КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теоретичної кібернетики

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

РОЗРОБЛЕННЯ ПРОГРАМНОЇ СИСТЕМИ ПЕРЕГЛЯДУ НАВЧАЛЬНОГО РОЗКЛАДУ З РІЗНИМ СТУПЕНЕМ ДЕТАЛІЗАЦІЇ ВІДОБРАЖЕННЯ

Виконав студент 4-го курсу
Валерій РУЧКО _____
(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Тетяна КАРНАУХ _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теоретичної кібернетики

«____»_____2023 р., протокол № ____

Завідувач кафедри

Юрій КРАК _____

(підпис)

Київ - 2023

РЕФЕРАТ

Обсяг роботи: 41 сторінка, 13 ілюстрацій, 19 використаних джерел, 1 таблиця, 2 додатки.

Ключові слова: ВЕБЗАСТОСУНОК, КЛІЄНТ, СЕРВЕР, TYPESCRIPT, REACT, NODE.JS, NEXT.JS, POSTGRESQL, HTTP.

Об’єкт роботи: використання вебтехнологій, а саме комунікація клієнта з сервером з використанням протоколу HTTP, де сервер виконавши необхідні запити до бази даних та провівши маніпуляції над даними повертає необхідну клієнту інформацію.

Предмет роботи: вебзастосунок, який має можливості відображення, навігації та вивантаження інформації, написаний з використанням мови TypeScript з допомогою бібліотек React та TypeORM, середовища Node.js виконання JavaScript-коду.

Мета роботи: створити вебзастосунок, який дозволить проводити перегляд розкладу типового факультету великого університету з різними ступенями деталізації відображення.

Інструменти розроблення: кросплатформове середовище розробки VSCode, система контролю версій Git, мінімалістичний текстовий редактор Nvim, pgAdmin4 — система адміністрації СУБД PostgreSQL, Thunar Client — клієнт для тестування роботи API.

Результат роботи: було реалізовано вебзастосунок для перегляду розкладу факультету та під час його реалізації досліджено процеси розробки вебзастосунків.

Застосунок є спрямованим на можливість використання адміністрацією університету та студентами, та має можливості вибору огляду частини розкладу для різноманітних освітніх програм, груп тощо.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

ВЕБЗАСТОСУНОК — розподілений застосунок, який зберігається на віддаленому сервері та передає свої дані через інтернет, для відображення їх крізь інтерфейс браузера [1].

ІНТЕРФЕЙС — це набір засобів та правил, який визначає взаємодію з певними елементами системи [2].

СУБД (**система управління базами даних**) — це збірка застосунків, яка містить в собі як і сховище даних (базу даних), так і інструменти для управління та зчитування цих даних.

ФРЕЙМВОРК — набір готових інструментів для розв’язання типових задач, що полегшує процес розроблення [3].

HTTP — протокол передачі гіпертекстових документів.

HTTP-GET-запит — один з методів рядка HTTP-запиту, який позначає запит від клієнта на отримання вмісту певного ресурсу.

ВСТУП

Оцінка сучасного стану об’єкта розроблення. Станом на сьогодні все більше компаній розміщують копії своїх застосунків для настільних комп’ютерів у хмарі, через що необхідність в застосунках, які потрібно завантажувати, поступово зменшується, а в користувачів з’являється більше вибору. Зараз існує багато всесвітньо відомих компаній, що проводять подібне перенесення власних застосунків, до прикладу: «Adobe Inc., Figma Inc., Autodesk Inc., GoodNotes Limited» [4, 01:30]. Саме тому актуальність веббраузера, як середовища для використання застосунків, що використовуються в різноманітних сферах ужитку (дизайн, офісна сфера, програмування тощо) на професійній основі, та які все менше обмежені в можливостях, порівняно з настільними аналогами, поступово зростає.

Одним з найпопулярніших типів вебзастосунків, які використовуються повсякденно, є системи для перегляду, редагування, та складання розкладів. Перевагами їх реалізації саме у вебсередовищі є доступність на різних пристроях та операційних системах, а також портативність — можливість мати доступ до інструментів застосунку в будь-якому місці, а не лише там, де стоїть персональний комп’ютер чи ноутбук.

Актуальність роботи та підстави для її виконання. На цей час застосунки перегляду розкладів реалізовані або як вбудовані сервіси у вже наявні системи, або ж вони є перевантаженими в плані функціонала, через що їх використання може бути якщо не надто складним, то недостатньо зручним. Створення окремого застосунку перегляду розкладу, що орієнтується на розв'язання однієї простішої задачі, дозволить зробити його більш інтуїтивним у використанні. Також такий застосунок буде більш доступним через розміщення у вебсередовищі.

Мета й завдання роботи. Спираючись на вищезазначений опис тенденцій створення застосунків, мета даної дипломної роботи — створити вебзастосунок, який дозволить переглядати з різними ступенями деталізації розклад типового

факультету великого університету та при цьому матиме достатньо простий інтерфейс для його використання.

Для того, щоб досягнути поставленої мети, необхідно:

- дослідити присутні інструменти, технології та засоби для розроблення вебзастосунку;
- навчитися користуватися обраними засобами реалізації вебзастосунку;
- визначити структурні елементи вебзастосунку, які необхідно реалізувати;
- спроєктувати структуру клієнтської та серверної частин застосунку та реалізувати її;
- спроєктувати архітектуру бази даних та наповнити її тестовими даними;
- забезпечити взаємодію клієнта та сервера.

Об’єктом розроблення є візуалізація обраної користувачем частини розкладу на факультеті університету з можливістю фільтрування відображуваної інформації за певними категоріями.

Методи та засоби розроблення. Ітеративне та інкрементне розроблення виконувалось мовою програмування TypeScript з використанням середовища виконання JavaScript коду Node.js, бібліотек React, TypeORM та фреймворку Next.js для розробки клієнтської частини застосунку.

Можливі сфери застосування. Застосунок можна застосовувати для перегляду розкладів в університеті, а також розширити для використання в таких місцях, як: школи, дитячі садки, центри розвитку тощо.

1 ОГЛЯД МОЖЛИВИХ ПІДХОДІВ ДО ПОБУДОВИ ЗАСТОСУНКУ ДЛЯ ВІДОБРАЖЕННЯ РОЗКЛАДУ

1.1 Особливості навчального розкладу факультету, що реалізує кілька освітніх програм бакалаврського рівня

Огляд розкладу для декількох освітніх програм, які реалізуються на факультеті, на цей момент проводиться з допомогою файлу формату .xlsx та програмного забезпечення Excel від компанії Microsoft.

Це спричинено тим, що саме в цьому настільному застосунку відбувається редагування та наповнення змістом самого розкладу. Вибір застосунку очевидний, адже компанія Microsoft є загальновідомим виробником якісного програмного забезпечення, яке використовується щодня як і в повсякденному житті, так і зарекомендувало себе як де-факто стандартом для використання в професійному середовищі для редагування текстів, створення таблиць, розв’язання проблем менеджменту тощо.

Однак, така універсальність та зрілість застосунку одночасно є і його недоліком — він занадто перевантажений функціоналом, та що важливіше, застарілим кодом, який погіршує показники швидкодії застосунку, а відповідно і зручність користування ним. Причина наявності застарілого коду в застосунку такого масштабу проста — Microsoft докладає значних зусиль задля підтримки редагування та огляду таблиць з Excel більш старих форматів, щоб зберегти зворотну сумісність з попередніми версіями.

Що це означає для перегляду розкладу на факультеті, в якому присутні кілька освітніх програм? Якщо розглянути поточний вигляд файлу розкладу (див. рис. 1.1), можна помітити, проблему з об’ємом його вмісту — в одному файлі вміщено розклади для усіх чотирьох курсів потоку, а в таблицях, що належать до розкладів кожного з курсів присутні дані для всіх освітніх програм та відповідно їх

груп, що існують на факультеті. Враховуючи, що розклад у файлі визначений для всіх робочих днів тижня, у сумі маємо двадцять таблиць присутніх у файлі розкладу.

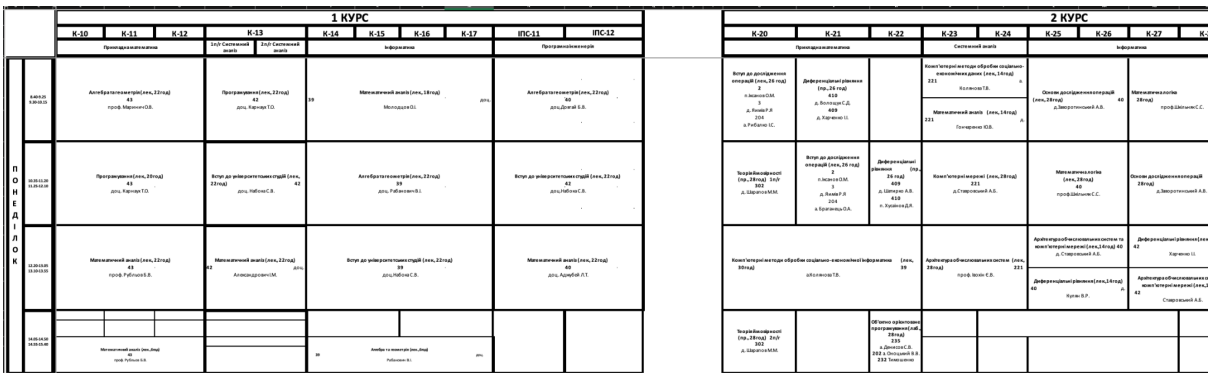

Рисунок 1.1 — Фрагмент поточного файлу розкладу факультету

Для редагування та наповнення вмісту розкладу, доцільність такої величезної структури цілком виправдана, адже маючи всю наявну інформацію в одному місці, можна ефективніше помічати та усувати недоліки розкладу.

