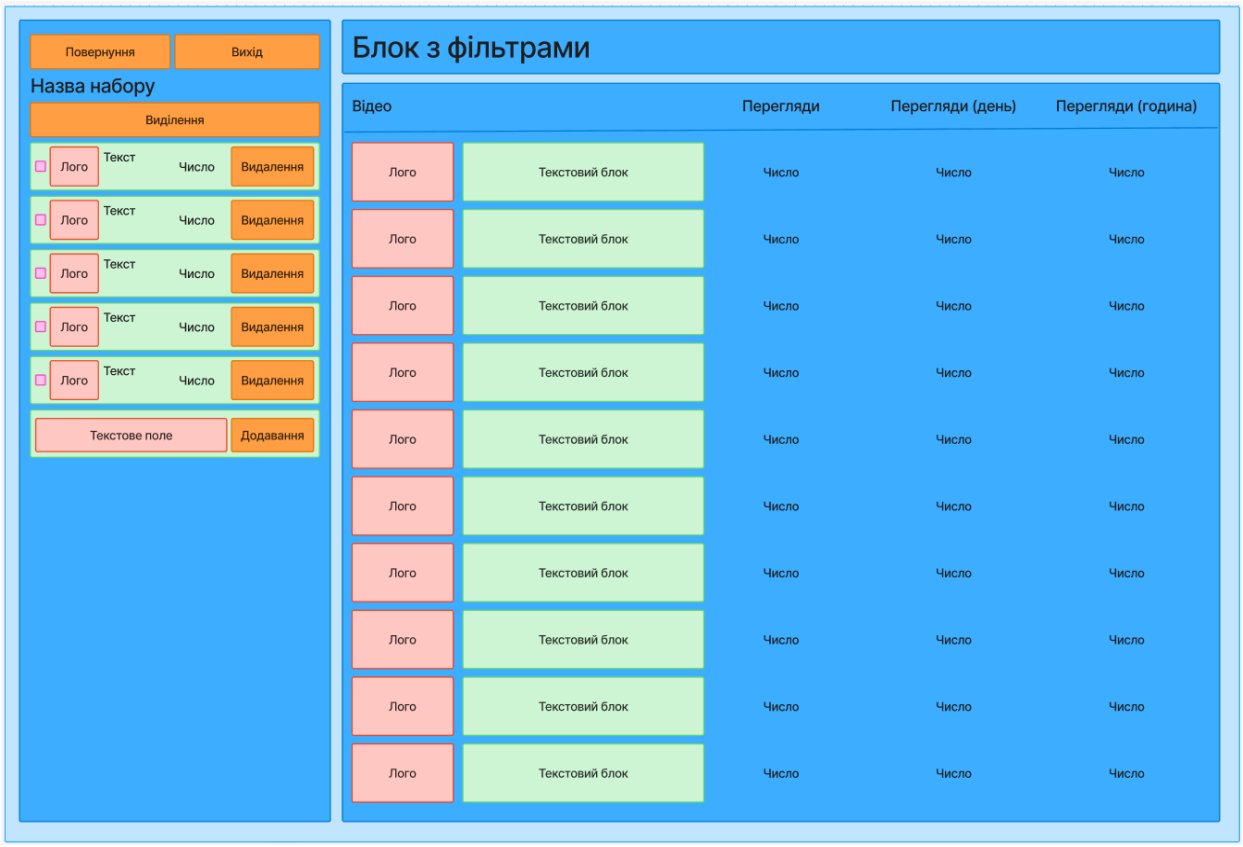


Рисунок 2.13 – Макет сторінки з статистичної інформацією

За наявності макетів сторінок легше розбити розробку вебдодатку на етапи. Такий підхід дозволяє паралельно виконувати дизайн і розробку окремих блоків HTML, що пришвидшує розробку.

2.4 Висновки до розділу 2

У сукупності розглянуті теми формують цілісну методологію проєктування інформаційної системи, що поєднує архітектурні рішення, моделювання бізнес-логіки і візуалізацію інтерфейсу.

Клієнт-серверна архітектура задає базовий розподіл обов’язків між вузлами. Набір UML-діаграм, від прецедентів до діаграм активності та станів, дозволяє описати як статичну структуру, так і динамічну поведінку системи. Функціональна та структурна схеми акцентують увагу на потоках інформації й внутрішній модульності, сприяючи чіткому розмежуванню відповідальностей. Додаткові нотації BPMN і IDEF1X – розширюють можливості формалізації бізнес-процесів та даних, зменшуючи неоднозначності при проєктуванні вебзастосунку. Макет сайту візуалізує концепцію, яка закладає основу майбутнього інтерфейсу.

Таким чином, застосування описаних підходів забезпечує всебічну перевірку архітектурних й функціональних аспектів проєкту ще на ранніх стадіях. Це призводить до мінімізації ризику розбіжностей та забезпечує послідовну реалізацію від моделі до готового вебдодатку.

3 РЕАЛІЗАЦІЯ ВЕБДОДАТКУ

У цьому розділі описано деталі практичної реалізації вебдодатку моніторингу YouTube-каналів, а також методи його тестування. Розглянуто структуру проєкту, налаштування середовища, опис основних компонентів фронтенду та бекенду, організацію бази даних, обмін з YouTube API, механізми валідації, аутентифікацію, оптимізацію та результати тестування.

3.1 Архітектура вебдодатку

Проєкт організовано за клієнт-серверною архітектурою з трирівневим поділом:

- фронтенд (/frontend) – створений на React, відповідає за відображення інтерфейсу та взаємодію з користувачем;
- бекенд (/backend) – реалізований на Node.js, обробляє API-запити, виконує логіку та керує доступом до бази даних;
- база даних – MySQL, зі скриптом ініціалізації init.sql для створення таблиць.

До складу репозиторію також входить docker-compose.yml. Цей файл відповідає за підняття контейнерів бекенду, фронтенду, MySQL та Adminer. Виконання docker-compose up автоматично піднімає контейнери описані у файлі docker-compose.yml, що зображено на рисунку 3.1.

Контейнер db використовує найновішу офіційну версію MySQL. Змінні задають початкову конфігурацію бази. Також прокидається порт MySQL назовні, щоб до БД можна було підключитися з хоста або інших контейнерів. data:/var/lib/mysql – зберігає дані бази в названому томі, щоб вони не губилися при перезапуску контейнера. ./init.sql:/docker-entrypoint-initdb.d/init.sql – дозволяє один раз при старті виконати скрипт ініціалізації. Щоб контейнери бачили один одного, контейнер підключається до спільної ізольованої мережі.

docker-compose.yml

```yaml
services:
  db:
    image: mysql:latest
    container_name: db
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: monitoring
      MYSQL_USER: user
      MYSQL_PASSWORD: password
    ports:
      - "3306:3306"
    volumes:
      - data:/var/lib/mysql
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql
    networks:
      - network
  adminer:
    image: adminer
    container_name: adminer
    depends_on:
      - db
    ports:
      - "8080:8080"
    environment:
      ADMINER_DEFAULT_SERVER: db
    networks:
      - network
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: backend
    depends_on:
      - db
    environment:
      DB_HOST: db
      DB_USER: user
      DB_PASSWORD: password
      DB_NAME: monitoring
      JWT_SECRET: '4f3d2e1a8b7c6d5e4f3a2b1c8d9e0f7a3b1c4d6e7f8a9b0c1d2e3f4a5b6c7d8'
      YT_API_KEY: AIzaSyBYak5avvSsB8f04oEceBQiO0nJLrcBmjQ
    ports:
      - "5000:5000"
    networks:
      - network
  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    container_name: frontend
    depends_on:
      - backend
    ports:
      - "3000:3000"
    volumes:
      - /app/node_modules
      - ./frontend:/app
    environment:
      CHOKIDAR_USEPOLLING: "true"
    tty: true
    networks:
      - network
volumes:
  data:
networks:
  network:
    driver: bridge
```

Рисунок 3.1 – Файл docker-compose.yml

Контейнер adminer гарантує, що MySQL стартує перед Adminer’ом. Також прокидається порт 8080.

