Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ

завідувач кафедри

мережевих та інтернет технологій

_____ Юрій КРАВЧЕНКО

«_____»_____2023 року

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

галузі знань 17 «Електроніка та телекомунікації»
за спеціальністю 172 «Телекомунікації та радіотехніка»
освітньо-професійна програма «Мережеві та інтернет технологій»

на тему:

РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ УПРАВЛІННЯ РОЗКЛАДОМ ЗАНЯТЬ ДЛЯ ПЕВНОЇ КАФЕДРИ З ПІДСИСТЕМОЮ ПОВІДОМЛЕННЯ СТУДЕНТІВ ПРО НАЙБЛИЖЧІ ТА ПОТОЧНІ ПАРИ

Виконав: студент групи МІТ -41
Мирослав ТОЛОШНИЙ _____
(ім'я та ПРІЗВИЩЕ) (підпис)

Керівник: завідувач кафедри мережевих та інтернет технологій
д.т.н., професор Юрій КРАВЧЕНКО _____
(науковий ступінь, вчене звання, ім'я та ПРІЗВИЩЕ) (підпис)

Київ 2023

ЗМІСТ

ВСТУП

В сьогоденні в освітній сфері людство стикнулося з таким явищем, як дистанційне навчання. Ні для кого не секрет, що багато різних навчальних закладів вже не тільки почали використовувати, а й активно продовжують, використовувати інформаційні технології задля комфортного проведення занять як з боку викладача, так і з боку студента. І тут ми, фактично, бачимо таку проблему, як планування занять, їх розподіл між викладачами та студентами, та швидкий доступ до їхнього редагування. І ми бачимо, серед запропонованої нам низки додатків якісь незрозумілі додатки з навантаженими інтерфейсами, в яких для того, щоб розібратись, потрібно провести принаймні півроку над інструкціями. Тож було вирішено, що нам необхідне гнучке рішення, яке б дозволило стартувати якомога швидше у внесенні розкладу занять, і учні не чекали місяцями на розклад, а могли долучитися до його огляду вже зараз. Також доволі важливим фактором постає швидкість отримання змін у розкладі студентами, що одразу наштовхує на певні особливості бази даних. Тож, як міні-висновок, можемо сказати, що хотілося б побачити деяку супер-пігулку, яка б вирішила проблему заплутаності студентів, які не знають, де саме в них зараз буде пара! Звичайно, ми маємо,наприклад, Гугл Классрум та Міт (див. рис 0.1)

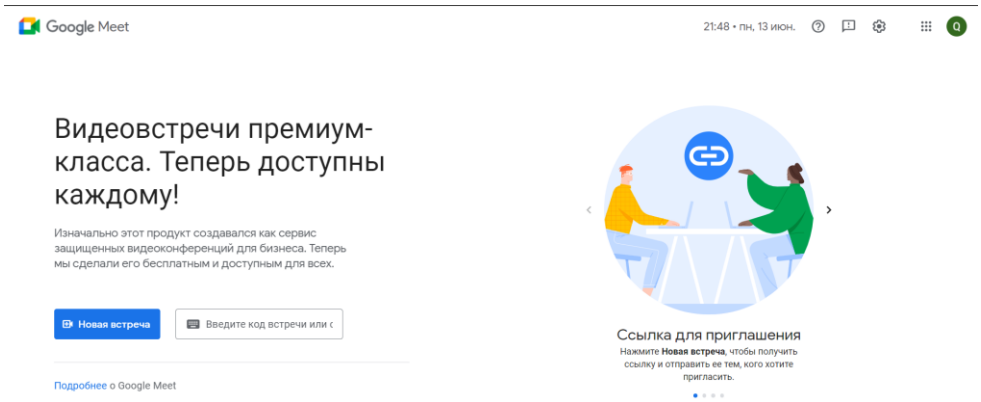


Рисунок 0.1 — Google Meet

але, чесно кажучи, цей інструмент є доволі важким для освоєння з нуля, та і функціонал трохи не той, що нам потрібен, там суто створення конференцій, а в нас - учбова інформаційна система.

ОГЛЯД КОНКУРЕНТІВ

Розглядаючи аналоги не можна не згадати такі додатки, як:

1. Zoom, який націлений саме на онлайн-конференції і яким користується переважна частина студентів по всіх ВНЗ країни і є одним із найпопулярніших інструментів для цього. Він пропонує можливість проводити односпрямовані та багатоспрямовані відеовиклики, обмін повідомленнями та спільну роботу над документами. Zoom також підтримує великі групові конференції з можливістю приєднання до кількох сотень учасників.
2. Microsoft Teams, що використовується переважно в ентерпрайз корпораціях та слугує незамінним помічником менеджмент шару будь-якого айті сегменту, є інтегрованою платформою для командної роботи, що включає в себе можливості для відеоконференцій. Вона надає інструменти для обміну повідомленнями, колаборації над документами та спільної роботи. Teams також інтегрований з іншими сервісами Microsoft, такими як Office 365. Тут же хочеться згадати Teams for Education, платформу, що надається Microsoft, спеціально розроблену для освітніх установ. Teams for Education пропонує вчителям та учням інструменти для створення віртуального класу, обміну матеріалами, завдань, обговорень та відеоконференцій. Він також інтегрований з іншими сервісами Microsoft, такими як OneNote та Office 365.
3. Cisco Webex: - популярний інструмент для відеоконференцій і віртуальних зборів. Він пропонує широкий спектр функцій,

включно з обміном повідомленнями, спільною роботою над документами, можливістю запису зборів і використанням віртуальної дошки.

4. Jitsi Meet - це відкрите програмне забезпечення для відеоконференцій, яке пропонує безкоштовні відеодзвінки високої якості. Він дає змогу створювати приватні кімнати для віртуальних зустрічей і підтримує спільну роботу над документами та обмін повідомленнями.
5. І навіть Skype - це одна з найстаріших платформ для відеоконференцій, яка була придбана Microsoft. Він пропонує можливості для односпрямованих і багатоспрямованих відеодзвінків, обміну повідомленнями та передачі файлів. Skype також підтримує групові виклики з кількома учасниками.
6. Schoology: Schoology - це платформа управління навчанням, призначена для шкіл та університетів. Вона дає змогу вчителям створювати курси, ділитися матеріалами, завданнями та спілкуватися з учнями. Schoology також пропонує можливості для співпраці та оцінювання успішності студентів.
7. Canvas - це платформа управління навчанням, широко використовувана у вищій освіті та школах. Вона дозволяє вчителям створювати курси, видавати завдання, вести онлайн-дискусії та оцінювати роботи студентів. Canvas також підтримує інтеграцію з іншими інструментами та сервісами.
8. Moodle - це відкрита платформа управління навчанням, яка широко використовується в освітніх закладах. Вона надає можливості для створення курсів, взаємодії з учнями, видачі завдань і перевірки успішності. Moodle також підтримує різноманітні плагіни та інтеграції.