Проте для перегляду розкладу, така громіздка структура наявна у файлі створює лише незручності, серед яких:

- зменшення швидкодії роботи застосунку, через що процес навігації розкладом та його перегляд стає повільним та незручним;
- через велику кількість даних та форматування, присутніх у файлі розкладу, застосунок потребує більше часу для обробки інформації, що спричинює його повільне відкривання, а часом створює непосильну задачу для студентів чи викладачів — навіть з використанням легкої, за рівнем навантаження на систему, програми LibreOffice, оригінальний xlsx файл розкладу не вдається відкрити навіть на достатньо потужному комп’ютері через перенасичення внутрішніми об’єктами;
- з точки зору використання файлу розкладу студентами та викладачами, можна зазначити, що найчастіше їх цікавить частина розкладу, яка відноситься лише до їхніх кафедр або ж взагалі груп, через що решту наявної інформації у файлі розкладу можна вважати не лише надлишковою для подібних користувачів, а й такою, що заважає пошуку необхідних даних.

Зважаючи на вищеописані проблеми, від застосунку, що може бути створений задля їх вирішення, очікується можливість фрагментації розкладу на менші структурні частини, наявність певного фільтра для відображення інформації, наявність зручної навігації, а також помітне покращення швидкодії роботи застосунку.

1.2 Вебзастосунок як форма розповсюдження програмного забезпечення

Для розв'язання проблеми відображення розкладу з різними ступенями деталізації стояв вибір створення застосунку одного з наступних двох типів:

- нативний застосунок, створений для окремої платформи, наприклад операційних систем Windows, MacOS, Linux;
- вебзастосунок, клієнт якого перебуває в середовищі браузера.

Хоча переваги написання нативного застосунку доволі привабливі — локальне збереження даних, відсутність необхідності мати стабільний доступ до інтернету та підлаштування під особливості конкретної платформи [5], у такого підходу до створення застосунків існує великий недолік — відсутність доступності до функціонала нативного застосунку користувачам платформ, відмінних від тієї, для якої застосунок написаний.

1.2.1 Розв'язання проблеми доступності

Основною перевагою вебзастосунків є те, що вони виконуються в ізольованому середовищі — веббраузері. Веббраузер своєю чергою має можливості інтерпретації динамічної мови програмування JavaScript, а також її компіляції на льоту («Just-in-time compilation» [6]), тобто під час виконання

програми.

Можливість інтерпретації дозволяє мові, а відповідно й застосункам, написаним з її допомогою, запускатися на будь-якій платформі яку підтримує веббраузер й мати при цьому універсальну поведінку, реалізація якої є незалежною від платформи.

Своєю чергою, компіляція коду на льоту збільшує швидкість його виконання, що відповідно покращує досвід користувачів під час використання застосунків.

З цього можна зазначити, що написання застосунку, доступ до якого відбувається безпосередньо через вебплатформу, робить його доступнішим для більшої категорії користувачів, а сам застосунок матиме прийнятну швидкість виконання для більшості типових завдань.

Окрім того, додатковими перевагами вебзастосунків є:

- розподіленість — серверна та клієнтська частина розташовуються в окремих площина, що дозволяє краще розділяти обов’язки щодо обробки інформації, її відображення, зберігання, а отже й контролювати використання ресурсів на стороні користувача;
- також через те, що вебзастосунок розташовується безпосередньо у мережі інтернет, впровадження у застосунок нового функціоналу та його відображення відбувається лише зі сторони розробника без будь-яких додаткових дій зі сторони користувача, через що останньому не потрібно нічого вивантажувати, щоб користуватися новими інструментами сервісу, яким він користується;
- вебзастосунки не потрібно завантажувати, що дозволяє користувачу економити місце на диску для більш важливої інформації.

1.2.2 Підхід до відображення розкладу

Для відображення розкладу, було вирішено створити вебзастосунок, який має наступні характеристики:

- чітке розділення відображення інформації за структурними частинами, а саме курсами та освітніми програмами;
- можливість при перегляді інформації звужувати кількість виведеної на екран інформації з допомогою фільтрів по *типу тижня* (парний, непарний), *типу предмету* (лекція, лабораторна, практика), *прізвищу викладача, назві групи та номеру авдиторії*;
- фільтрування можливе одночасно за декількома наявними категоріями;
- процес навігації застосунком має чітко інформувати користувача про те, яку частину навчального розкладу йому буде показано;
- застосунок має бути швидким та не навантажувати систему користувача;
- застосунок має бути сприятливим до розширення в майбутньому.

Також, застосунок повинен мати можливість адаптації елементів інтерфейсу до екранів невеликого розміру та бути доступним у браузерах з різними рушіями.

2 ЗАСОБИ ДЛЯ СТВОРЕННЯ ВЕБЗАСТОСУНКУ

2.1 Огляд технологій TypeScript, Node.js, React

TypeScript — це «високорівнева, строго-типізована мова програмування, створена компанією Microsoft як надмножина над мовою JavaScript» [7].

Загальною метою створення цієї мови було надання JavaScript нових можливостей з допомогою додаткового синтаксису, які допомогли б мові:

- підвищити читабельність коду для розробників, з допомогою надання можливості вибіркового позначення типів змінних, параметрів функцій;
- досягти кращих можливостей у підтримці проєктів, шляхом покращення зручності використання концептів парадигми об’єктно-орієнтованого програмування;
- спростити інтеграцію мови з інтегрованим середовищем програмування, що дозволяє останньому краще вловлювати та сигналізувати про помилки ще на етапі розробки, а також надавати більш точні підказки та документацію;
- з допомогою статичної типізації зменшити кількість помилок, пов’язаних з присвоєнням та передачею змінних, параметрів, класів неправильного типу;
- спростити підключення модулів у проєкті.

Найголовнішою особливістю TypeScript є повна зворотна сумісність з JavaScript, через що будь-який JavaScript код вважається чинним TypeScript кодом, що суттєво полегшує перенесення вже зрілих проєктів на нову мову програмування. Окрім цього, дана особливість дозволяє TypeScript бути доступною у всіх середовищах, де запускається JavaScript, що надає мові перевагу у вигляді вже напрацьованої протягом багатьох років екосистеми з купою корисних бібліотек.

Node.js — це «асинхронне, подіє-орієнтоване середовище виконання JavaScript коду» [8]. Саме це середовище виконання дозволяє виконувати мову

JavaScript поза межами браузера, для розробки серверів, інтерфейсів командного рядка, скриптів тощо.

Це середовище обране для виконання завдання створення серверної частини вебзастосунку та комунікації з базою даних через простоту у використанні, уніфікацію використовуваної мови програмування як для серверної, так і для клієнтської частин, що значно пришвидшить розробку, а також через ефективну роботу з виконання запитів до бази даних та обробки запитів клієнта, що дозволяє застосунку швидко знаходити та віддавати необхідні дані.

Також через свою неблокуючу природу (майже усі операції вводу та виводу проходять поза межами головного потоку, на якому виконується програма), середовище може обробляти велику кількість одночасних запитів до сервера, що значно покращує його пропускну спроможність, відповідно більше користувачів зможуть комфортно користуватися застосунком.

React — це JavaScript бібліотека, створена компанією Meta, що має на меті полегшення створення інтерфейсів вебзастосунків та вирішення проблем пов’язаних з оновленням частини інформації на вебсторінці [9].

Бібліотека буде корисною у вебзастосунку в першу чергу через підвищення модульності коду клієнтської частини застосунку, що дозволить повторне використання компонентів інтерфейсу. Також, активніше використовуючи можливість оновлення частин сторінки, можна буде проводити обробку даних для відображення безпосередньо зі сторони клієнта більш ефективно, що дасть змогу робити менше запитів до бази даних, і відповідно прискорити швидкість відгуку застосунку.

2.2 Next.js та server-side rendering

Next.js — це фреймворк з відкритим вхідним кодом, призначений для веброзроблення, що базується на використанні бібліотеки для створення

інтерактивних вебзастосунків React [10].

Як і більшість фреймворків для веброзроблення, він надає велику кількість інструментів для вирішення типових проблем, що виникають під час розроблення вебзастосунків, як то навігація сторінками, створення запитів для отримання даних чи автоматичне налаштування конфігурації проєкту.

Проте основними його перевагами є дві речі:

1. Даний фреймворк має власну чітко визначену файлову структуру та правила, щодо розміщення тих чи інших частин вебзастосунку в проєкті. Це покращує орієнтацію в компонентах проєкту для окремого розробника, а також робить розроблення застосунку зручнішою для роботи в команді.
2. Будь-який проєкт, розроблений та запущений на Next.js, отримує можливість використання вбудованих у фреймворк інструментів для оптимізації швидкодії, які в іншому випадку доводилося би реалізувати самостійно, або ж шукати відповідні для цього пакунки.