ADMINER_DEFAULT_SERVER: db – вказує за замовчуванням сервер бази даних під ім’ям db.

Контейнер backend:

– Dockerfile будує образ із Dockerfile у локальній папці ./backend;
– depends_on: db чекає на старт MySQL перед запуском бекенду;
– DB_HOST, DB_USER, DB_PASSWORD, DB_NAME – передає параметри підключення до БД через змінні середовища;
– JWT_SECRET – секретний ключ для підпису JWT-токенів;
– YT_API_KEY – API-ключ для Youtube Data Api v3.

Контейнер frontend:

– Dockerfile будує образ із Dockerfile у локальній папці ./frontend;
– depends_on: backend чекає на старт backend перед запуском фронтенду;
– /app/node_modules – щоб залежності не переписувалися при монтуванні коду;
– ./frontend:/app – монтує локальну папку з кодом, дає можливість розробляти з перезавантаженням;
– CHOKIDAR_USEPOLLING: "true" – змушує систему файлів опитувати зміни, щоб hot-reload правильно працював у контейнері;
– tty: true – залишає термінал відкритим.

Volumes: data – назва тому для зберігання даних MySQL незалежно від життєвого циклу контейнера.

Networks: network з драйвером bridge створює ізольовану мережу, де контейнери бачать один одного за іменами сервісів і не відкриті безпосередньо назовні.

3.2 Проєктування бази даних

Усі визначення таблиць будуть зосереджені в одному файлі. Так як ми вже розробляли модель IDEF1, в ході роботи залишилося тільки написати код.

На рисунку 3.2 зображено файл init.sql.

init.sql

```sql
CREATE TABLE IF NOT EXISTS users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  login VARCHAR(255) NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS sets (
  id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT NOT NULL,
  name VARCHAR(255) NOT NULL,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS channels (
  id INT AUTO_INCREMENT PRIMARY KEY,
  yt_channel_id VARCHAR(50) NOT NULL,
  uploads_id VARCHAR(50) NOT NULL,
  title VARCHAR(255),
  logo_url VARCHAR(500),
  subscriber_count BIGINT DEFAULT 0,
  video_count INT DEFAULT 0,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS set_channels (
  id INT AUTO_INCREMENT PRIMARY KEY,
  set_id INT NOT NULL,
  channel_id INT NOT NULL,
  UNIQUE KEY (set_id, channel_id)
);

CREATE TABLE IF NOT EXISTS videos (
  id INT AUTO_INCREMENT PRIMARY KEY,
  yt_video_id VARCHAR(50) NOT NULL,
  channel_id INT NOT NULL,
  title VARCHAR(255),
  thumbnail_url VARCHAR(500),
  publish_date DATETIME,
  duration INT NOT NULL,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS video_stats (
  id INT AUTO_INCREMENT PRIMARY KEY,
  video_id INT NOT NULL,
  view_count BIGINT DEFAULT 0,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Рисунок 3.2 – Файл init.sql

Файл init.sql створює таблиці users, sets, channels, set_channels, videos, video_stats. Це мінімальний набір даних, які потрібні нам під час виконання функціональних вимог.

3.3 Розробка фронтенду

Для початку виділимо сторінки, які мають бути реалізовані у вигляді компонентів:

- App є точкою входу програми, в яку генеруються всі компоненти;
- MainPage – головна сторінка з короткою інформацією про вебдодаток;
- LoginPage – форма авторизації з контролем полів логін та пароль;
- RegisterPage – форма регістрації з контролем полів логін та пароль;
- SetsPage – відображає перелік наборів користувача, дозволяє додавати/видаляти набори;
- TablePage – показує канали та статистику обраного набору.

Також повинні бути реалізовані допоміжні компоненти, які не є цілою сторінкою, а містять лише конкретний шаблон елемента:

- PrivateRoute – обгортка для захищених сторінок, перевіряє наявність валідного JWT у localStorage та при необхідності перенаправляє на /login;
- FilteredTable – компонент фільтрації даних у таблиці, відображає списки для вибору проміжку часу та тривалості відео;
- Loading – компонент індикатора завантаження, що виводить спіннер під час очікування;
- SidePanel – бокова панель керування набором каналів; відображає список каналів із чекбоксами для вибору, дозволяє додавати або видаляти канали, а також виходити з облікового запису;
- VideoTable – компонент таблиці відео з підтримкою сортування за різними метриками (дата публікації, загальна кількість переглядів тощо), форматуванням даних і виведенням детальної інформації про кожне відео;
- Pagination – компонент управління навігацією сторінок, показує поточну сторінку та загальну кількість сторінок, а також містить поле введення для переходу на обрану сторінку.

Вже для розроблених компонентів потрібні CSS-файли, які відповідають за візуальний вигляд компонентів. Загальний стиль сайту у сучасній темній палітрі з глибокими відтінками синьо-чорного та градієнтами. Кнопки оформлені в яскраво-блакитному кольорі. Всі блоки мають округлі кути.

Розберемо структуру найскладнішої сторінки вебдодатку, а саме TablePage. Розпочнемо з використаних useState, що зображені на рисунку 3.3:

– loading – булевий прапорець, що визначає, чи відображати компонент завантаження (<Loading />);
– videos – масив об’єктів відео для поточної сторінки;
– total – загальна кількість відео в наборі, потрібна для розрахунку кількості сторінок у пагінації;
– setName – назва поточного набору каналів;
– channels – масив об’єктів каналів, що входять до набору;
– isChannelsInitialized – булевий прапорець, який сигналізує, чи вже було виконано початкову установку вибраних каналів після завантаження channels;
– selectedChannelIds – множина ідентифікаторів каналів, що наразі вибрані для фільтрації відео;
– page – номер поточної сторінки пагінації;
– dateFilter – рядок із вибраним фільтром за датою публікації відео;
– durationFilter – рядок із вибраним фільтром за тривалістю відео;
– sortField – поле, за яким сортуються відео (‘publish_date’, ‘views’ тощо);
– sortOrder – напрямок сортування (‘asc’ або ‘desc’).

```
19    const [loading, setLoading] = useState(true);
20    const [videos, setVideos] = useState([]);
21    const [total, setTotal] = useState(0);
22    const [setName, setSetName] = useState('');
23    const [channels, setChannels] = useState([]);
24    const [isChannelsInitialized , setisChannelsInitialized ] = useState(false);
25    const [selectedChannelIds, setSelectedChannelIds] = useState(new Set());
26    const [page, setPage] = useState(1);
27
28    const [dateFilter, setDateFilter] = useState('all');
29    const [durationFilter, setDurationFilter] = useState('all');
30    const [sortField, setSortField] = useState('publish_date');
31    const [sortOrder, setSortOrder] = useState('desc');
```

Рисунок 3.3 – Використані useState

Коли будь-який із цих станів оновлюється, React автоматично запускає функцію компонента заново, підставляючи нові значення стейту. Після цього він порівнює попередню й нову версію віртуального DOM і оновлює тільки змінені частини реального DOM.

На рисунку 3.4 зображені функції завантаження даних, які заповнюють стани з даних бекенду.