9. Edmodo - це соціальна платформа для навчання, яка дає змогу вчителям та учням спілкуватися, ділитися матеріалами, завданнями та зворотним зв'язком. Він пропонує інструменти для створення віртуального класу, а також підтримує можливість інтеграції з іншими сервісами.

ПІДСУМОК

Кожен з цих сервісів має свої переваги, такі як:

Широкий функціонал з обміном повідомленнями, інтеграції з іншими сервісами. Віртуальні класи та маніпуляції в них, Можливість проведення відеоконференцій і підтримка великих групових конференцій. Можливість запису зборів і використання віртуальної дошки. Одне із найголовніших - безкоштовні відеодзвінки високої якості. Також Створення приватних кімнат для віртуальних зустрічей.

І тут я хочу виділити спільні риси описаних вище сервісів: Колаборація та обмін інформацією: Як Google Meet, так і його аналоги забезпечують можливість спільної роботи та обміну інформацією між учасниками. Усі вони пропонують інструменти для обміну повідомленнями, спільної роботи над документами та завданнями. Інтеграція з іншими сервісами: Деякі аналоги Google Meet і Google Клас, такі як Microsoft Teams, інтегровані з іншими сервісами та інструментами. Це дає змогу користувачам працювати з іншими додатками та використовувати різні функції в рамках однієї платформи. Управління навчанням: можливість управління навчанням. Всі вони дають змогу викладачам створювати курси, видавати завдання, перевіряти успішність студентів і спілкуватися з учнями.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ ПРОЕКТУ

1.1 ПРОБЛЕМАТИКА В КЕРУВАННІ УЧБОВИМ ПРОЦЕСОМ

І наший розклад пар на цей рік складений у таблиці EXCEL, де ми його й переглядаємо, а викладачі його вписують у комірки (рис 1.1). І така картина виглядає хаотично, адже трохи незрозуміло, куди тут дивитися, і, що найголовніше, це не дуже зручно з точки зору UX:

B	C	D
час	3 курс (21 студ.) 24.01-22.05	
	3 курс	
	група МІТ	
час	група МІТ-**31**	
9.00-10.20	**Соціально політичні студії (С) 5т**	
10.30-11.50	**Бази даних та інформаційні системи (лаб) [31.01-14.03]** доц. Миколайчук Р.А.	**Об'єктно-орієнтовне програмування(лаб.) [07.03-14.03]** Герасименко О. Ю.
	Об'єктно-орієнтовне програмування(лаб.) [21.03-18.04, 16.05] Герасименко О. Ю.	
12.10-13.30	**Об'єктно-орієнтовне програмування(Пр) [31.01-21.03]** Герасименко О. Ю. Бази даних та інформаційні системи (Пр.) [28.03-18.04, 16.05]+3 доц. Миколайчук Р.А.	
13.40-15.00	**Об'єктно-орієнтовне програмування(лаб.) [07.03-14.03]** Герасименко О. Ю.	**Бази даних та інформаційні системи (лаб) [31.01-14.03]** доц. Миколайчук Р.А.
15.10-16.30		**Об'єктно-орієнтовне програмування(лаб.) [21.03-18.04, 16.** Герасименко О. Ю.

Рисунок 1.1 — Старий вигляд розкладу занять

Тож така ситуація надихнула мене на створення єдиної системи планування розкладу, з чітким регламентом і простим та зрозумілим інтерфейсом, як для студента, так і для викладача. Також потрібно орієнтуватися саме на проблеми гнучкості зміни вищезазначеного розкладу! Адже не раз студенти стикаються з тим, що не знають, в яку аудиторію, або ж на яку конференцію їм заходити. Робимо міні-висновок: потрібен простий додаток з мінімалістичним інтерфейсом для перегляду поточної та найближчої пар. А також API з більш сучасним підходом до редагування записів у розкладі, і, яке б могло по-новому представити цей же розклад. Назвати додаток було вирішено ***Scheduler***. Scheduler - це система для спрощеного керування розкладом занять. Він складається із таких підсистем, як:

Editor - Керуючий додаток із реєстрацією викладачів. Також містить у собі засоби для створення, видалення та зміни списку викладачів, предметів, груп та найголовніше - розкладів.

Admin - REST система розподілу АПІ та ендпоінтів із набором CRUD операцій та засобів фільтрування, сортування та менеджменту всіх наявних сутностей.

UI - Студентська підсистема для учнів із можливістю переглянути розклад + поточну і наступну пари за токеном групи. (Токен видається викладачем).

1.2 USER FLOW

Далі мною буде розглянуто весь шлях від реєстрації до створення розкладу занять і перегляду в студентському додатку (UI) спільного результату роботи систем.Крок 1 - Реєстрація як викладача. Спершу потрібно перейти до **Editor** за маршрутом */register* (див. Рис 1.2)