До таких методик оптимізації відносяться: автоматичне розділення коду — включення при відображенні сторінки лише того коду, який вона потребує для роботи; попереднє завантаження сторінки (page prefetching), на яку можна перейти з поточної; відмальовування елементів інтерфейсу користувача на стороні сервера (server-side rendering).

Server-side rendering — техніка відмальовування всього вмісту HTML сторінки на серверній частині застосунку. Після цього, відмальована сторінка надсилається до клієнта, де на його стороні у сторінку вбудовується необхідний для роботи сторінки JavaScript код. Це дозволяє сторінкам з великою кількістю статичного змісту та невеликою кількістю інтерактивних елементів завантажуватися набагато швидше ніж відбувається процес відмальовування ідентичної сторінки на стороні клієнта.

Враховуючи особливості вебзастосунку для відображення розкладу, де інтерактивними елементами будуть лише фільтри пошуку та елементи навігації, але аж ніяк не частини розкладу, дана методика є гарним вибором для відображення великої кількості інформації, що присутня в університетському розкладі.

2.3 Огляд протоколу HTTP

HTTP (HyperText Transfer Protocol) — це «протокол передачі даних прикладного рівня, призначений для розподілених, колаборативних, гіпермедійних інформаційних систем» [11].

Цей протокол вживається ледве не найчастіше серед усіх існуючих для комунікації клієнта з сервером та передачі повідомлень між ними. Зазвичай це відбувається за моделлю «запит — відповідь», де запит відбувається зі сторони клієнта, після чого починається процес очікування відповіді. Також особливістю цього протоколу є те, що власний стан ним не відстежується, тобто при кожному наступному запиті результати усіх попередніх не бувають відомими.

Структура рядка запиту HTTP-протоколу складається з наступних п’яти частин:

- версія протоколу HTTP (на цю мить, найчастіше вживаною є версія HTTP/1.1, випущена 1999-го року);
- уніфікований ідентифікатор ресурсу (URI) [12] — простіше кажучи це рядок, що вказує на розташування сервера або клієнта, щоб в подальшому можна було встановити між ними комунікацію;
- заголовки запиту — набір пар ключ — значення, що є обов’язковими для кожного запиту та відповіді на нього, адже вони містять всі необхідні деталі для комунікації клієнта з сервером, такі як інформація про те, який браузер використовується клієнтом; хто є власником ресурсу; оригінальний ідентифікатор ресурсу тощо;
- тип методу запиту до сервера: GET, POST, PUT, PATCH, DELETE;
- тіло запиту — є необов’язковим, та зазвичай використовується для передачі даних, що були введені клієнтом чи згенеровані сервером.

Нижче можна побачити приклад заголовка запиту, що був зроблений в створюваному застосунку перегляду розкладів (див. рис. 2.1).

```
host: 'localhost:3000',
connection: 'keep-alive',
'sec-ch-ua': '"Google Chrome";v="113", "Chromium";v="113", "Not-A.Brand";v=
'sec-ch-ua-mobile': '?0',
'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/
'sec-ch-ua-platform': '"macOS"',
accept: '*/*',
origin: 'http://localhost:8080',
'sec-fetch-site': 'same-site',
'sec-fetch-mode': 'cors',
'sec-fetch-dest': 'empty',
referer: 'http://localhost:8080/',
'accept-encoding': 'gzip, deflate, br',
'accept-language': 'uk-UA,uk;q=0.9,en-GB;q=0.8,en;q=0.7,en-US;q=0.6,ru;q=0.
'if-none-match': 'W/"4a58-i+6mJ6TOMR5s8iCUbuBQmGCWVEg"'
```

Рисунок 2.1 — HTTP-запит, зроблений з клієнтської частини застосунку

Відповідь сервера, своєю чергою, має лише три частини: заголовки відповіді, тіло відповіді, та статусний код, який служить індикацією того, наскільки успішним чи невдалим був запит від клієнта до сервера.

Коди статусу запиту у свою чергу ділять на категорії — всі коди, що існують в межах 100 – 199 є інформаційними; коди в межах 200 – 299 індикують про успішну відповідь на запит; 300 – 399 є найменш уживаними, й свідчать про те, що користувача буде перенаправлено на іншу сторінку на сайті; 400 – 499 свідчать про помилку на стороні користувача; 500 – 599 кажуть, що запит був неуспішно оброблений сервером [13].

Нижче можна побачити приклад заголовка запиту, що був зроблений в створюваному застосунку перегляду розкладів (див. рис. 2.2).

```
▼NextResponse {Symbol(internal response): {…}, type: 'default', url: '', redirected: false, status: 200, …}
  ▶Symbol(internal response): {cookies: ResponseCookies, url: undefined}
   body: (...)
   bodyUsed: false
  ▶headers: Headers {}
   ok: true
   redirected: false
   status: 200
   statusText: ""
   type: "default"
   url: ""
   cookies: (...)
```

Рисунок 2.2 — Об’єкт-відповідь сервера з індикатором успішної відповіді на запит

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 План виконання проєкту

Для реалізації вебзастосунку, потрібно було виконати такі завдання:

- означити структуру застосунку;
- визначити структуру бази даних та реалізувати її схему з допомогою сутностей;
- установити між базою даних та сервером зв’язок з допомогою програмно — визначених сутностей;
- реалізувати та запустити HTTP-сервер обробки запиту з методом GET, що буде взаємодіяти з базою даних та клієнтом;
- втілити на сервері операції необхідні для витягнення з бази даних потрібних даних під час запитів, та передачі їх клієнту;
- визначити схему навігації вебзастосунком зі сторони клієнта;
- втілити визначену схему навігації;
- створити компоненти інтерфейсу користувача з можливістю повторного використання;
- навчитися робити відповідний запит до бази даних в залежності від того, яку частину інформації нам необхідно отримати;
- створити елементи для фільтрації, які контролюватимуть відображення тієї чи іншої частини інформації на сторінці;
- визначити вимоги до апаратного та програмного забезпечення для використання вебзастосунку;

Розглянемо окремі структурні елементи вебзастосунку та як виглядає їх реалізація.

3.2 Структура проєкту

Проєкт має структуру, розділену на декілька частин: клієнтська частина вебзастосунку, серверна частина, база даних.

Розглянемо схему проєкту, зображену нижче (рис. 3.1).

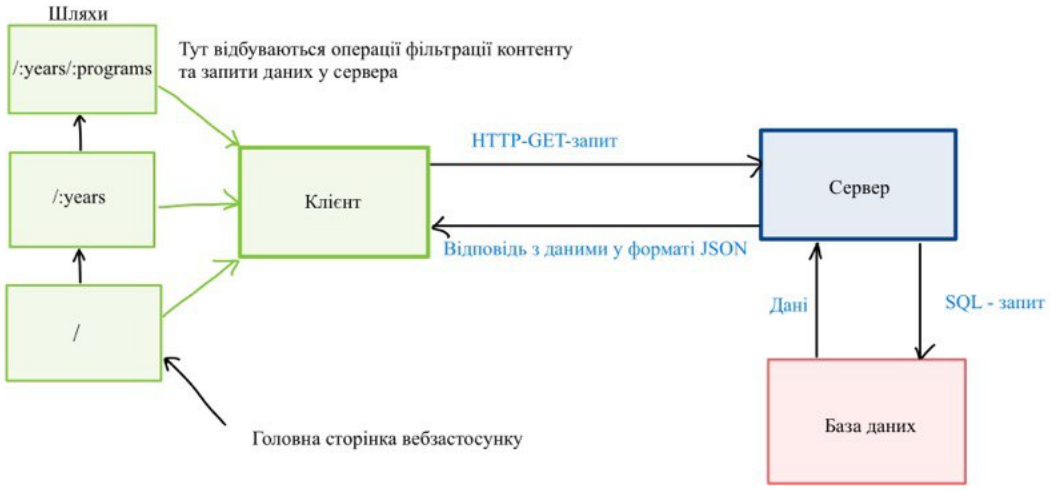

Рисунок 3.1 — Схематичне зображення структури вебзастосунку

Зліва можемо побачити, що клієнтська частина має три рівні переходів між сторінками з відповідними адресними частинами:

1. Сторінка за адресою /, тобто та, яку користувач бачить першою, маючи в адресному рядку лише назву *хоста*, або ж власника ресурсу, є вхідною в застосунок, та не робить жодних запитів до сервера. На ній користувач з допомогою елементів, розміщених в центрі сторінки, обирає курс, для якого хоче побачити відповідну частину розкладу (див. підрозділ А.1).
2. Наступна сторінка */:years* має динамічну адресу (вона позначається з допомогою двокрапки перед іменем адреси), яка називається параметром шляху, й тут визначається за вибором користувача на попередній сторінці. На цій сторінці користувач обирає, розклад якої освітньої програми обраного курсу він хоче переглянути (див. підрозділ А.2).

3. Остання сторінка має додатковий параметр шляху *:programs,* де відбувається відображення частини розкладу для обраних курсу та освітньої програми. Також, саме на цій сторінці можна відфільтрувати отриману інформацію за певними категоріями (див. підрозділ А.3).

Саме на останній сторінці за двома, отриманими від користувача, параметрами шляху й відбувається GET — запит до сервера для отримання потрібної частини розкладу.