```
33    const loadMeta = useCallback(async () => {
34      try {
35        const params = {
36          channels: Array.from(selectedChannelIds).join(','),
37          dateFilter,
38          durationFilter,
39        };
40        const [detRes, chRes] = await Promise.all([
41          getSetDetails(setId),
42          getSetChannels(setId, params),
43        ]);
44        setSetName(detRes.data.name);
45        setChannels(chRes.data);
46      } catch (err) {
47        console.error(err);
48      }
49    }, [setId, dateFilter, durationFilter, selectedChannelIds]);
50
51    const loadVideos = useCallback(async () => {
52      try {
53        const params = {
54          page,
55          PAGE_SIZE,
56          channels: Array.from(selectedChannelIds).join(','),
57          dateFilter,
58          durationFilter,
59          sortField,
60          sortOrder,
61        };
62        const res = await getSetVideos(setId, params);
63        setVideos(res.data.data);
64        setTotal(res.data.total);
65      } catch (err) {
66        console.error(err);
67      } finally {
68        setLoading(false);
69      }
70    }, [
71      setId,
72      page,
73      selectedChannelIds,
74      dateFilter,
75      durationFilter,
76      sortField,
77      sortOrder,
78    ]);
79
80    useEffect(() => {
81      loadMeta();
82    }, [loadMeta]);
83
84    useEffect(() => {
85      loadVideos();
86    }, [loadVideos]);
87
88    useEffect(() => {
89      if (!isChannelsInitialized && channels.length > 0) {
90        setSelectedChannelIds(new Set(channels.map(ch => ch.yt_channel_id)));
91        setisChannelsInitialized(true);
92      }
93    }, [channels, isChannelsInitialized]);
```

Рисунок 3.4 – Функції завантаження даних

Функції, обгорнені в useCallback, щоб не створювати новий екземпляр без потреби. За допомогою Promise.all одночасно запитуємо інформацію про набор та список каналів. Функція loadVideos отримує список відео передаючи номер сторінки, розмір сторінки, вибрані канали, фільтри дати й тривалості, а також порядок сортування. UseEffect викликає loadMeta та loadVideos одразу після монтування компонента та щоразу, коли змінюється будь-яка зі залежностей відповідних функцій. Третій useEffect відповідає за первинну ініціалізацію вибраних каналів та запускається лише один раз, щойно масив каналів заповниться.

Отримання повідомлень про помилки по сокету з бекенду, що зображено на рисунку 3.5 також реалізоване через useEffect.

```
useEffect(() => {
  const socket = io('http://localhost:5000');
  socket.on('yt-error', ({ stage, message }) => {
    alert(`Помилка на етапі ${stage}: ${message}`);
  });
  return () => {
    socket.disconnect();
  };
}, []);
```

Рисунок 3.5 – Отримання повідомлень з бекенду

На рисунку 3.6 зображені обробники фільтрів та сортування. HandleFilterChange приймає об’єкт з новими значеннями фільтрів, оновлює стани та встановлює першу сторінку, щоб відобразити результати з початку. HandleSort встановлює за яким стовбцем йде сортування та за яким порядком.

```
97   const handleFilterChange = ({ dateFilter, durationFilter }) => {
98     setDateFilter(dateFilter);
99     setDurationFilter(durationFilter);
100    setPage(1);
101  };
102
103  const handleSort = field => {
104    if (sortField === field) {
105      setSortOrder(prev => (prev === 'asc' ? 'desc' : 'asc'));
106    } else {
107      setSortField(field);
108      setSortOrder('desc');
109    }
110    setPage(1);
111  };
```

Рисунок 3.6 – Обробники фільтрів та сортування

На рисунку 3.7 зображені UI-блоки сторінки з таблицею. Поки йде завантаження даних, відразу повертається компонент <Loading />. В цей час основний блок не рендерується.

Коли завантаження завершено на сторінці відображається другий блок. В його складі компоненти реалізація яких знаходиться в окремих файлах.

SidePanel отримує дані про набір і канали та колбеки для зміни вибору й оновлення. FilteredTable – поточні фільтри й обробник їх зміни. VideoTable – масив відео зі станом сортування. Pagination – номер сторінки, загальну кількість та колбек зміни сторінки.

```
113   if (loading) return <Loading />;
114
115   return (
116     <div className="app">
117       <SidePanel
118         setId={setId}
119         setName={setName}
120         channels={channels}
121         videos={videos}
122         selectedChannelIds={selectedChannelIds}
123         onSelectionChange={ids => {
124           setSelectedChannelIds(ids);
125           setPage(1);
126         }}
127         onDataChange={loadMeta}
128       />
129       <div className="data-table">
130         <FilteredTable
131           dateFilter={dateFilter}
132           durationFilter={durationFilter}
133           onFilterChange={handleFilterChange}
134         />
135         <VideoTable
136           videos={videos}
137           sortField={sortField}
138           sortOrder={sortOrder}
139           onSort={handleSort}
140         />
141         <Pagination
142           page={page}
143           pageCount={pageCount}
144           onPageChange={setPage}
145         />
146       </div>
147     </div>
148   );
149 }
```

Рисунок 3.7 – UI-блоки

Для оптимізації коду константи та деякі функції було вирішено виділити окремі файли. Це дозволяє централізовано зберігати й керувати статичними даними (наприклад, опціями фільтрації дат і тривалості). На рисунках 3.8 – 3.9 зображені файли з константами та додатковими функціями.

```
export const dateMap = { '24h': 1, '7d': 7, '28d': 28, '90d': 90, '365d': 365 };

export const DATE_OPTIONS = [
    { value: 'all', label: 'Усі' },
    { value: '24h', label: 'до 24 годин' },
    { value: '7d', label: 'до 7 днів' },
    { value: '28d', label: 'до 28 днів' },
    { value: '90d', label: 'до 90 днів' },
    { value: '365d', label: 'до року' },
];

export const DURATION_OPTIONS = [
    { value: 'all', label: 'Всі' },
    { value: 'short', label: 'до 3 хвилин' },
    { value: 'long', label: 'більше 3 хвилин' },
];

export const PAGE_SIZE = 25;
```

Рисунок 3.8 – Файл constants.js

```
export function fmt(n) { return n.toString().replace(/\B(?=(\d{3})+(?!\d))/g, ','); }

export function fmtc(n) {
  if (n >= 1e6) {
    const mln = n / 1e6;
    const str = mln % 1 === 0 ? `${mln}` : mln.toFixed(1);
    return `${str} млн. підп.`;
  } else if (n >= 1e3) {
    const tis = n / 1e3;
    const str = tis % 1 === 0 ? `${tis}` : tis.toFixed(1);
    return `${str} тис. підп.`;
  } else {
    return `${n} підп.`;
  }
}

export function parsePublishDate(dateString) {
    if (!dateString) return 'N/A';
    const published = new Date(dateString);
    const diffMs = Date.now() - published.getTime();
    const days = Math.floor(diffMs / 86400000);
    const hours = Math.floor(diffMs / 3600000);
    if (days < 1) return hours < 1 ? 'менше години тому' : `${hours} год тому`;
    if (days < 30) return `${days} дн тому`;
    const months = Math.floor(days / 30);
    if (months < 12) return `${months} міс тому`;
    return `${Math.floor(days / 365)} р тому`;
}
```

Рисунок 3.9 – Файл utils.js

Усього маємо чотири унікальні поля введення, а саме: логін, пароль, назву каналу, назву набору. Для кожного з них необхідно розробити валідацію, щоб уникнути зайвих помилок. На рисунках 3.10 – 3.12 зображені відповідний код валідації.

```
15      const newErrors = {};
16      if (!login.trim()) {
17        newErrors.login = 'Логін не може бути порожнім';
18      }
19
20      if (!password) {
21        newErrors.password = 'Пароль не може бути порожнім';
22      } else if (password.length < 6) {
23        newErrors.password = 'Пароль повинен містити не менше 6 символів';
24      }
25
26      if (Object.keys(newErrors).length) {
27        alert(Object.values(newErrors).join('\n'));
28        return;
29      }
```

Рисунок 3.10 – Валідація логіну та паролю

```
39      const newErrors = {};
40      if (!channelHandle.trim()) {
41        newErrors.channelHandle = 'Назва набору не може бути порожньою';
42      }
43
44      if (Object.keys(newErrors).length) {
45        alert(Object.values(newErrors).join('\n'));
46        return;
47      }
```

Рисунок 3.11 – Валідація назви каналу

```
25      const newErrors = {};
26      if (!newSetName.trim()) {
27        newErrors.newSetName = 'Назва набору не може бути порожньою';
28      }
29
30      if (Object.keys(newErrors).length) {
31        alert(Object.values(newErrors).join('\n'));
32        return;
33      }
34
35      try {
36        await createSet(newSetName.trim());
37        setNewSetName('');
38        fetchSets();
39      } catch (error) {
40        console.error('Помилка створення набору:', error);
41      }
42    };
```

Рисунок 3.12 – Валідація назви набору

Також до всіх input був доданий атрибут maxLength, щоб обмежити символи за довжиною.

Реалізація запитів до API. Файл api.js слугує єдиною точкою взаємодії фронтенду з REST-API бекенду. Він налаштовує клієнтську бібліотеку axios з базовим URL, автоматично додає заголовок авторизації з JWT із localStorage до кожного запиту і експортує набір функцій, що відповідають за авторизацію користувача та операції над наборами і їхніми каналами й відео. На рисунку 3.13 зображено відповідний файл.