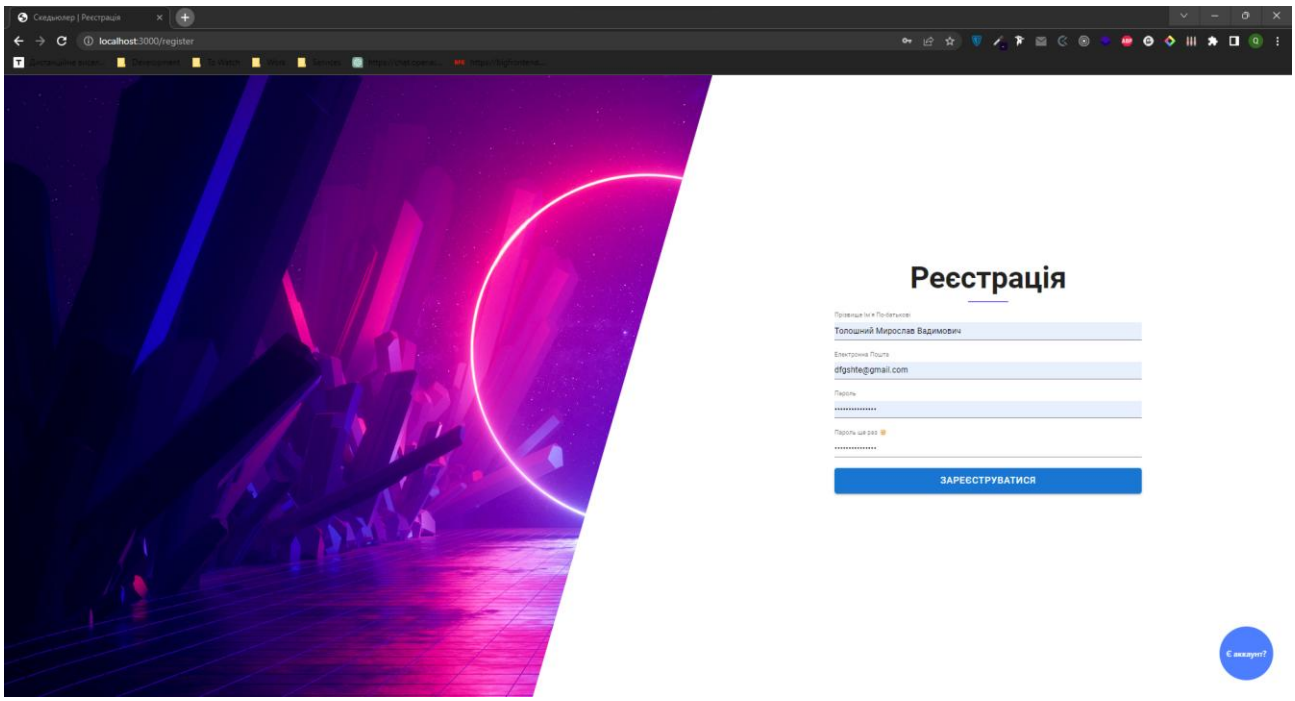


Рисунок 1.2 — Сторінка реєстрації в Editor

Крок 2 - Вхід за допомогою логіну та паролю (своїх даних або credentials.

Переходимо на сторінку */login* та вводимо логін та пароль використані під час реєстрації (див. Рис 1.3)

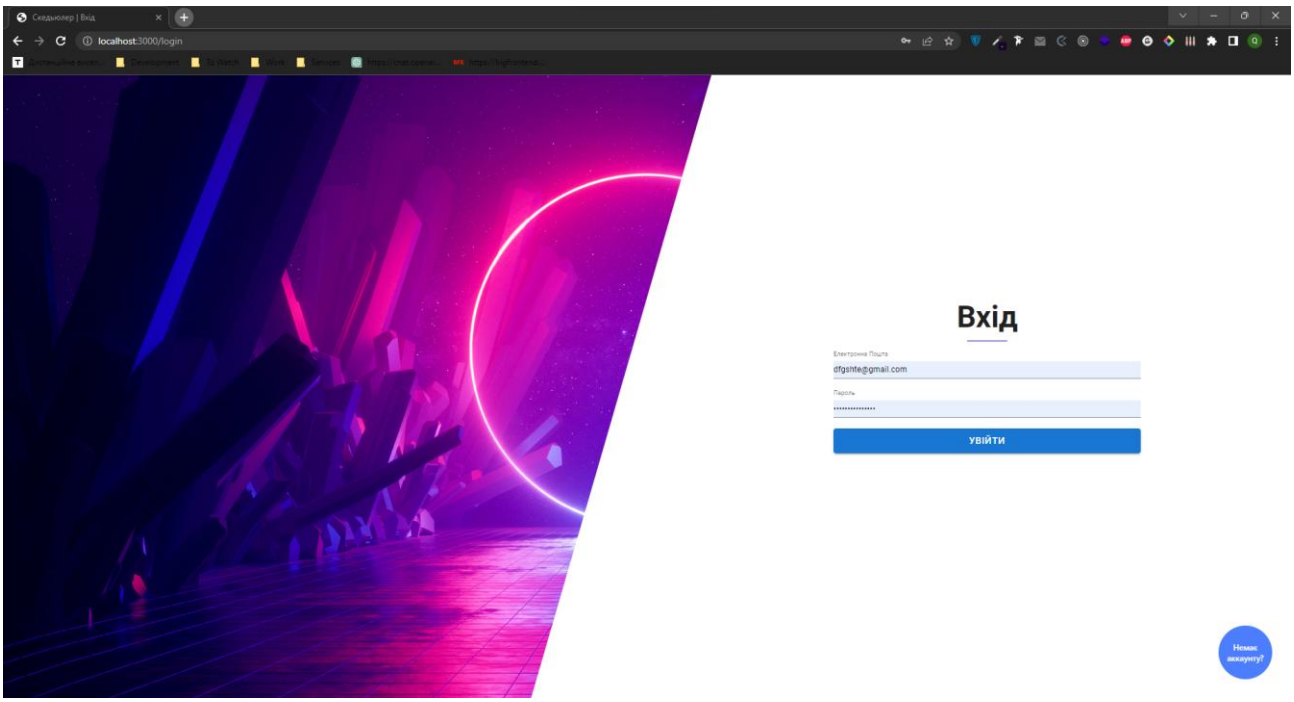


Рисунок 1.3 — Сторінка входу в Editor

Крок 3 - Дашборд та його дії (actions) з управління. Після успішного логіну нас перекине на сторінку */dashboard*, де бачимо доступні операції (див. Рис 1.4)