Після надсилання запиту на сервер, він надсилає SQL — запит до бази даних, що базується на отриманих від клієнта параметрах, чекає на результат, та надсилає його кодом — індикатором статусу обробки запиту 200 назад до клієнта.

З точки зору користувача, застосунок не обтяжує його тягарем вибору через перевантаженість інтерфейсу, адже на кожному кроці навігації вебзастосунком, кількість доступних варіантів дій обмежена.

На головній сторінці застосунку вибір полягає лише у вирішенні того, частину розкладу для якого курсу користувач хоче побачити.

На другій сторінці застосунку, користувач може або обрати релевантну для нього освітню програму для перегляду її розкладу, або ж повернутися на головну сторінку застосунку, задля зміни результату свого першого рішення.

На останній сторінці застосунку, користувач може або проглянути розклад як він є, без жодних накладених фільтрів; або ж застосувати один та більше фільтрів для перегляду тієї інформації, яка йому необхідна; або ж, після завершення огляду необхідної інформації чи розуміння, що потрібно переглянути іншу частину розкладу, повернутися на головну сторінку застосунку.

3.2.1 Серверна частина

Тепер розглянемо структуру та код реалізації серверної частини.

Спершу, нам необхідно розв'язати задачу під’єднання до СУБД, в якій будуть

зберігатися дані нашого розкладу.

Як основну СУБД було обрано PostgreSQL — реляційну СУБД з відкритим вхідним кодом, написану мовою C, а для комунікації сервера з нею буде використовуватися бібліотека для створення об’єктно-реляційних відображень TypeORM [14] (об’єктно-реляційне відображення — техніка пов’язування сутностей бази даних з концептуальними частинами об’єктно-орієнтованих мов програмування [15]).

Для початку, ініціалізуємо об’єкт, який відповідатиме за те, до якої бази даних нам потрібно доєднатися, який матиме дані щодо авторизації в ній та перелік сутностей, що мають бути створені в СУБД, у разі якщо їх не існує.

```
import { DataSource } from "typeorm";
import { Teacher } from "./entity/Teacher";

export const AppDataSource = new DataSource({
  type: "postgres",
  host: "localhost",
  port: 5432,
  username: "xxxxxxxxx",
  password: "xxxxxxxxxx",
  database: "schedule_viewer",
  synchronize: true,
  logging: false,
  entities: [Teacher, Subjects, Schedule_item, Groups],
  migrations: [],
  subscribers: [],
});
```

Спочатку імпортуємо в файл об’єкт для ініціалізації джерела даних, тобто того місця, куди сервер буде робити запити.

На другому рядку також можемо побачити імпорт сутності, яка буде відповідати таблиці переліку вчителів в базі даних, і за якою ми будемо до цієї

таблиці мати доступ. Аналогічним чином імпортуються й решта сутностей: Subjects, Groups та Schedule_item.

Після того, як всі необхідні пакунки у файлі присутні, з допомогою об’єкта ініціалізації ми визначаємо необхідні дані про нашу базу даних та її наявні таблиці.

Тепер, у вхідному файлі сервера нашого вебзастосунку проводимо ініціалізацію з урахуванням визначених даних.

```
AppDataSource.initialize()
  .then(async () => {
    console.log("Datasource connected to PostgreSQL");
  })
  .catch((error) => console.log(error));
```

Та визначаємо функцію, яка буде витягати дані з бази даних в залежності від параметрів шляху, що надсилаються клієнтом під час запиту до сервера:

```
async function getData(yearU: number, spec: string) {
  const repo = AppDataSource.getRepository(Schedule_item);
  const res = await repo
    .createQueryBuilder("schedule_item")
    .leftJoinAndSelect("schedule_item.group", "groups")
    .where("groups.specializationName = :specializationName", {
      specializationName: spec,
    })
    .andWhere("groups.year = :year", {
      year: yearU,
    })
    .leftJoinAndSelect("schedule_item.teacher", "teacher")
    .leftJoinAndSelect("schedule_item.subjects", "subjects")
    .getMany();

  return res;
}
```

Функція приймає на вхід два параметри: рік (тобто курс), для якого буде витягуватися частина розкладу, та назву освітньої програми.

Далі проводиться ініціалізація репозиторію для певної сутності, щоб можна було робити до бази даних запити, обмежені лише контекстом (полям та даними з таблиць, на які вказують зовнішні ключі даної) цієї сутності.

Опісля ініціалізації репозиторію, виконуємо пошук всіх записів в таблиці, які відповідають курсу та спеціалізації, зазначеним у параметрах функції, та повертаємо змінну, що містить результат.

Нижче можемо бачити код, який виконується для HTTP-запиту з методом GET для адресного рядка з двома параметрами шляху.

```
app.get("/:years/:programs", async function(req: Request, res: Response){
  res.setHeader("Content-Type", "application/json");
  let compRes: object[] = [];

  if (req.params.programs === "comp-science") {
    compRes = await getData(
      parseInt(req.params.years.slice(0, 1)),
      SpecName.COMPSCIENCE
    );
  } else if ……

res.status(200);
res.json(compRes);
});
```

Перевіряємо які параметри шляху у нас є у запиті, та викликаємо функцію getData(), після чого сигналізуємо про успішне виконання запиту від клієнта та надсилаємо дані до нього у форматі JSON.

І запускаємо сервер, який буде під час роботи на визначеному порті приймати усі запити:

```
app.listen(port, () => {
  console.log(`App is running on port: ${port}`);
});
```

Після запуску сервера, бачимо, що підключення до бази даних проведено успішно, а сам сервер запущено (див. рис. 3.2).

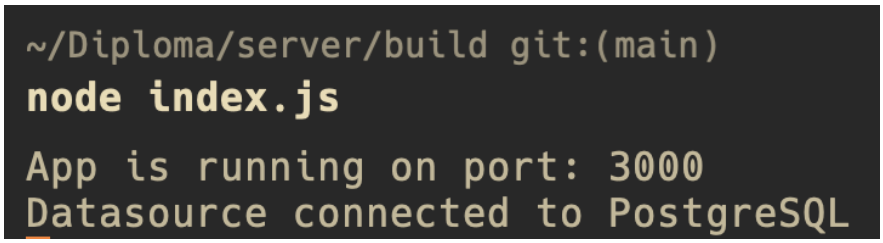

Рисунок 3.2 — Повідомлення після запуск сервера застосунку

3.2.2 База даних

При побудові застосунку, створено схему база даних, яка не була б перенавантаженою зайвими полями, та яка має максимально релевантну інформацію до тієї, що відображається на вебсторінці (див. рис. 3.3).

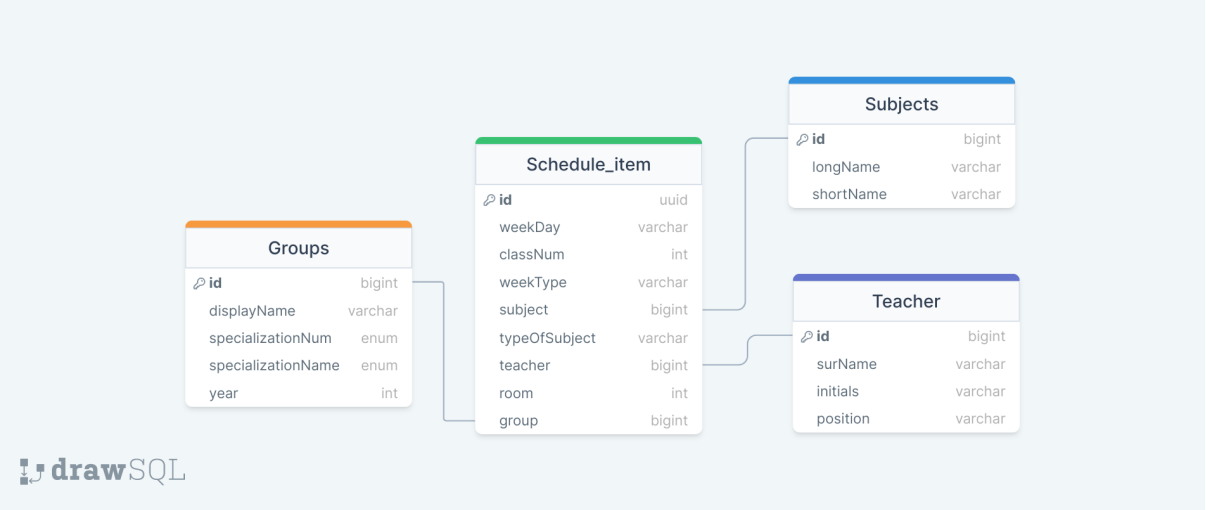

Рисунок 3.3 — Схема бази даних

Центральною таблицею у базі даних є *Schedule_item* — за своєю суттю, один запис у ній відображає одну клітинку навчального розкладу. Також у ній присутні зовнішні ключі до таблиць - складових Teacher, Groups та Subjects, які зберігають інформацію про наявних викладачів, групи та наявні предмети відповідно.

Дивлячись на програмну реалізацію сутності на стороні сервера, що

відповідає таблиці з іменем *Schedule_item* (див. додаток Б), можна помітити, що таблиці - складові пов’язані з нею типом зв’язку Багато до Одного (Many - to - One [16]), тому що кожній клітинці розкладу, яких багато, може відповідати лише один з предметів, вчителів та груп.