```
1   import axios from 'axios';
2
3   const API_URL = 'http://localhost:5000/api';
4
5   const apiClient = axios.create({
6     baseURL: API_URL,
7   });
8
9   apiClient.interceptors.request.use(config => {
10    const token = localStorage.getItem('token');
11    if (token) {
12      config.headers.Authorization = `Bearer ${token}`;
13    }
14    return config;
15  });
16
17  export const registerUser = (login, password) =>
18    apiClient.post('/auth/register', { login, password });
19
20  export const loginUser = (login, password) =>
21    apiClient.post('/auth/login', { login, password });
22
23  export const getSets = () =>
24    apiClient.get('/sets');
25
26  export const createSet = name =>
27    apiClient.post('/sets', { name });
28
29  export const removeSet = setId =>
30    apiClient.delete(`/sets/${setId}`);
31
32  export const getSetDetails = setId =>
33    apiClient.get(`/sets/${setId}`);
34
35  export const addChannelToSet = (setId, { handle }) =>
36    apiClient.post(`/sets/${setId}/channels`, { handle });
37
38  export const removeChannelFromSet = (setId, channelId) =>
39    apiClient.delete(`/sets/${setId}/channels/${channelId}`);
40
41  export const getSetVideos = (setId, params = {}) =>
42    apiClient.get(`/sets/${setId}/videos`, { params });
43
44  export const getSetChannels = (setId, params = {}) =>
45    apiClient.get(`/sets/${setId}/channels`, { params });
```

Рисунок 3.13 – Файл api.js

Налаштовуємо клієнт. Інтерцептор запитів перевіряє наявність токена в localStorage і, якщо він є, додає його до заголовка Authorization: Bearer <token>.

Авторизація користувача:

– registerUser(login, password) – POST /auth/register, надсилає login та password;
– loginUser(login, password) – POST /auth/login, надсилає login та password й повертає JWT.

Робота з наборами:

– getSets() – GET /sets, повертає всі набори користувача;
– getSetDetails(setId) – GET /sets/:setId, повертає назву набору по id;
– createSet(name) – POST /sets, створює новий набір із вказаною назвою;
– removeSet(setId) – DELETE /sets/:setId, видаляє набір по id.

Робота з каналами в наборі:

– getSetChannels(setId) – GET /sets/:setId/channels, повертає список каналів у наборі;
– addChannelToSet(setId, { handle }) – POST /sets/:setId/channels, додає канал за його handle;
– removeChannelFromSet(setId, channelId) – DELETE /sets/:setId/channels/:channelId, видаляє канал з набору.

Отримання відео з набору:

– getSetVideos(setId) – GET /sets/:setId/videos, отримує відео всіх каналів у зазначеному наборі.

Завдяки такій організації, будь-який компонент вебдодатку може імпортувати потрібну функцію з api.js й виконувати запити до серверу, не дублюючи налаштування URL та заголовків авторизації.

Остаточна організація фронтенду зображена на рисунку 3.14. Фронтенд поділено за відповідальністю: у components/ – повторно використовувані UI-блоки (таблиці, панель, пагінація тощо), у pages/ – сторінки для роутів, у services/api.js – виклики до бекенду, у utils/ – константи й допоміжні функції, всі стилі лежать у styles/. Це дає чітке розділення логіки й полегшує підтримку.

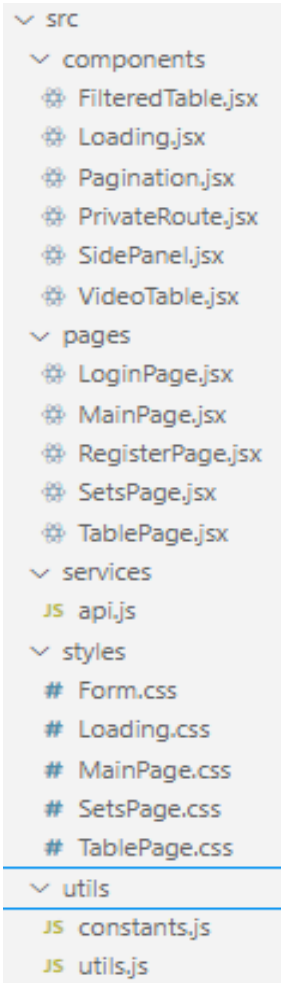


Рисунок 3.14 – Структура папки /frontend

3.4 Розробка бекенду

Розробка бекенду була складена з таких ключових компонентів:

– ініціалізація Express-серверу;
– реалізація аутентифікації;
– реалізація CRUD-операцій;
– робота з YouTube Data API v3;
– підключення до бази даних;
– налаштування сокету для виведення помилок;
– налаштування регулярного оновлення статистики відео.

Виходячи з цих пунктів було побудовано структуру бекенду, що зображена на рисунку 3.15.

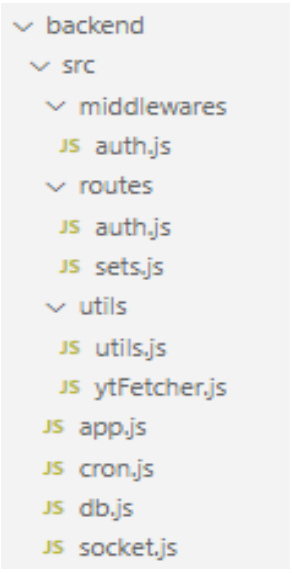


Рисунок 3.15 – Структура папки /backend

App.js є головною точкою входу до бекенд-сервера, реалізованого на базі Express. Відповідний файл зображено на рисунку 3.16.

```
const { startCronJobs } = require('./cron');
const express = require('express');
const cors = require('cors');
const http = require('http');
const { initSocket } = require('./socket');
const app = express();

const authRoutes = require('./routes/auth');
const setsRoutes = require('./routes/sets');
const { authMiddleware } = require('./middlewares/auth');

app.use(cors());
app.use(express.json());

app.use('/api/auth', authRoutes);

app.use('/api/sets', authMiddleware, setsRoutes);

startCronJobs();

const server = http.createServer(app);
initSocket(server);

server.listen(5000, () => {
  console.log('Сервер працює на порту 5000');
});
```

Рисунок 3.16 – Файл app.js

У ньому визначаються маршрутизатори та middleware: імпортуються модулі авторизації та роботи з наборами каналів з перевіркою авторизації

authMiddleware. App.use встановлює глобальні middleware для дозволу запитів із різних джерел та автоматичного парсингу JSON-тіла запитів. Далі реєструється публічний маршрут /api/auth, доступ до якого не вимагає авторизації, і захищений маршрут /api/sets, доступ до якого вимагає авторизації. Також ініціалізуємо сокет та запускаємо періодичні завдання, що виконує оновлення статистики. Сервер прослуховує HTTP-запити на порті 5000.

Наступним етапом є підключення до бази даних. Код, зображений на рисунку 3.17, відповідає за налаштування та експорт підключення до MySQL.

```
1   const mysql = require('mysql2/promise');
2
3   const pool = mysql.createPool({
4     host: process.env.DB_HOST || 'db',
5     user: process.env.DB_USER || 'user',
6     password: process.env.DB_PASSWORD || 'password',
7     database: process.env.DB_NAME || 'name'
8   });
9
10  module.exports = pool;
```

Рисунок 3.17 – Файл db.js

Створюємо пул з’єднань через mysql.createPool, в налаштуваннях якого вказуємо параметри підключення. Опції беруться з файлу docker-compose.yml. Створений pool експортується, що дозволяє використовувати його в інших модулях додатку.

Авторизація буде розроблена з використанням бібліотеки jsonwebtoken. У файлі, зображеному на рисунку 3.18, визначається middleware-функція authMiddleware, яка бере заголовок Authorization із запиту. Якщо заголовок відсутній або не відповідає формату Bearer <token>, відправляється повідомленням про помилку.

У разі наявності токену виконується його перевірка через jwt.verify. Якщо під час верифікації виникла помилка, наприклад, час життя токену вичерпано, клієнту повертається відповідний текст помилки. Так само було розроблено реєстрацію та авторизацію.