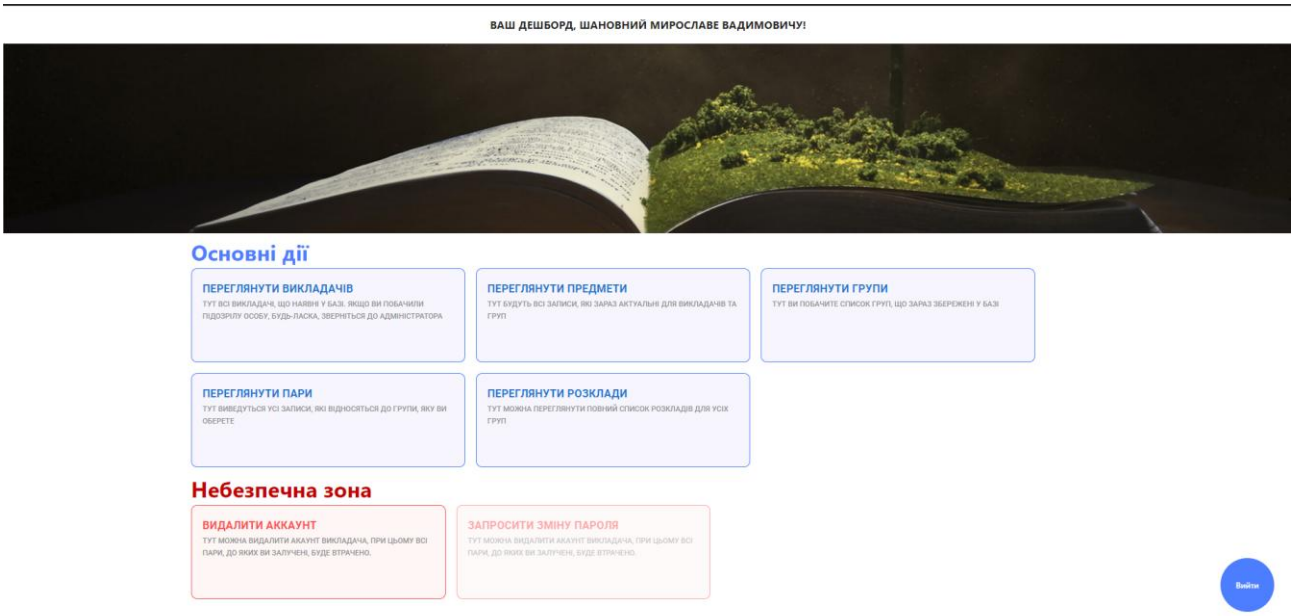


Рисунок 1.4 — Дашборд в Editor

Крок 4 - додати групу. Натиснувши на відповідне посилання на сторінці

можна потрапити всередину та створити необхідну групу, наприклад я додав MIT-51, тепер її можна побачити тут (див. Рис 1.5)

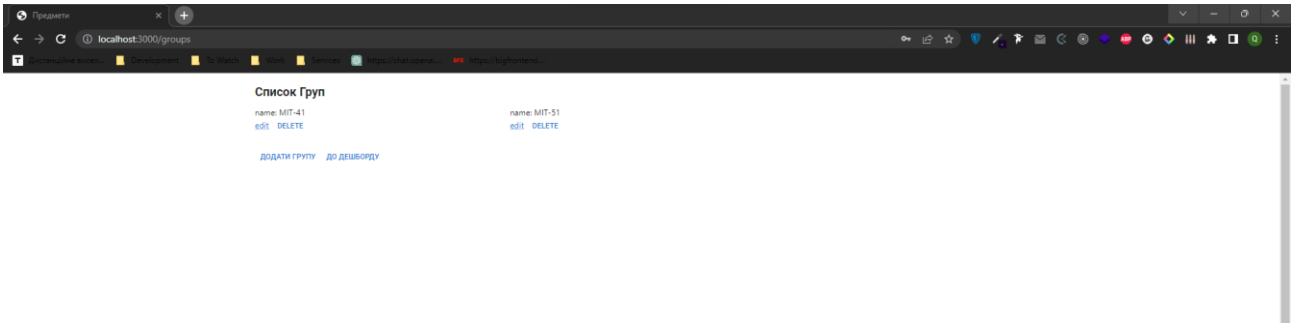


Рисунок 1.5 — Сторінка груп в Editor

Крок 5 - Повторити із додаванням предмету, пари, та розкладом. Тут потрібно уточнити, що предмет - це сутність, що зв'язана з викладачем, має назву, але не має часу проведення. А пара - це саме предмет з певним часом проведення. Час проведення задається в форматі (ГГ:ХХ - ГГ:ХХ). На цьому ж кроці ми можемо внести зміни в, наприклад, розклад або інший будь-який запис, якщо його сформовано неправильно. Також на сторінці розкладів для кожної групи доступний токен, саме його потрібно копіювати та надавати студентам для доступу до Скедьюлеру (див. Рис 1.6).

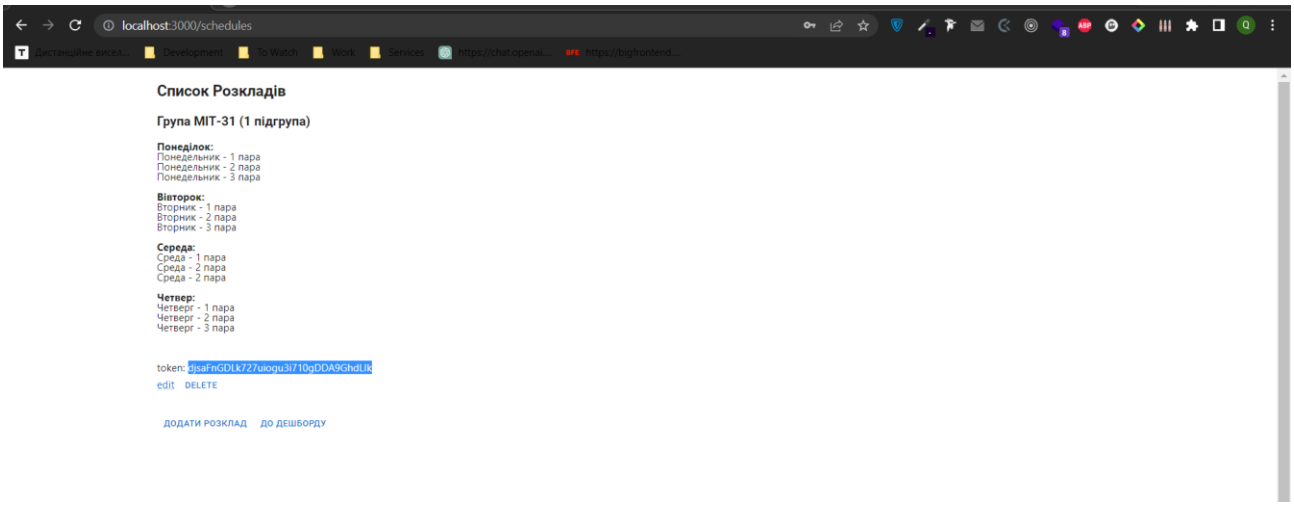


Рисунок 1.6 — Сторінка з розкладами та токен групи

Крок 6 - Перевірка розкладу пар на студентському додатку. Для цього треба перейти на UI частину за відповідною URL-адресою. (див. Рис 1.7)