Параметр в оголошенні даного типу зв’язку *eager: true* позначає автоматичне довантаження даних з таблиць, пов’язаних з *Schedule_item.*

Після того, як усі сутності були ініціалізовані аналогічно до сутності *Schedule_item*, таблицю було програмно наповнено випадковими тестовими даними для подальшого розроблення та тестування вебзастосунку.

3.2.3 Клієнт

Для реалізації інтерфейсу користувача було створено шість функціональних елементів, короткий опис яких наведено в таблиці 3.1.

Таблиця 3.1 — Опис реалізованих елементів інтерфейсу

Компонент	**Опис**
NavBar	Містить елемент навігації для повернення на головну сторінку та може використовуватися в майбутньому для додаткових функціональних елементів застосунку.
Footer	Нижній колонтитул вебзастосунку
Card	Елемент, що містить посилання на наступні сторінки застосунку. Використовується для навігації користувачем до бажаної частини відображуваного розкладу.

Продовження таблиці 3.1

ScheduleCard	Елемент, що відображає клітинку розкладу та містить відповідну інформацію в ньому.
Dropdown	Випадаючий список з опціями вибору для фільтрування контенту за певною категорією.
Theme	Компонент, що містить візуальний стиль усього застосунку.

Розглянемо найбільш важливі з них детальніше, щоб краще зрозуміти, як саме застосунок функціонує. Такими елементами є Dropdown, Card та ScheduleCard.

Елемент Dropdown є основною складовою частинкою нашого застосунку, адже саме він реалізує функціональну частину фільтрування за категоріями. Для початку, розглянемо наступний інтерфейс:

```
interface filtVal {
 title: string;
 setFilt: any;
 options: {
   val: string;
   text: string;
 }[];
}
```

Інтерфейс filtVal визначає структуру, яку має втілювати кожен параметр властивостей, що передається елементу Dropdown на вхід.

Поле title відповідає за назву категорії присутню біля фільтра, щоб користувач знав, фільтр до якої саме категорії він застосовує.

У поле setFilt ми передаємо функцію, яка буде відповідати за відстежування вибору користувачем значення фільтру, застосовного в даній категорії. Самі ж можливі варіанти значень для фільтрів передаються у вкладеному масиві

об’єктів options.

Відповідні варіанти для вибору формуються з вхідного масиву варіантів наступним чином:

```
{props.options.map((v) => {
          return (
            <option key={`${v.text}`} value={v.val}>
              {v.text}
            </option>
          );
        })}
```

Функція вище бере почергово кожне значення з вхідного масиву варіантів фільтрування в межах категорії, та повертає сформований за його даними елемент HTML-списку `<option></option>`, з відповідним присвоєнням значення та тексту, який відповідає цьому значенню та який бачить користувач. На зображенні нижче можемо побачити приклад сформованого елемента для фільтрування за категорією *Група* (див. рис. 3.4).

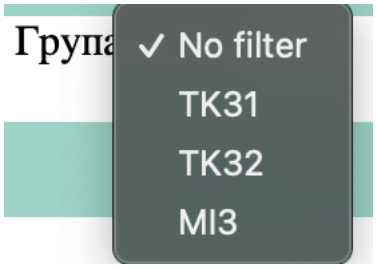

Рисунок 3.4 — Приклад згенерованого фільтру для категорії *Група*

Елемент Card є основним елементом навігації сторінками, який імплементує інтерфейс CardProps:

```
interface CardProps {
 title: string;
 path: string;
}
```

Перше поле в даному інтерфейсі відповідає за назву сторінки, на яку користувач переходить при натисканні (в контексті застосунку, назва позначає або ж номер курсу, або ж освітню програму), друге ж поле *path* позначає шлях до відповідної сторінки, який окрім всього є значенням, що додається на місце параметрів шляху *:years* або *:programs*.

Елемент ScheduleCard є основною складовою останньої сторінки, де відбувається відображення розкладу. Його параметр властивостей має втілювати інтерфейс ScheduleProps, наведений нижче, що відповідає вмісту даних, отриманих з сервера для окремої клітинки розкладу.

```
interface ScheduleProps {
 groupName: string;
 subjectName: string;
 subjType: string;
 posName: string;
 teachName: string;
 time: number;
 weekType: string;
 room: number;
}
```

Після отримання параметра властивостей, елемент розділяє необхідну інформацію на складові, для кращого відображення та можливості адаптації вмісту для екранів менших розмірів, та вбудовує їх в HTML-розмітку сторінки.

Для відображення розкладу використовується наступний вираз з використанням тернарного оператора виконання умови:

```
<div className="flex_table_wrapper">
        {weekT === "" &&
        subjT === "" &&
        teachSur === "" &&
        roomN === "" &&
        groupN === ""
          ? checkScheduleCard("monday", serverData!)
```

```
            : checkScheduleCard("monday", filteredData!)}
        </div>
```

Вираз зліва від символу ? проходить перевірку на істинність — якщо він істинний (тобто користувач застосував жодного фільтрування вмісту), відображаємо елементи всього розкладу за отриманими з сервера початковими даними (цим займається функція checkScheduleCard, яка повертає елементи ScheduleCard спираючись на приналежність їх до певного дня тижня та масив даних), інакше — відображаємо відфільтровану за категоріями інформацію.

Тепер оглянемо частину застосунку, що відповідає за основну його робочу складову. Для початку ми оголошуємо функцію, яка буде виконуватися щоразу, як користувач переходить на сторінку */:years/:programs,* та робитиме запит до бази даних, щоб витягнути необхідні дані за вказаними параметрами рядка.

```
useEffect(() => {
    async function getData(path: string) {
      const data = await fetch(`http://localhost:3000${path}`).then(
        (response) => {
          return response.json();
        }
      );
      setData(data);
    }
    getData(pathName);
  }, [pathName]);
```

[pathName] — позначає залежну змінну, яку функція відстежує та при зміні значення якої (переході користувача на іншу сторінку) виконує операції означені у тілі функції.

Далі з отриманих даних з сервера витягуються необхідні для елементів - фільтрів значення полів, щоб не було можливості фільтрації за полями, які не відображаються на сторінці.

Нижче наведено наповнення можливих варіантів вибору для категорії *Викладач*:

```
 const [teachSur, setTeachSur] = useState("");
  let teachSurOpt: DropdownOpt[] = [];
  if (serverData) {
    let surnNames = Array.from(
      new Set(
        serverData.map((value) => {
          return value.teacher.surName;
        })
      )
    );
    surnNames.map((value) => {
     teachSurOpt.push({ val: value, text: value });
    });
  }
```

Процес фільтрації поміщається у функцію, аналогічну за структурою до тієї, що виконує запит даних, та відбувається на стороні клієнта, адже об’єм даних недостатньо великий, щоб сприяти значному перенавантаженню використання ресурсів додатком на стороні користувача.

Нижче бачимо фільтр за категорією *Викладач:*

```
if (teachSur !== "") {
res = res?.filter((value) => {   return
value.teacher.surName.toLowerCase().match(teachSur.toLowerCase())
          ? true
          : false;
      });
    }
```

Таким чином, сторінка показує як і отриману на початку інформацію від сервера, не відфільтровану за жодними категоріями, так і результат після застосування фільтрів.

3.3 Вимоги до апаратного та програмного забезпечення

Для того, щоб застосунок коректно був запущений на комп’ютері, повинно бути встановлено:

- Node.js версії 18.16.0 або вище;
- npm версії 9.5.0 або вище;
- Next.js версії 13.4 або вище;

  Щодо апаратного забезпечення, посилаючись на вимоги для браузера Chrome [17], для *клієнта* необхідно мати:

  - процесор Intel Pentium 4 або вище;
  - щонайменше 4 ГБ оперативної пам’яті (стільки повинно вистачати для самого браузера та відкритої вкладки застосунку, який використовує до 100 МБ пам’яті, як зазначено на рис. 3.4);

Серверу ж для запуску бази даних, серверу та клієнтської частини, необхідно мати хоча б процесор з чотирма ядрами та тактовою частотою вище 2.8 ГГц та не менше 8 ГБ оперативної пам’яті.

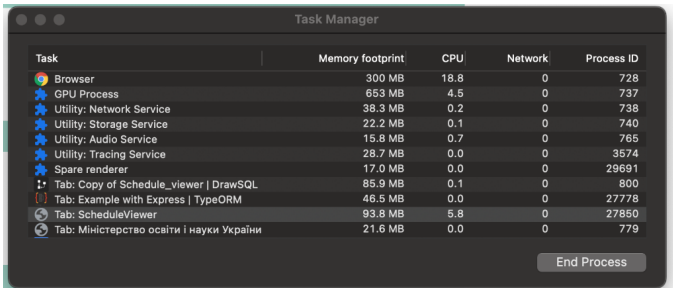

Рисунок 3.5 — Кількість споживаної пам’яті застосунком ScheduleViewer

Було проведено діагностику роботи вебзастосунку на стороні клієнта з допомогою вбудованого в браузер *Chrome* інструменту розробника під назвою *Lighthouse* [18], який проводить автоматизований аудит швидкодії сторінки та інших характеристик. Посилаючись на його показники (див. рис. 3.5), можна стверджувати, що серйозні причини для низької продуктивності застосунку відсутні, тому мінімальних вимог на стороні клієнта має бути цілком достатньо.