```
exports.authMiddleware = (req, res, next) => {
  const authHeader = req.headers.authorization;
  if (!authHeader) {
    return res.status(401).json({ message: 'No token provided' });
  }

  const [scheme, token] = authHeader.split(' ');
  if (scheme !== 'Bearer' || !token) {
    return res.status(401).json({ message: 'Invalid token format' });
  }

  jwt.verify(token, JWT_SECRET, (err, decoded) => {
    if (err) {
      const msg = err.name === 'TokenExpiredError'
        ? 'Token has expired'
        : 'Invalid token';
      return res.status(401).json({ message: msg });
    }
    req.user = { id: decoded.id, login: decoded.login };
    next();
  });
};
```

Рисунок 3.18 – Файл auth.js

Розберемо ендпоінт реєстрації, що зображена на рисунку 3.19. Відбувається звернення до бази даних через об’єкт pool та виконується запит вибірки всіх записів у таблиці users. Якщо такий запис уже існує, код повертає відповідь з повідомленням про те, що користувач з таким логіном вже зареєстрований.

```
router.post('/register', async (req, res) => {
  const { login, password } = req.body;
  if (!login || !password) {
    return res.status(400).json({ message: 'Login and password required' });
  }

  const [rows] = await pool.query('SELECT * FROM users WHERE login = ?', [login]);
  if (rows.length > 0) {
    return res.status(400).json({ message: 'User already exists' });
  }

  const hashedPassword = await bcrypt.hash(password, 10);

  await pool.query(
    'INSERT INTO users (login, password_hash) VALUES (?, ?)',
    [login, hashedPassword]
  );

  res.status(201).json({ message: 'User registered successfully' });
});
```

Рисунок 3.19 – HTTP-ендпоінт /register

Якщо логін вільний, система переходить до захисту пароля. За допомогою бібліотеки bcrypt пароль користувача хешується з використанням

10 ітерацій солі. Після успішного утворення хешу виконується вставка нового запису в таблицю users, де зберігаються логін та згенерований хеш пароля.

Найважливішою частиною бекенду є збір даних. Для цих цілей у нашому випадку відмінно підходить Youtube Data Api v3. Це RESTful-інтерфейс, що дозволяє взаємодіяти з сервісом YouTube. З його допомогою є можливість виконувати запити для отримання інформації про канали, плейлисти, відео та коментарі.

Кожен запит до API будується навколо ресурсів. Ресурси – це сутності YouTube (search, videos, channels, playlists). Відповіді надходять у форматі JSON. Для роботи потрібен API-ключ. API-ключ – для публічних запитів. Система квот розподіляє ліміти запитів, щоб запобігти зловживанням, тобто за кожен виклик списують певну кількість квот. Всього на один обліковий запис виділено 10000 квот. Для розширення лімітів необхідно подавати заявку від імені компанії. Це потрібно буде врахувати у разі масштабування вебдодатку.

Важливим фактором є грамотне використання квот. Наприклад, отримання списку відео через сутність search видасть максимум 500 відео з одного каналу, при цьому один запит може містити тільки 50 відео, а його вартість становить 100 квот за 50 відео. Це нераціональне використання, оскільки є сутність playlists. Кожен канал має плейлист із завантаженнями, що дозволяє отримувати дані витрачаючи лише 1 квоту за 50 відео. Також у цієї сутності немає обмеження на загальну кількість відео. Разом за допомогою безкоштовних квот ми зможемо обробляти близько 18000 відео. Цього вистачить для реалізації вебпрограми з урахуванням раціонального використання.

Розглянемо функцію оновлення даних. Функція updateChannelsVideosView, що зображена на рисунку 3.20, працює в кілька кроків. Спочатку перевіряю зміни кількості відео у каналах. Викликається videoCountChannelsCheck, яка порівнює збережену в базі video_count з актуальними даними з YouTube. Якщо кількість відео змінилася, то

оновлюються всі записи відео для тих каналів, де виявлено розбіжності. Далі отримуємо об’єкт з інформацією про канали разом із списками відео з бази даних.

```
async function updateChannelsVideosView() {
  try {
    await videoCountChannelsCheck();
    const channels = await getChannels();
    await Promise.all(channels.map(updateVideosView));
  } catch (err) {
    console.error(`Помилка при оновленні переглядів каналів: ${err.message}`);
    try {
      getIo().emit('yt-error', {
        stage: 'оновлення переглядів усіх каналів',
        message: err.response?.data?.error?.message || `невідома помилка`
      });
    } catch (e) {
      console.error('Socket emit error:', e.message);
    }
  }
}
```

Рисунок 3.20 – Функція updateChannelsVideosView

Наступний етап – це оновлення статистики відео одночасно для кожного каналу в режимі Promise. В функції updateVideosView дані розбиваються на чанки по 50 відео, по кожному чанку запитуються перегляди та тривалість. Нові значення зберігаються в таблиці video_stats і оновлюються в стовпчику duration у таблиці videos. Відео, що не повернулися з API, видаляються з бази, бо власник каналу їх видалив.

Завершальним етапом є обробка помилок, через Socket.IO відправляється повідомлення на фронтенд із описом стадії, на якій вона відбулася, та текстом помилки.

Також розглянемо функцію додавання каналу, щоб детальніше розглянути структуру API-запитів. Перший блок коду, що зображено на рисунку 3.21, створює інстанс клієнта axios, налаштований на роботу з YouTube Data API v3. За допомогою baseURL він автоматично додає базову частину URL для всіх запитів, а в params передається ключ API. Це дозволяє в подальшому в коді викликати методи типу youtube.get без повторного зазначення повного шляху.

```
const youtube = axios.create({
  baseURL: 'https://www.googleapis.com/youtube/v3',
  params: { key: apiKey },
});
```

Рисунок 3.21 – Окремий екземпляр axios для YouTube API

На рисунках 3.22 – 3.25 зображено уривки функції addChannelDataByHandle з запитом до Youtube Data Api v3. Спочатку виконується GET-запит до кінцевої точки /channels, з параметрами ‘snippet,statistics,contentDetails’ і заданим handle, щоб отримати базову інформацію про канал, його статистику та ID плейлиста з усіма завантаженнями.

Якщо канал за вказаним handle не знайдено, відразу повертається відповідь із статусом 404. Інакше з об’єкта відповіді витягуються потрібні дані. Якщо запису немає, створюється новий, після чого для доданого каналу викликаються допоміжні функції завантаження даних про всі відео та оновлення переглядів, тривалості відео. Уривок коду зображено на рисунку 3.22.

```
const { data } = await youtube.get('/channels', {
  params: {
    part: 'snippet,statistics,contentDetails',
    forHandle: handle,
  },
});

const item = data.items?.[0];
if (!item) {
  return res.status(404).json({ message: 'Channel not found'});
}

const channelId = item.id;
const { title } = item.snippet;
const logoUrl = item.snippet.thumbnails.default.url;
const subscriberCount = item.statistics.subscriberCount || 0;
const videoCount = item.statistics.videoCount;
const uploadsId = item.contentDetails.relatedPlaylists.uploads;
```

Рисунок 3.22 – Уривок функції addChannelDataByHandle (1)

```
const [result] = await pool.query(
  'INSERT INTO channels (yt_channel_id, title, logo_url, subscriber_count, video_count, uploads_id) VALUES (?, ?, ?, ?, ?, ?)',
  [channelId, title, logoUrl, subscriberCount, videoCount, uploadsId]
);
channelDbId = result.insertId;

await updateVideosData({ id: channelDbId, uploads_id: uploadsId });
const channel = await getChannels(channelDbId);
await updateVideosView(channel);
```

Рисунок 3.23 – Уривок функції addChannelDataByHandle (2)

Після успішного додавання каналу функція встановлює зв’язок між каналом і набором у таблиці set_channels за допомогою INSERT IGNORE, щоб уникнути дублюючих зв’язків. На рисунку 3.24 зображено відповідний код.