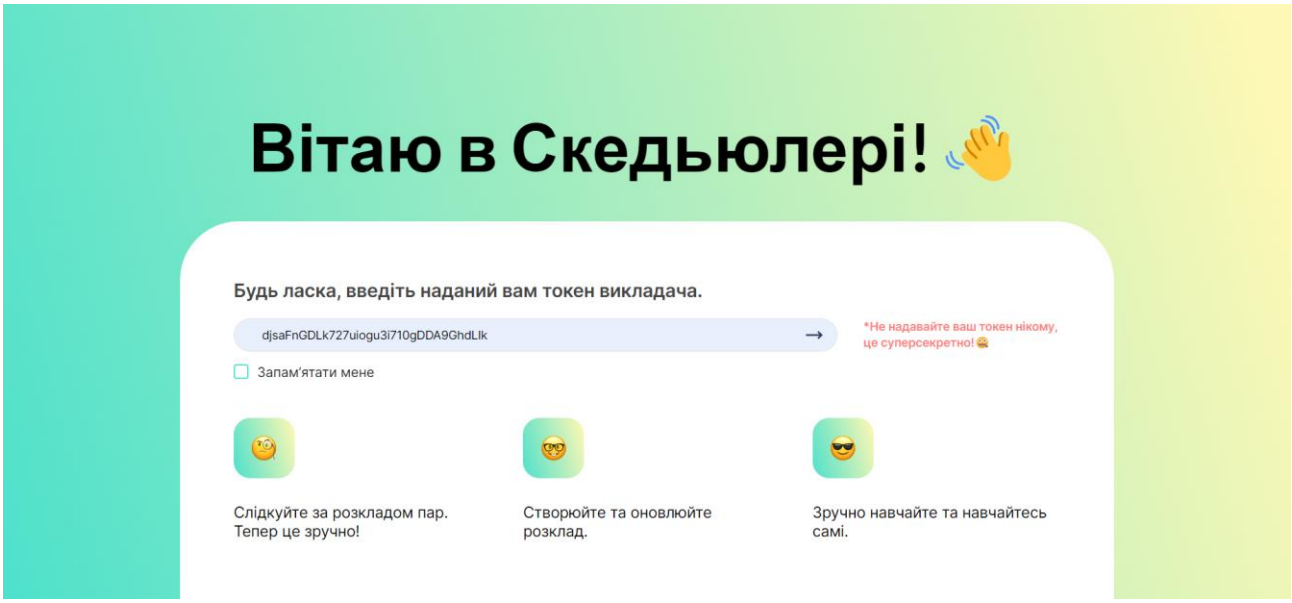


Рисунок 1.7 — Головна сторінка Скедюлера UI

Крок 7 - ввести токен групи, який взято зі сторінки розкладів у відповідне поле та дочекатися відповіді від серверу. Далі побачимо найближчі пари, які маємо.

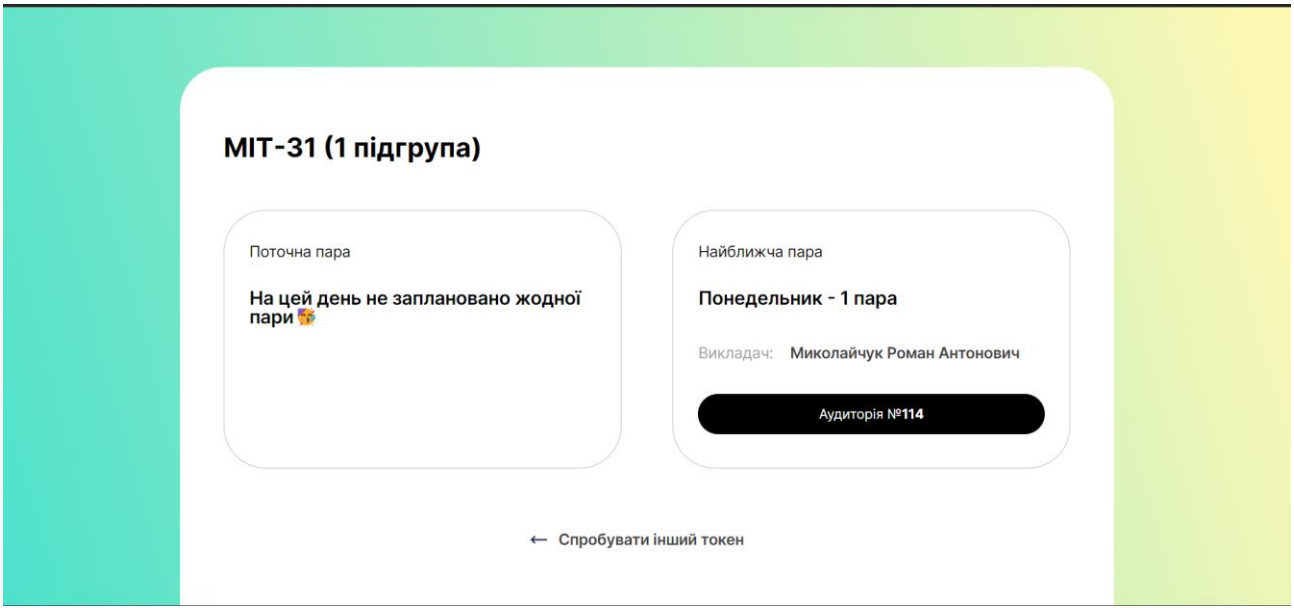


Рисунок 1.8 — Результат роботи Скедьюлера

Якщо пара проводиться очно в аудиторії - буде номер аудиторії, а якщо онлайн - то буде посилання на зустріч в Зум або в Гугл Міт.