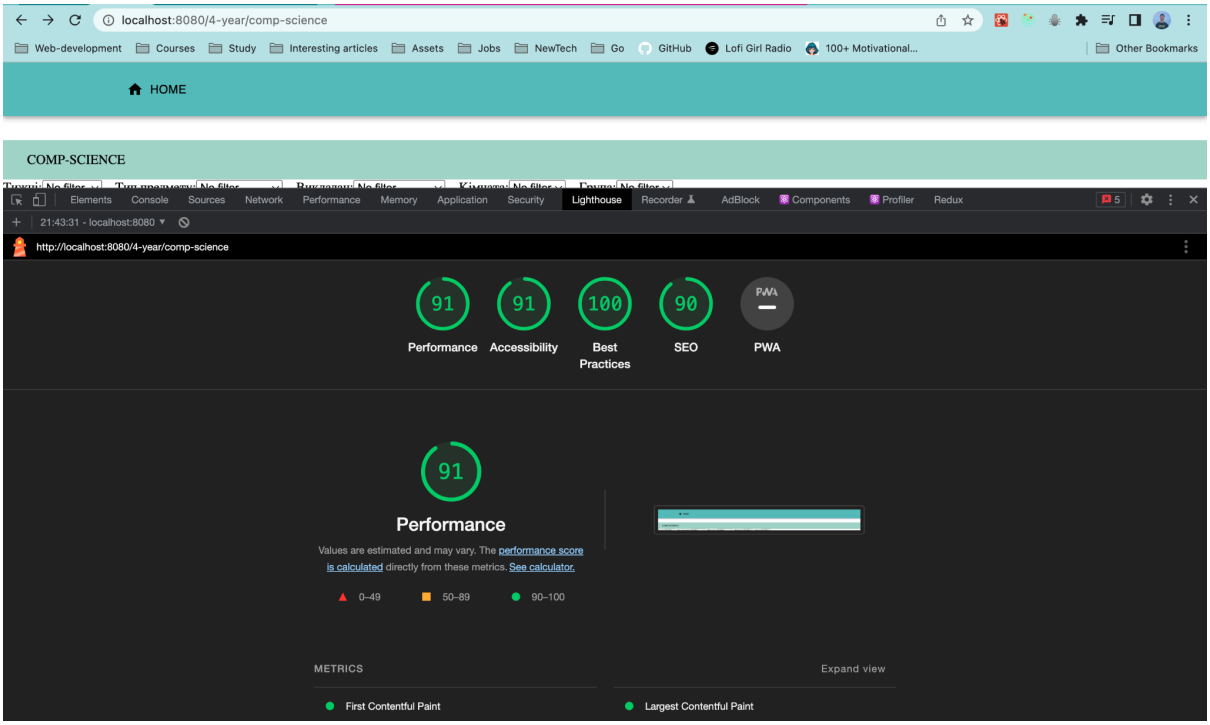

Рисунок 3.6 — Звіт роботи застосунку, згенерований Lighthouse

3.4 Інструкція з розгортання

Для того, щоб розгорнути клієнтську частину застосунку, можна використовувати декілька варіантів хостингів, проте найбільш легким способом буде використання хостингу від компанії, що створила фреймворк Next.js — Vercel, який до того ж є безплатним для некомерційного використання.

Для цього потрібно виконати наступні дії:

- створити обліковий запис на сторінці хостингу;
- імпортувати репозиторій системи контролю версій Git, в якому знаходиться проєкт до облікового запису (для цього потрібно буде здійснити вхід через обліковий запис GitHub);
- зробити налаштування опису проєкту та того, де яка складова частина у ньому знаходиться;

Щодо серверної частини застосунку, потрібно:

- вивантажити директорію з GitHub сторінки серверної частини проєкту (або ж завантажити її вручну);
- виконати команду *npm install*, для встановлення всіх необхідних залежностей, список яких присутній у файлі package.json;
- запустити встановлений сервер СУБД PostgreSQL;
- перейти у директорію серверної частини проєкту під назвою *build*, та виконати команду *node index.js*.

3.5 Оцінка зручності використання застосунком

Наостанок, проведемо оцінку того, наскільки добре реалізовано інтерфейс користувача даного вебзастосунку з точки зору зручності користування користувачем.

Найбільш популярним критерієм, що використовується для даної оцінки, є евристики Якоба Нільсена [19].

Перелічимо ці евристики, та визначимо, які з них задовольняються реалізованим інтерфейсом користувача:

1. Видимість статусу роботи системи — система має інформувати користувача про те, що відбувається в ній. *Так як* в реалізованому інтерфейсі користувача не існує прихованих процесів, а всі операції одразу відображаються на вебсторінці, користувач завжди знає про стан системи. Задовольняє.
2. Відповідність системи до реалій навколишнього світу — система мусить використовувати фрази та терміни, що є знайомими користувачу, а не є специфічними до системи. *Застосунок* використовує термінологію та образи, пов’язані лише з сутностями розкладу університету, які є відомими для потенційних користувачів — студентів та викладачів. Задовольняє.

3. Свобода користувача та надання йому свободи дій — користувач мусить мати можливість у будь-який момент вийти з небажаної сторінки без збереження змін редагування чи внесення даних. *Застосунок* не має функціоналу для редагування чи внесення даних, тому дана евристика до нього не застосовна. Проте присутній елемент, який дозволяє користувачу у будь-який момент часу повернутися на головну сторінку, якщо він того забажає, тобто сторінка дотримується описаного евристикою правила, хоч і в іншому вигляді. Не застосовна евристика.
4. Постійність та стандартизованість — елементи інтерфейсу для взаємодії та відображення мусять мати однозначні позначення без прихованих сенсів. *Застосунок* дотримується принципів виконання того, що описано на елементах інтерфейсу. Задовольняє.
5. Запобігання виникненню помилок. *Усі варіанти* для фільтрування за категоріями генеруються автоматично на основі даних, що отримуються з серверу для конкретної сторінки. Тому неіснуючих варіантів фільтрування обрати не можна. Задовольняє.
6. Розпізнавання, а не згадування — усі можливі дії на той чи інший момент мають бути відображенні користувачу, щоб він не мусив їх запам’ятовувати. *Цього досягнуто* на кожній сторінці застосунку, адже прихованих та важкодоступних функційних елементів немає (див. підрозділи А.1, А.2, А.3). Задовольняє.
7. Гнучкість та ефективність використання — користувач мусить мати можливість налаштування частих дії. Такої можливості у застосунку не реалізовано, весь інтерфейс має прямолінійну, завчасно визначену структуру. Не задовольняє.
8. Естетичний та мінімалістичний дизайн — в інтерфейсі не мусить бути нерелевантної інформації по відношенню до вмісту, що є основним. *Застосунок* відображає лише ту інформацію, яка відображає вибір користувача. Задовольняє.

9. Потрібно допомагати користувачу діагностувати та визначати помилки. *Сервер* надсилає клієнту помилку неможливості отримати дані, якщо вони відсутні у базі даних (код статусу 400). Задовольняє.
10. Має бути присутня інструкція з користування застосунком. Такою документацією є дана кваліфікаційна робота, яка описує як шляхи користування застосунком, так і кроки до його розгортання. Задовольняє.

Як бачимо, застосунок не задовольняє тільки одну з десяти евристик Нільсена (при цьому одна до нього не застосовна, а тому можемо вважати, що порушень нема), тому реалізований інтерфейс користувача можна вважати прийнятним до використання.

Наостанок, було проведено тестування доступності вебзастосунку у різних браузерах та на екранах меншого розміру. Як бачимо на рис. 3.7, застосунок працює ідентично у найпопулярніших браузерах на цей час (Microsoft Edge, Chrome, Firefox), а його елементи адаптуються до дисплеїв малого розміру.