```
await pool.query(
  'INSERT IGNORE INTO set_channels (set_id, channel_id) VALUES (?, ?)',
  [setId, channelDbId]
);
```

Рисунок 3.24 – Уривок функції addChannelDataByHandle (3)

Якщо під час будь-якого з цих кроків виникає помилка, вона ловиться у блоці catch. За допомогою WebSocket, генерується подія на фронтенд із деталями про помилку. На рисунку 3.25 зображено відповідний код.

```
} catch (err) {
  console.error(`Не вдалося додати канал ${handle}:`, err.message);
  try {
    getIo().emit('yt-error', {
      stage: 'додавання каналу',
      message: err.response?.data?.error?.message || `невідома помилка`
    });
  } catch (e) {
    console.error('Socket emit error:', e.message);
  }
  return res.status(500).json({ message: 'Failed to fetch channel info', error: err.message });
}
```

Рисунок 3.25 – Уривок функції addChannelDataByHandle (4)

Реалізація функції getIo розташована в окремому файлі socket.js, що зображено на рисунку 3.26.

```
const { Server } = require('socket.io');
let io;

function initSocket(server) {
  io = new Server(server, {
    cors: { origin: '*' }
  });
  io.on('connection', socket => {
    console.log('Socket підключений:', socket.id);
  });
  return io;
}

function getIo() {
  if (!io) throw new Error('Socket.IO не ініціалізований');
  return io;
}

module.exports = { initSocket, getIo };
```

Рисунок 3.26 – Файл socket.js

Цей файл реалізує просту обгортку для підключення та використання бібліотеки Socket.IO. Функція initSocket приймає HTTP-сервер як параметр та створює на його основі екземпляр Socket.IO із відкритою політикою та налаштовує обробник подій. При кожному новому підключенні клієнта в консоль виводиться повідомлення, що дозволяє відслідковувати активні з'єднання.

Наприкінці визначаємо CRUID операції для вебдодатку. CRUID – це абревіатура дій над таблицею бази даних, тобто з англійської створювати, читати, оновлювати, вставляти, видаляти. Ці операції визначено в файлі set.js, структура якого зображена на рисунку 3.27.

```
router.post('/', authMiddleware, async (req, res) => {···
});

router.get('/', authMiddleware, async (req, res) => {···
});

router.delete('/:id', authMiddleware, async (req, res) => {···
});

router.get('/:id', authMiddleware, async (req, res) => {···
});

router.get('/:id/channels', authMiddleware, async (req, res) => {···
});

router.post('/:id/channels', authMiddleware, async (req, res) => {···
});

router.delete('/:id/channels/:channelId', authMiddleware, async (req, res) => {···
});

router.get('/:id/videos', authMiddleware, async (req, res) => {···
});
```

Рисунок 3.27 – Структура файлу set.js

Операції над наборами каналів:

– POST створює новий набір з іменем, що передано в тілі запиту;
– GET повертає всі набори поточного користувача;
– DELETE /:id: видаляє зазначений набір, а також очищає зайві дані;

– GET /:id: повертає назву конкретного набору;
– GET /:id/channels: повертає всі виділені канали з набору;
– POST /:id/channels: додає канал до набору за переданим юзернеймом;
– DELETE /:id/channels/:channelId: видаляє зв’язок каналу з набором.

Періодичне оновлення реалізовано через функцію startCronJobs, що зображена на рисунку 3.28. Вона запускається у app.js. Усередині неї викликається cron, який запускає переданий асинхронний колбек щогодини.