РОЗДІЛ 2. ПЛАНУВАННЯ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ ДЛЯ ЗРУЧНОЇ ТА БЕЗПЕЧНОЇ ВЗАЄМОДІЇ

Враховуючи попередні ідеї в мене вимальовується приблизно наступна картина (рис 2.1):

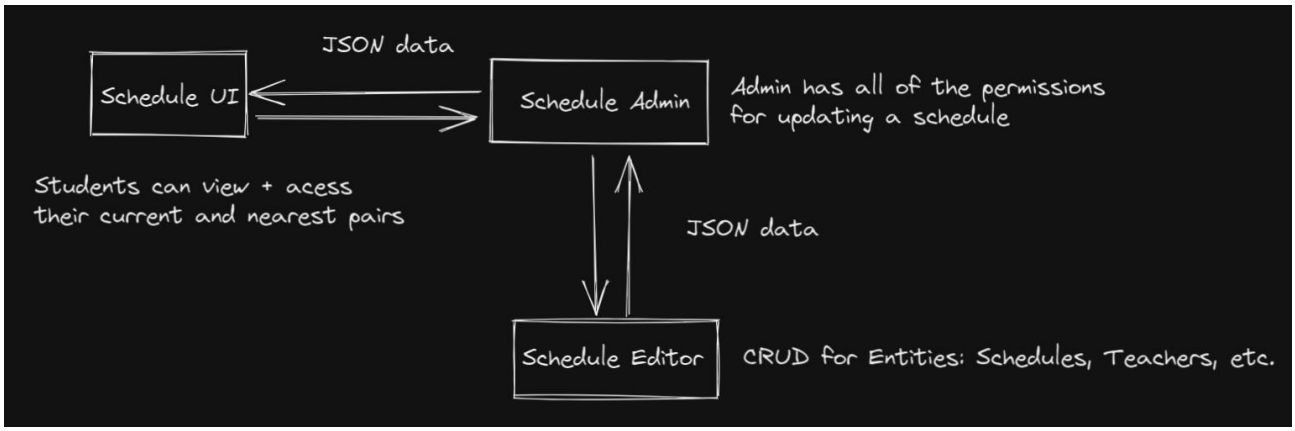


Рисунок 2.1 — Первинна схема взаємодії декомпозованих сервісів між собою

Маємо перший студентський сервіс (scheduler-ui) для доступу до пар, другий (scheduler-admin) загальний сервіс, що поєднує в собі функціонал повного та часткового доступу до розкладу і третій, суто викладацький інтерфейс для виконання необхідних операцій (scheduler-editor). Також до цієї схеми пізніше додається сторедж у вигляді бази даних і кінцева схема буде приблизно такою (див. рис 2.2):

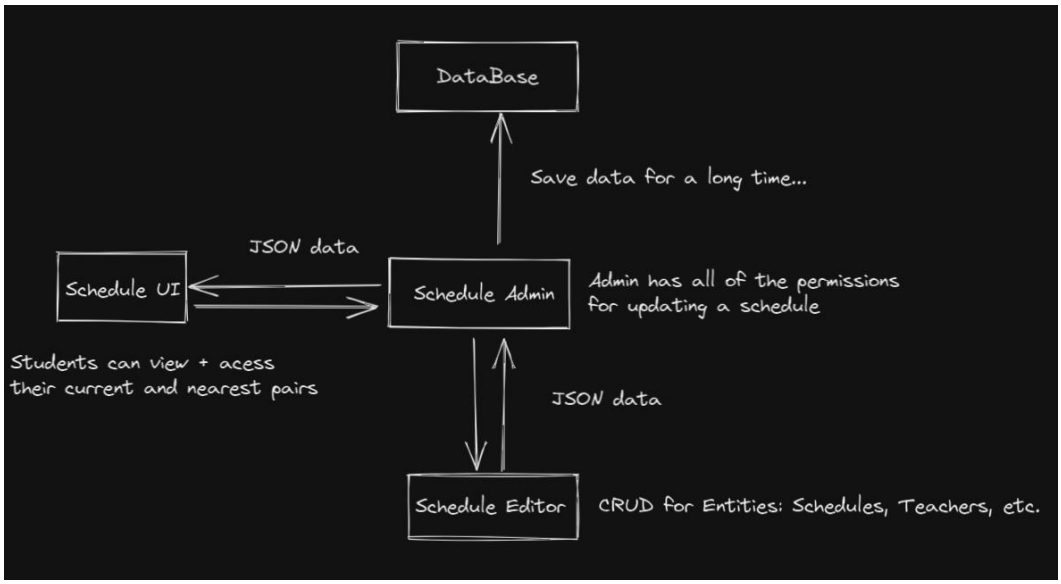


Рисунок 2.2 — Проміжна схема взаємодії декомпозованих сервісів між собою

Далі формуємо відображення для викладача (Views) Тут потрібно врахувати те, що буде дозвіл змінювати його, тож не треба перенавантажувати інтерфейс.

Правила легкого інтерфейсу в дизайні допомагають створити зрозумілий, інтуїтивно зрозумілий і приємний користувацький досвід. Ось кілька ключових правил: Простота і мінімалізм: Намагаюся створювати інтерфейс із мінімальною кількістю елементів та інформації. Уникайте перевантаженості екрана і зайвої складності. Зробіть інтерфейс простим і легким для сприйняття.

Чітка ієрархія: Організовую елементи інтерфейсу в логічній ієрархії. Використовую сітку і вирівнювання, щоб створити чітку структуру. Впорядковую елементи за важливістю і частотою використання.

Ясність і зрозумілість: Використовую зрозумілі та консистентні метафори, іконки та маркери, щоб допомогти користувачам зрозуміти функціональність і дії. Використовуйте зрозумілі тексти та інструкції, уникаючи складних термінів і фраз.

Колір і контрастність: Використовую обмежену палітру кольорів, яка добре поєднується і створює контраст між елементами інтерфейсу. Звертаю увагу на доступність і зручність читання, особливо для людей з обмеженим зором.

Навігація та чуйність: Забезпечую легку та інтуїтивно зрозумілу навігацію по додатку або веб-сайту. Створіть швидкий відгук на користувацькі дії, щоб запобігти затримкам і поліпшити сприйняття швидкості.

Консистентність: Підтримую консистентність візуального стилю, макетів і взаємодії між різними частинами інтерфейсу. Використовуйте повторювані елементи і шаблони, щоб допомогти користувачам швидко орієнтуватися і впізнавати інтерфейс (див рис 2.3).

Рисунок 2.3 — Схематичний інтерфейс для викладацького додатку

Особливо важливим аспектом підготовки даних є запис розкладу. Я починаю процес структурування бази даних, і першим кроком я ідентифікую основні сутності, які будуть присутні в моїй базі даних. Я задаюся питанням: "Які основні об'єкти будуть зберігатися і управлятися в системі?". Це можуть бути користувачі, продукти, замовлення і багато іншого. Для кожної сутності я визначаю унікальний ідентифікатор, щоб гарантувати унікальність кожного запису.

Потім я переходжу до визначення відносин між сутностями. Я розмірковую про те, які сутності пов'язані між собою і як ці зв'язки виражаються. Деякі зв'язки можуть бути один-до-одного, інші - один-до-багатьох або багато-до-багатьох. Я створюю зв'язки між таблицями, використовуючи зовнішні ключі, щоб підтримувати ці відносини.