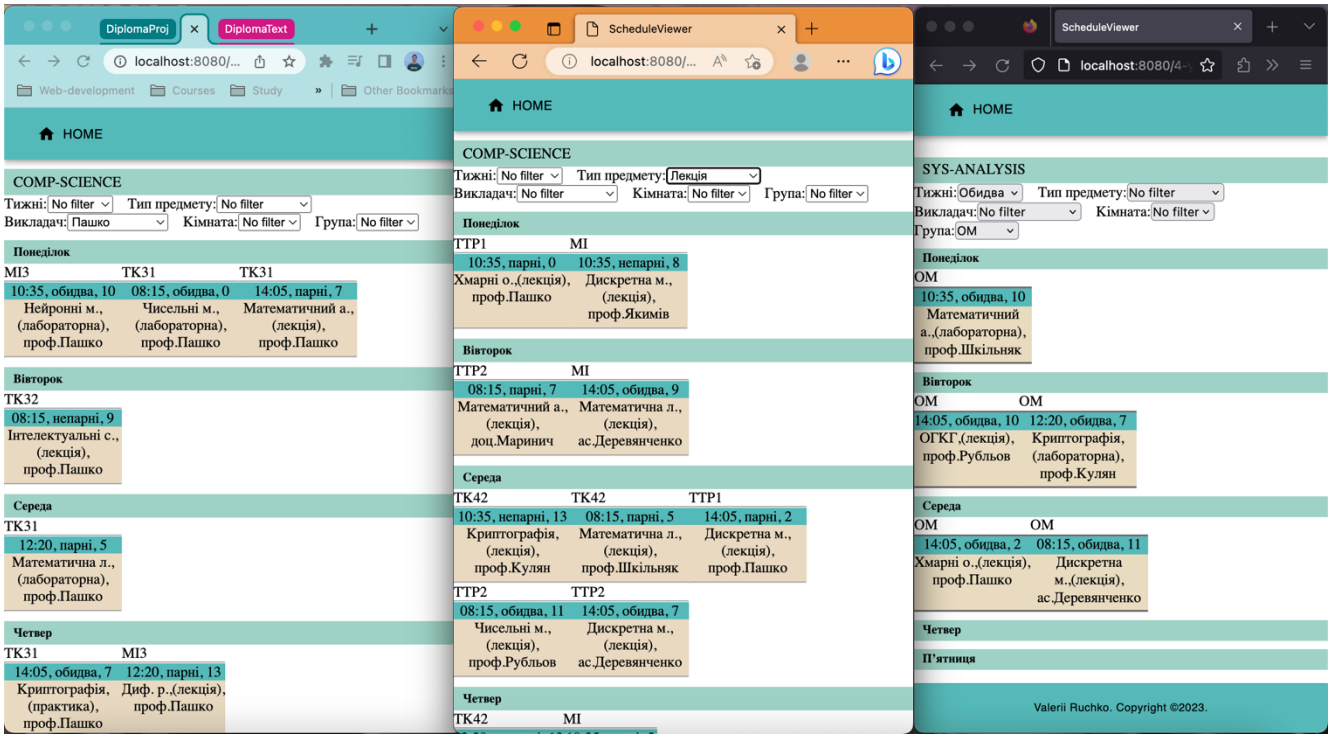

Рисунок 3.7 — Демонстрація роботи застосунку у трьох різних браузерах

ВИСНОВКИ

Метою цієї роботи було створення вебзастосунку для перегляду розкладу з можливістю фільтрації для звуження кількості відображуваного контенту на сторінці. Під час її виконання усі заплановані кроки було виконано та планований вебзастосунок створено, а саме було:

— досліджено наявні інструменти, технології та засоби для розроблення вебзастосунків;
— опановано вміння, необхідні для використання обраних засобів реалізації вебзастосунку;
— визначено структурні елементи вебзастосунку, які необхідно реалізувати;
— спроєктовано структуру клієнтської та серверної частин застосунку, після чого її було реалізовано;
— спроєктовано архітектуру бази даних та наповнено її тестовими даними;
— забезпечено взаємодію клієнта та сервера;
— реалізовано функціонал необхідний для навігації сайтом та фільтрування даних, що відображаються за різноманітними категоріями.

Також була проведена оцінка переваг та недоліків різних форм розповсюдження програмного забезпечення, визначення особливостей вимог та проблем, притаманних задачі, та подальший вибір найбільш прийнятної для завдання форми розповсюдження та реалізації.

У процесі втілення вебзастосунку було розглянуто роботу протоколу HTTP, а також різноманітні способи оптимізації роботи вебзастосунків, задля використання яких було опрацьовано та використано фреймворк Next.js.

Окрім того, було проведено діагностику роботи вебзастосунку, задля того, щоб визначити мінімальні апаратні вимоги для його використання, та оцінити рівень його швидкодії.

Для побудованого застосунку було виконано експертну оцінку зручності його використання, яка виявилась високою.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Contributor T. What is web application (web apps) and its benefits [Електронний ресурс] / TechTarget Contributor // Software Quality. – Режим доступу: https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app (дата звернення: 24.05.2023). – Назва з екрана.
2. What is an interface? [Електронний ресурс] // CMU School of Computer Science. – Режим доступу: https://www.cs.cmu.edu/afs/cs/academic/class/15212-s98/www/java/tutorial/java/more/interfaceDef.html (дата звернення: 24.05.2023). – Назва з екрана.
3. What is a framework? [Електронний ресурс] // Codecademy Blog. – Режим доступу: https://www.codecademy.com/resources/blog/what-is-a-framework/ (дата звернення: 26.05.2023). – Назва з екрана.
4. Google Chrome Developers. WebAssembly: a new development paradigm for the web [Електронний ресурс], 2023 / Google Chrome Developers // YouTube. – Режим доступу: https://www.youtube.com/watch?v=RcHER-3gFXI (дата звернення: 24.05.2023). – Назва з екрана.
5. Web application vs desktop application: pros and cons [Електронний ресурс] // POSITIWISE. – Режим доступу: https://positiwise.com/blog/web-application-vs-desktop-application-pros-and-cons (дата звернення: 25.05.2023). – Назва з екрана.
6. Clark L. A crash course in just-in-time (JIT) compilers – Mozilla Hacks - the Web developer blog [Електронний ресурс] / Lin Clark // Mozilla Hacks – the Web developer blog. – Режим доступу: https://hacks.mozilla.org/2017/02/a-crash-course-in-just-in-time-jit-compilers/ (дата звернення: 26.05.2023). – Назва з екрана.
7. JavaScript with syntax for types. [Електронний ресурс] // TypeScript: JavaScript With Syntax For Types. – Режим доступу: https://www.typescriptlang.org/ (дата звернення: 26.05.2023). – Назва з екрана.

8. About | node.js [Електронний ресурс] // Node.js. – Режим доступу: https://nodejs.org/en/about (дата звернення: 26.05.2023). – Назва з екрана.
9. React [Електронний ресурс] // React. – Режим доступу: https://react.dev/ (дата звернення: 26.05.2023). – Назва з екрана.
10. Learn | next.js [Електронний ресурс] // Next.js by Vercel - The React Framework. – Режим доступу: https://nextjs.org/learn/foundations/about-nextjs/what-is-nextjs (дата звернення: 26.05.2023). – Назва з екрана.
11. RFC 2616: hypertext transfer protocol -- HTTP/1.1 [Електронний ресурс] // IETF Datatracker. – Режим доступу: https://datatracker.ietf.org/doc/html/rfc2616#section-10.5.1 (дата звернення: 24.05.2023). – Назва з екрана.
12. RFC 3986: uniform resource identifier (URI): generic syntax [Електронний ресурс] // IETF Datatracker. – Режим доступу: https://datatracker.ietf.org/doc/html/rfc3986 (дата звернення: 27.05.2023). – Назва з екрана.
13. HTTP response status codes - HTTP | MDN [Електронний ресурс] // MDN Web Docs. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status (дата звернення: 27.05.2023). – Назва з екрана.
14. TypeORM - amazing ORM for typescript and javascript (ES7, ES6, ES5) [Електронний ресурс] // TypeORM - amazing ORM for typescript and javascript (ES7, ES6, ES5). – Режим доступу: https://typeorm.io/ (дата звернення: 28.05.2023). – Назва з екрана.
15. Awati R. What is object-relational mapping (ORM)? – TechTarget Definition [Електронний ресурс] / Rahul Awati // TheServerSide.com. – Режим доступу: https://www.theserverside.com/definition/object-relational-mapping-ORM (дата звернення: 28.05.2023). – Назва з екрана.
16. Getting started with relational databases: one-to-many relationship [Електронний ресурс] // The Support Group Blog. – Режим доступу: https://blog.supportgroup.com/getting-started-with-relational-databases-one-to-many-relationship (дата звернення: 28.05.2023). – Назва з екрана.

17. Chrome browser system requirements - chrome enterprise and education help [Електронний ресурс] // Google Help. – Режим доступу: https://support.google.com/chrome/a/answer/7100626?hl=en (дата звернення: 28.05.2023). – Назва з екрана.
18. Lighthouse overview - chrome developers [Електронний ресурс] // Chrome Developers. – Режим доступу: https://developer.chrome.com/docs/lighthouse/overview/ (дата звернення: 28.05.2023). – Назва з екрана.
19. Nielsen's 10 usability heuristics - heurio [Електронний ресурс] // Heurio - The Simple Way to Collaborate on Websites. – Режим доступу: https://www.heurio.co/nielsens-10-usability-heuristics (дата звернення: 30.05.2023). – Назва з екрана.