```
function startCronJobs() {
  cron.schedule('*/60 * * * *', async () => {
    try {
      console.log('Початок оновлення даних YouTube...');
      await updateChannelsVideosView();
      console.log('Оновлення даних YouTube завершено.');
    } catch (error) {
      console.error('Помилка при оновленні даних:', error);
    }
  });
}
```

Рисунок 3.28 – Уривок з файлу cron.js

В ньому також виконується функція updateChannelsVideosView, що відповідає за збір статистики.

Для ручного тестування застосунку було виконано такі основні етапи:

– аутентифікація, а саме створення нового аккаунту, вхід із правильними та неправильними даними, перевірка повідомлень про помилки;
– керування наборами каналів, а саме створення/видалення набору, додавання каналу через handle з коректним та неіснуючим handle, повторне додавання, видалення каналу;
– робота з таблицею, а саме завантаження даних, перевірка розрахунків, фільтрування, сортування, пагінації;
– імітація з перевищення лімітів за API, невірного API-ключа.

Отже, завдяки опрацюванню функціональних сценаріїв, UI-перевіркам і тестам на обробку помилок, було виправлено всі критичні дефекти.

3.5 Висновки до розділу 3

У результаті було розроблено вебдодаток з використанням запланованого стеку технологій. Фронтенд на React забезпечує інтерфейс із гнучкою системою фільтрації, сортування та пагінації, бекенд на Node.js – надійне опрацювання запитів, авторизацію, взаємодію з MySQL та YouTube Data API.

Основні механізми взаємодії між клієнтом і сервером реалізовані через RESTful API й WebSocket-сповіщення. Базу даних спроєктовано відповідно до моделі IDEF1, а ініціалізація через Docker-контейнери гарантує розгортання середовища на більшості пристроїв. Ретельно опрацьовані валідація полів, централізовані константи й утиліти, інтерцептори для авторизації та оптимізовані запити до YouTube API забезпечують якість коду та легкість подальшого масштабування. Ручне тестування функціональних сценаріїв підтвердило стабільність роботи системи. Таким чином, вебдодаток виконав усі функціональні та нефункціональні вимоги.

ВИСНОВКИ

В межах виконання даної кваліфікаційної роботи був досліджений та описаний процес розробки вебзастосунку у всіх ключових аспектах.

Збір та аналіз вимог дозволили визначити перелік функціональних і нефункціональних характеристик вебсервісу. Моделювання діаграм забезпечило візуалізацію архітектурних рішень на ранніх етапах розробки. У результаті детального вивчення предметної області сформовано чітке уявлення про етапи створення сучасного вебсервісу для моніторингу відеоконтенту. Був створений вебсервіс для моніторингу YouTube-каналів із