Коли відносини визначено, я застосовую правила нормалізації, щоб усунути надмірність даних і забезпечити цілісність бази даних. Я розбиваю дані на окремі таблиці та уникаю залежностей і аномалій. Це допомагає мені

зробити мою базу даних більш ефективною і легкою в обслуговуванні.

Кожній сутності я приписую атрибути, які описують її властивості. Я встановлюю правильні типи даних для кожного атрибута і додаю обмеження, як-от унікальність і обов'язковість, щоб гарантувати правильність і узгодженість даних. Одним із важливих аспектів вже структурування бази даних є індексування. Я розробляю стратегію індексування, щоб поліпшити продуктивність запитів. Правильно створені індекси допомагають швидко знаходити та витягувати дані на основі певних стовпців. Я намагаюся знайти баланс між кількістю індексів та їхнім оновленням, щоб не створювати зайве навантаження на базу даних (див. рис. 2.4).

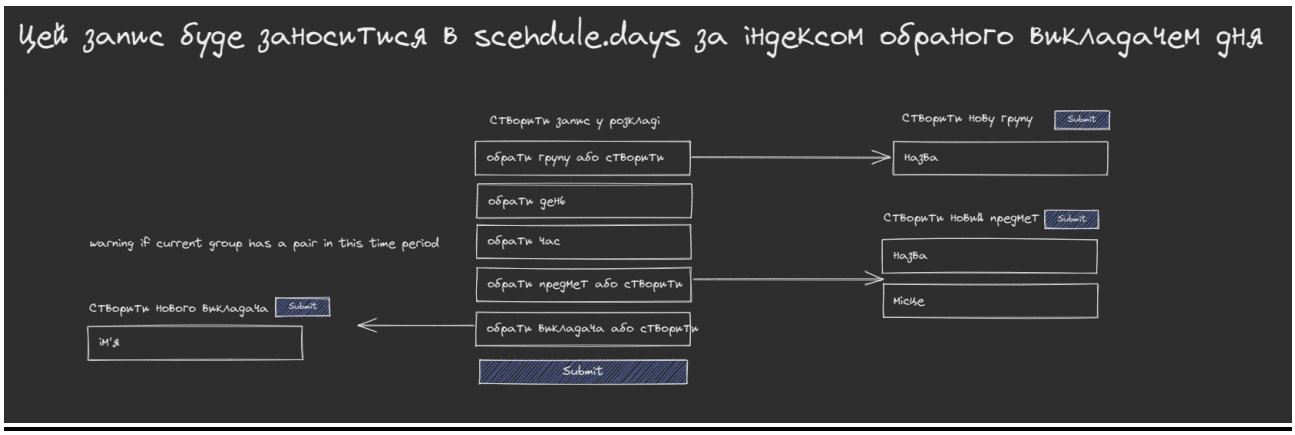


Рисунок 2.4 — Схема запису сутності розкладу до БД за індексом

РОЗДІЛ 3. ОБРАННЯ СТЕКУ ТЕХНОЛОГІЙ ПІД ЗАДАНІ ВИМОГИ

Після того, як схама була складена і функціонал був поділений між сервісами, ми можемо обрати технології, які надалі ми використовуємо для розробки кожного із сервісів:

- **MUI** (MUI пропонує повний набір інструментів інтерфейсу користувача, які допоможуть нам швидко додавати нові функції. Також є можливість додати свою власну систему проектування до готових до виробництва компонентів)
- **JavaScript | React + Next.js** (Фреймворк React для Розробки, Next.js надає найкращий розробницький досвід з усіма функціями, необхідними для розробки: гібридний статичний і серверний рендеринг, підтримка TypeScript, інтелектуальне об’єднання, попередня вибірка маршрутизації, і, на великий рахунок, конфігурація не потрібна)
- **TypeScript** (Це строго типізована мова програмування, яка базується на JavaScript, що дає вам кращі інструменти в будь-якому масштабі)
- **Redis** (Сховище даних у пам’яті з відкритим вихідним кодом, яке використовується мільйонами розробників як база даних, кеш-пам’ять, механізм потокової передачі та посередник повідомлень)
- **Vercel** (Дозволяє розробникам створювати та публікувати цілі веб-доданки, а також створює продукти для розробників такі, як Next.js)
- **Heroku** (Платформа PAAS, заснована на керованій контейнерній системі, з інтегрованими службами даних і потужною екосистемою, для розгортання та запуску сучасних програм. Досвід розробників Heroku — це підхід, орієнтований на додатки, до доставки

забезпечення, інтегрований із найпопулярнішими інструментами та робочими процесами розробників сьогодні).

- **C#** (Мова програмування, яка дозволяє вам працювати швидше та ефективніше інтегрувати наші системи)**.**
- **ASP.NET** ASP.NET (Active Server Pages.NET) являється фреймворком для створення веб-додатків, розробленим компанією Microsoft. І ось декілька переваг ASP.NET: Платформа .NET: ASP.NET базується на платформі .NET, яка забезпечує потужний інструментарій для розробки і управління веб-додатками. Він пропонує нам, як розробникам, множину бібліотек і функцій, які значно спрощують розробку і підвищують ефективність (performance).
- **MSSQL** (реляційна система управління базами даних (СУБД), розроблена компанією Microsoft. Вона надає потужні можливості для зберігання, управління та аналізу структурованих даних).
- **Docker** (Відкрита платформа для розробки, доставки та запуску програм. Docker дає змогу відокремити програми від інфраструктури, щоб ми могли швидко постачати програмне забезпечення. За допомогою Docker ми можемо керувати своєю інфраструктурою так само, як і своїми програмами. Використовуючи переваги методології Docker для швидкої доставки, тестування та розгортання коду, ми можемо значно скоротити затримку між написанням коду та його запуском у production).
- **Jira** - це популярна система відстеження помилок, управління проектами та спільної роботи, розроблена компанією Atlassian. Вона широко використовується в різних галузях і командами розробників програмного забезпечення для планування, відстеження та

управління завданнями і проектами. Надає можливість створювати й організовувати завдання, користувацькі історії, баг-репорти та інші елементи проекту. Ви можете призначати відповідальних, встановлювати терміни виконання, відстежувати прогрес і керувати пріоритетами (див. рис. 3.1).