ДОДАТОК А
ДЕМОНСТРАЦІЯ РОБОТИ ЗАСТОСУНКУ

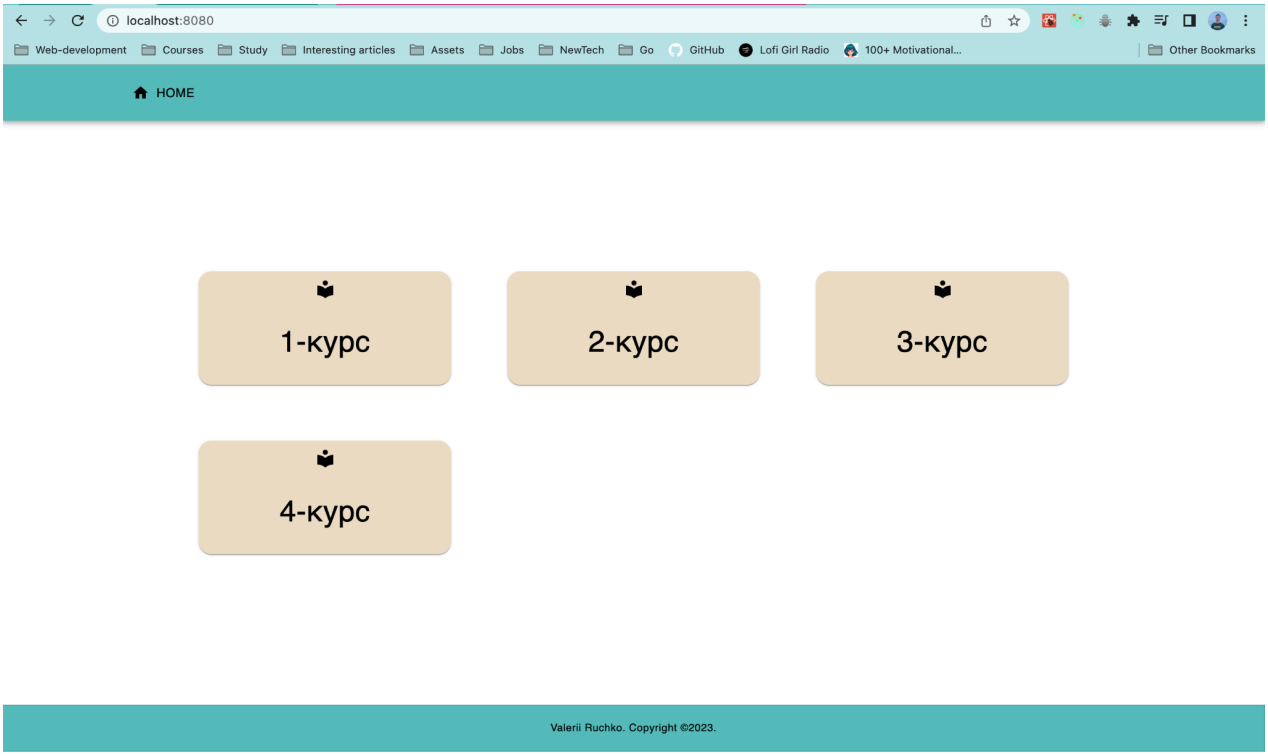

Рисунок А.1 – Початкова сторінка застосунку для вибору курсу

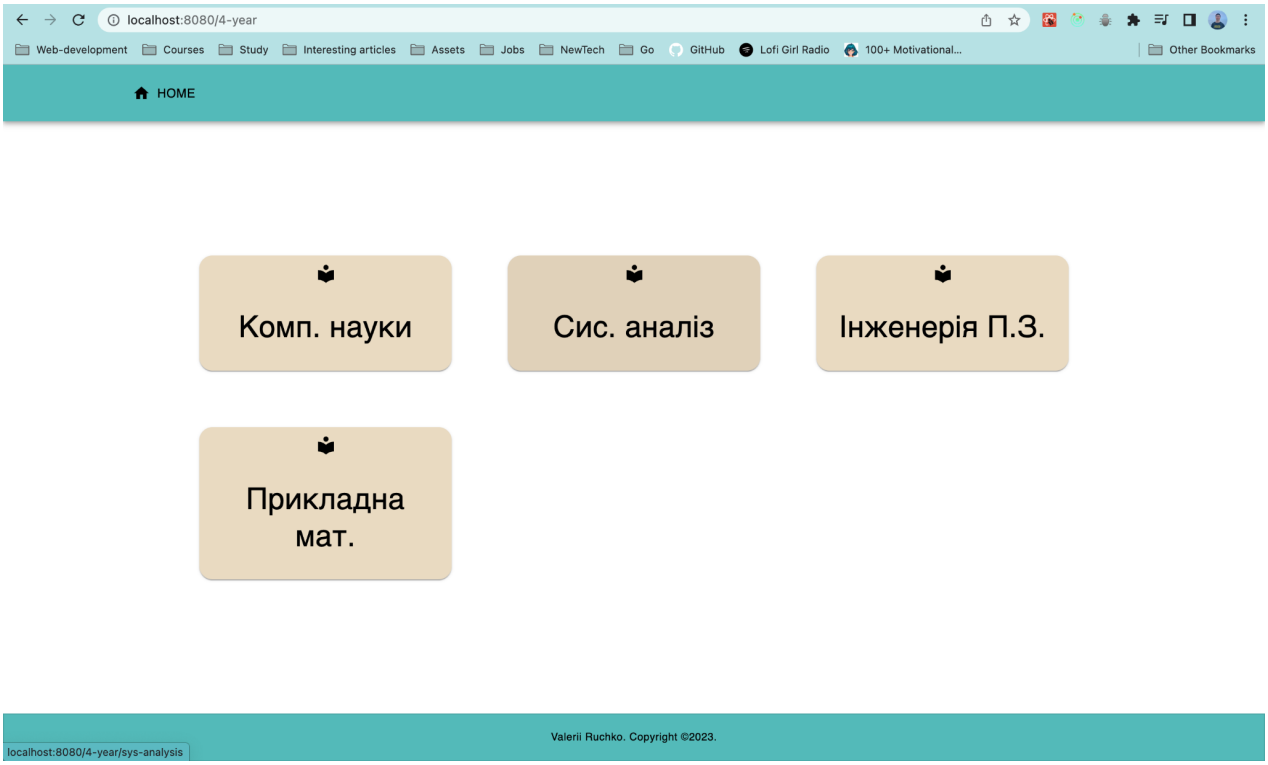

Рисунок А.2 – Сторінка вибору освітньої програми

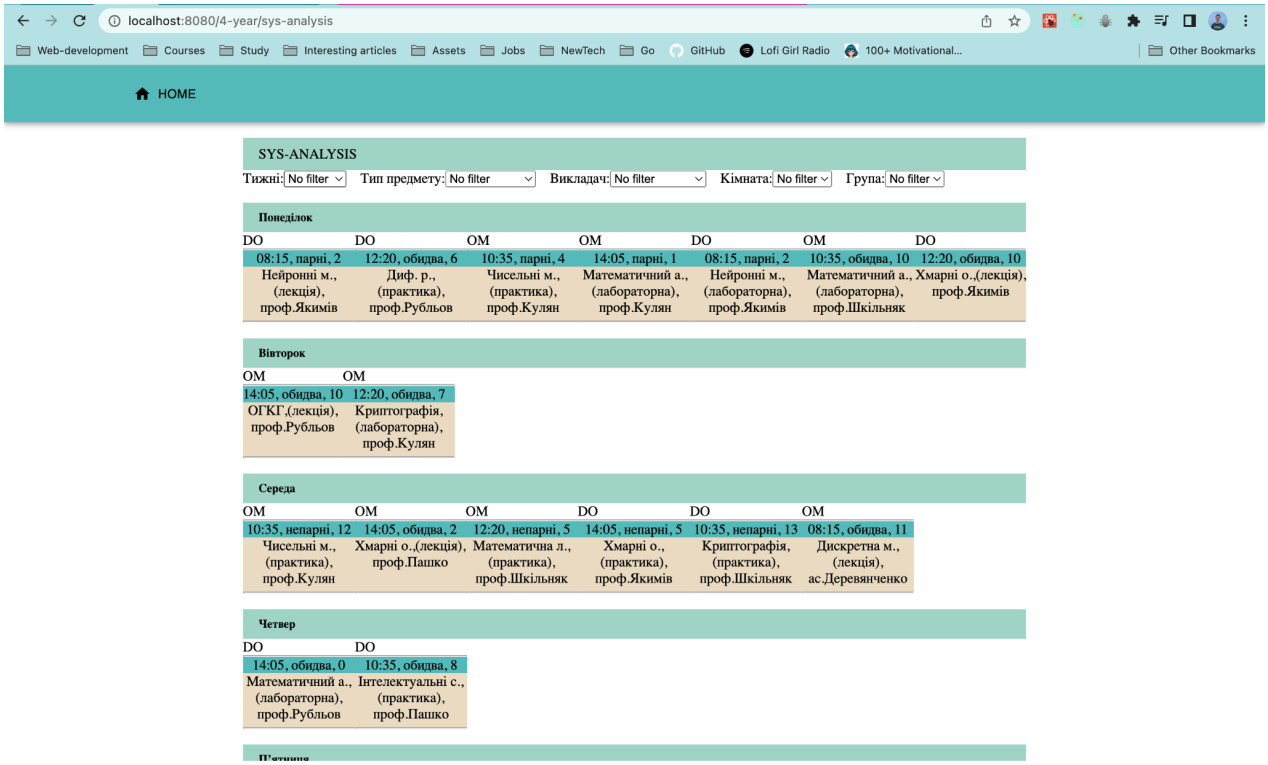

Рисунок А.3 – Сторінка відображення вибраної користувачем частини розкладу

ДОДАТОК Б
СТРУКТУРА ТАБЛИЦІ SCHEDULE_ITEM

```
@Entity()
export class Schedule_item {
  @PrimaryGeneratedColumn("uuid")
  id!: string;
  @Column()
  weekDay!: string;
  @Column()
  classNum!: number;
  @Column()
  weekType!: string;
  @ManyToOne(
    () => Subjects,
    (subject) => {
      subject.id;
    },
    {
      cascade: true,
      onDelete: "CASCADE",
      eager: true,
    }
  )
  subjects!: Subjects;
  …. // повторювані елементи коду для інших полів та зв’язків
}
```