періодичним оновленням даних через YouTube Data API v3 завдяки проєктуванню трьохрівневої клієнт-серверної архітектури з використанням React, Node.js та MySQL. Було реалізовано аутентифікації користувачів за допомогою JWT та налаштування періодичних cron-завдань для автоматичного збору статистики. Збудований зручний інтерфейс із використанням однотипних стилів. Ручне тестування функціоналу підтвердило коректність роботи системи.

Практична реалізація показала, що обраний стек технологій відповідає вимогам щодо продуктивності, масштабованості та зручності підтримки. Вебсервіс стабільно функціонує під час перезапусків, коректно обробляє помилки API-запитів та забезпечує користувацьку аутентифікацію. Отже, виконана робота підтверджує доцільність обраних підходів для створення вебсервісу моніторингу популярності відеоконтенту на платформі YouTube.

ПЕРЕЛІК ПОСИЛАНЬ

1. Web Application in subject area: Computer Science. *ScienceDirect*. URL: https://tinyurl.com/ecns8vmy (дата звернення: 19.02.2025).

2. What is REST? *Codecademy*. URL: https://www.codecademy.com/article/what-is-rest (дата звернення: 20.02.2025).

3. Cross-Origin Resource Sharing (CORS). URL: https://developer.mozilla.org/en-us/docs/web/http/guides/cors (дата звернення: 11.03.2025).

4. Web Development. What is JSON? *W3schools*. URL: https://www.w3schools.com/whatis/whatis_json.asp (дата звернення: 20.02.2025).

5. VidIQ. URL: https://vidiq.com (дата звернення: 12.03.2025).

6. 10 корисних інструментів для UI/UX-дизайнера. URL: https://goit.global/ua/articles/10-korysnykh-instrumentiv-dlia-ui-ux-dyzaynera/ (дата звернення: 19.03.2025).

7. Вступ до REST API – RESTful вебсервіси. URL: https://robotdreams.cc/uk/blog/466-vstup-do-rest-api-restful-vebservisi (дата звернення: 19.03.2025).

8. Макет вебсайту: що це таке і як його зробити. URL: https://dizz.in.ua/uk/maket-veb-sajtu-shho-cze-take-i-yak-jogo-zrobiti/ (дата звернення: 25.03.2025).

ДОДАТОК А

Інтерфейс сторінок додатку

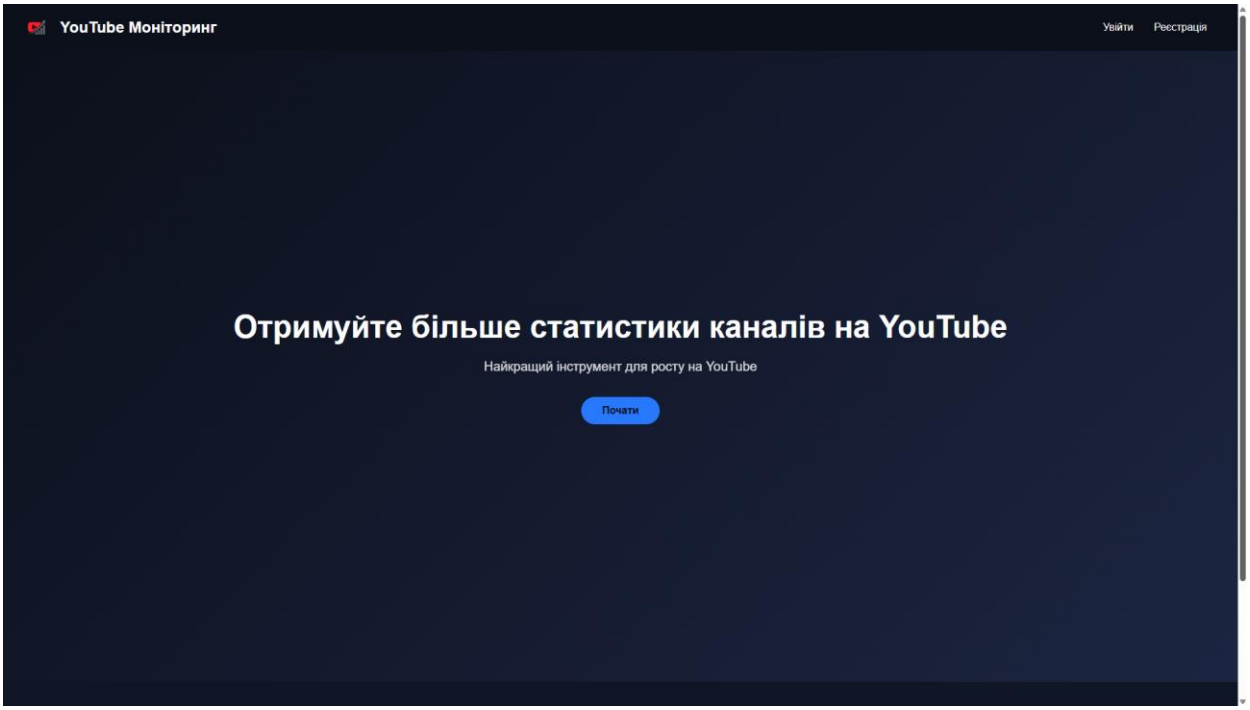


Рисунок А.1 – Головна сторінка

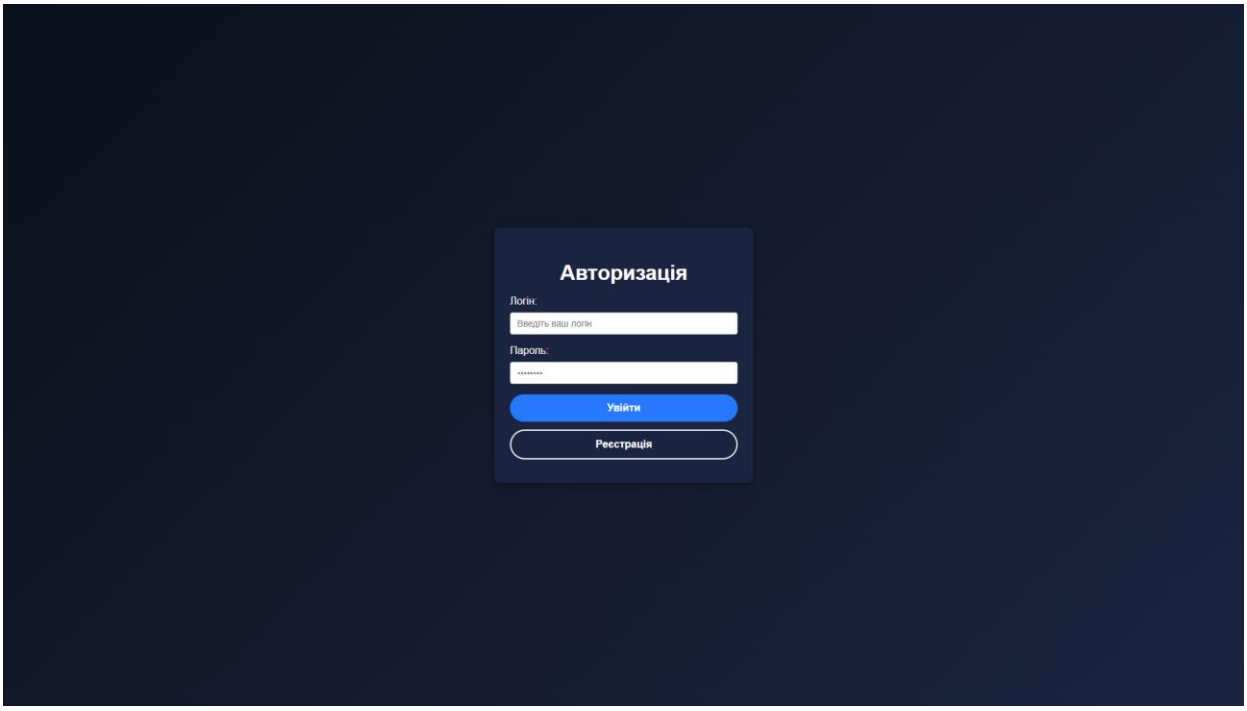


Рисунок А.2 – Сторінка авторизації

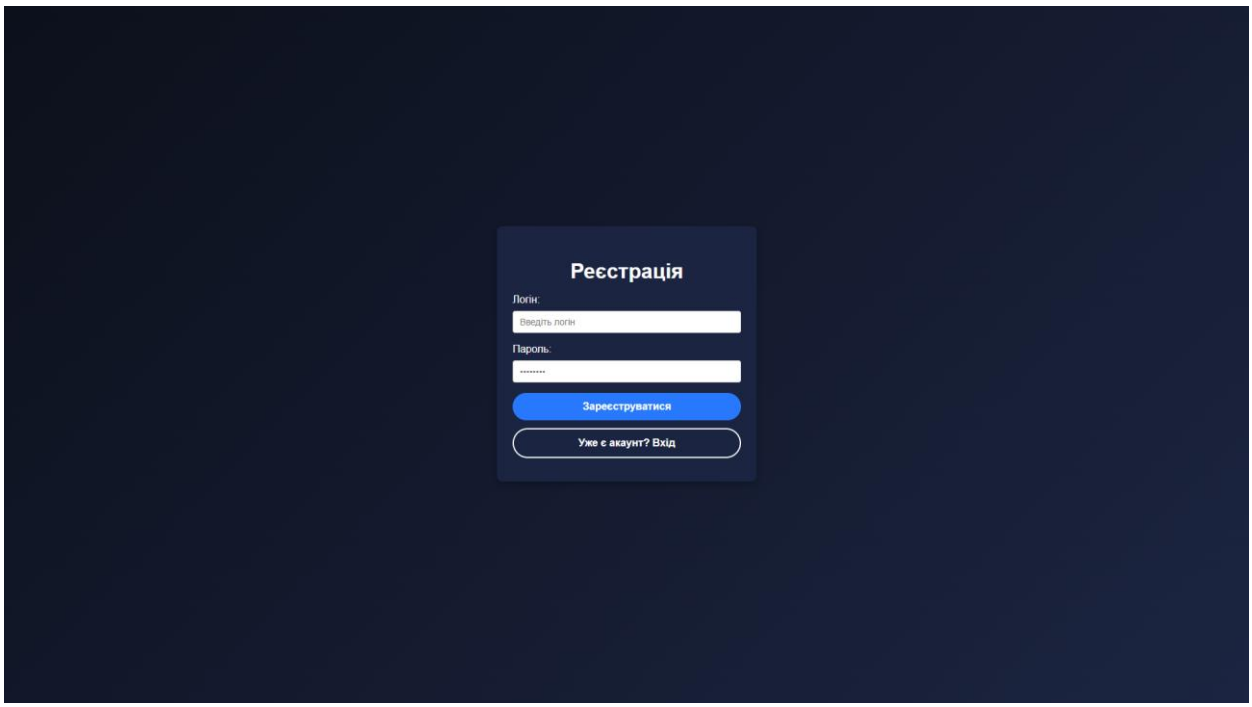


Рисунок А.3 – Сторінка реєстрації

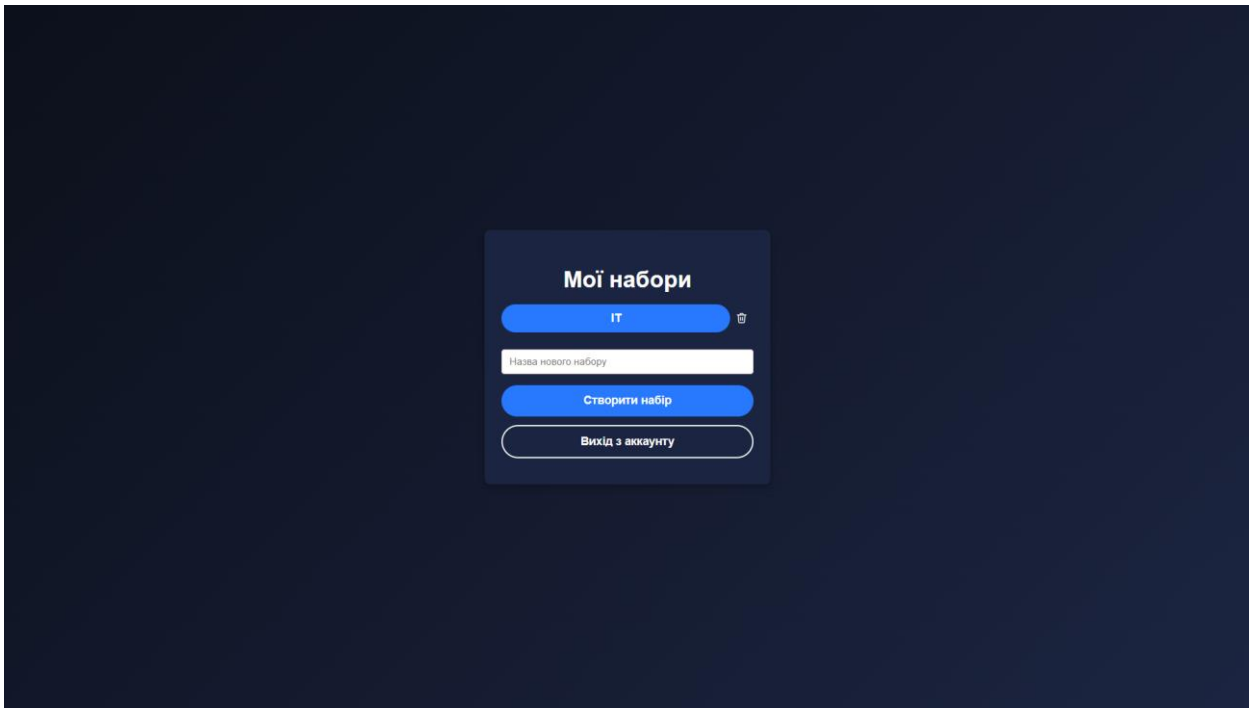


Рисунок А.4 – Сторінка зі списком наборів

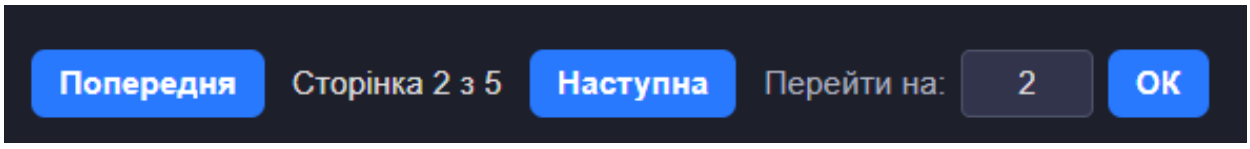


Рисунок А.5 – Елемент вибору сторінки наприкінці таблиці

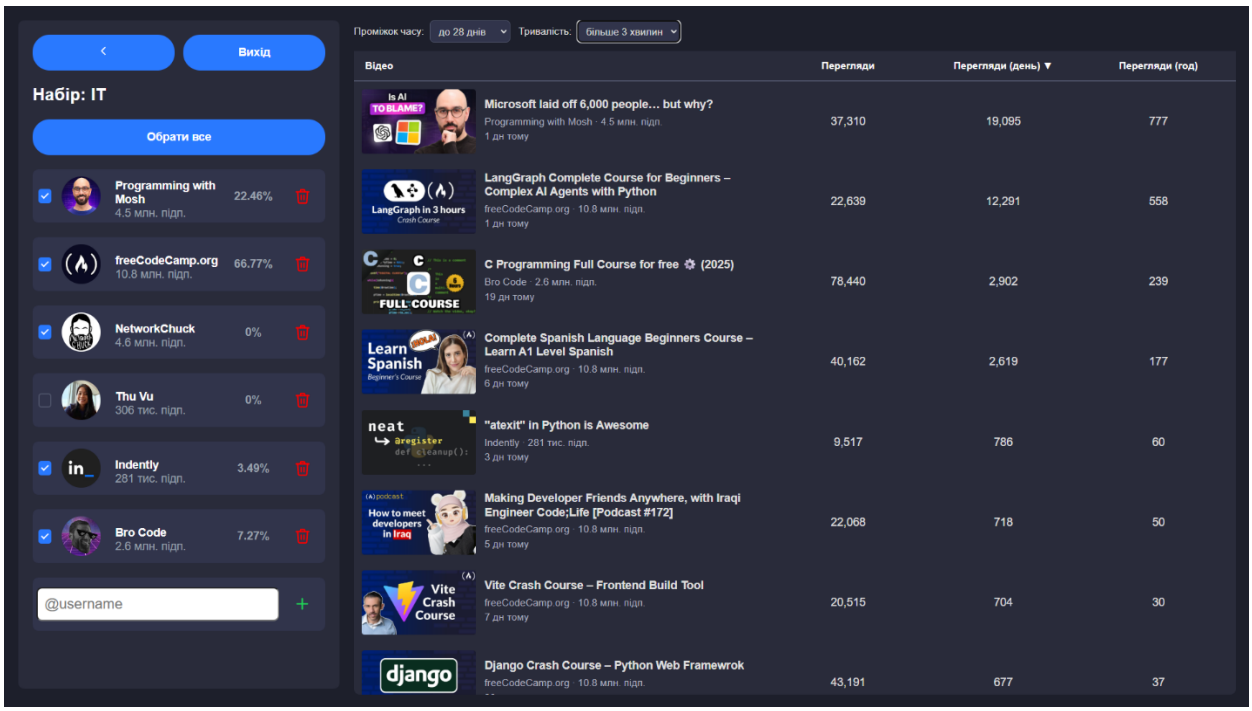


Рисунок А.6 – Сторінка з виведенням статистики

ДОДАТОК Б

Програмна реалізація

Програмна реалізація розміщена у відкритому доступі за посиланням на GitHub: https://github.com/markkrukow/ytmonitoring.