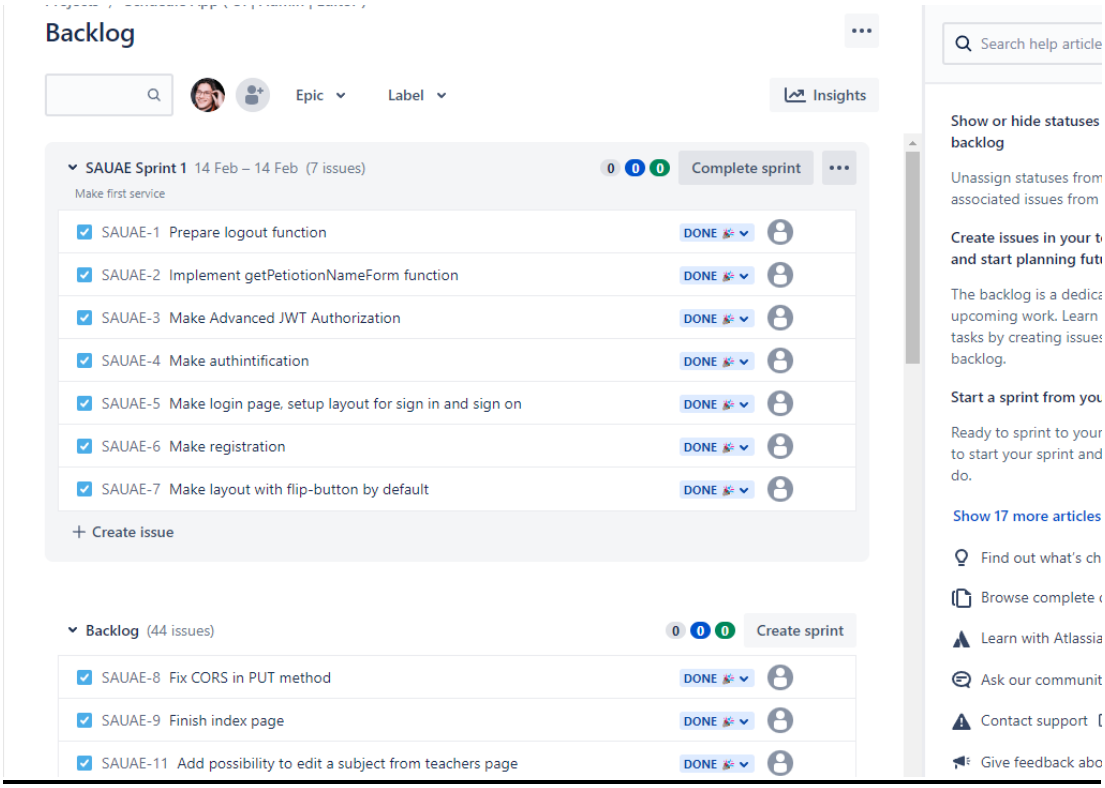


Рисунок 3.1 — Система Jira

- **Planet Scale** спеціалізується на хмарних базах даних, з фокусом на горизонтальному масштабуванні та реплікації для додатків, що використовують бази даних MySQL. Вони пропонують PlanetScaleDB - повністю керовану хмарну базу даних, яка забезпечує високу доступність, масштабованість і відмовостійкість (див. рис. 3.2).

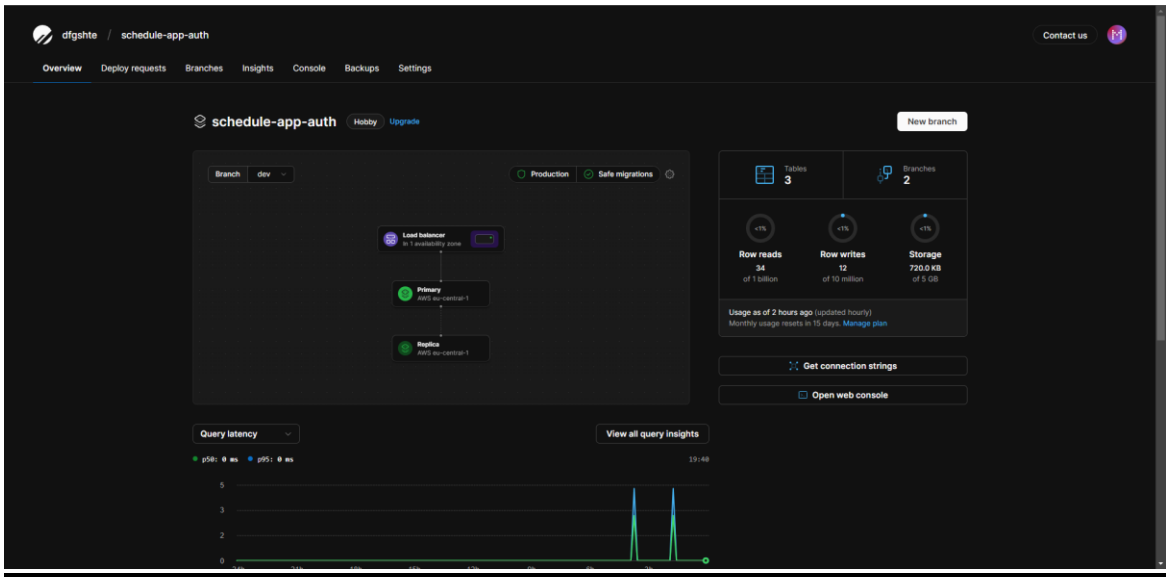


Рисунок 3.2 — Хмарна БД Planet Scale

Next.js пропонує злегка розважливий та оптимізований підхід до створення додатків з використанням React. Це галузевий стандарт, і ми пишаємося тим, що будуємо на його основі. Prisma - це найкращий спосіб працювати з базами даних на TypeScript. Вона надає простий, типо-безпечний API для запитів до вашої бази даних і може використовуватися з більшістю діалектів SQL (і Mongo теж!). TypeScript допоможе вам стати кращим веб-розробником. Незалежно від того, чи ви новачок у JS, чи досвідчений професіонал, "строгість" TypeScript забезпечує більш плавну побудову. Також мені потрібна гнучка, безпечна і масштабована авторизація, NextAuth.js - найкращий варіант. Він підключається до вже існуючої бази даних і надає простий API для управління користувачами і сесіями.

Скедьюлер орієнтований під серверний рендеринг, я зосередився на цій технології тому що вона має певний перелік переваг над статичними сайтами, а саме: Покращена продуктивність: При використанні SSR веб-сторінки рендеряться на сервері перед надсиланням клієнту. Як результат, користувач отримує повністю готову сторінку, що покращує час відгуку і знижує затримку, особливо на повільних або слабких пристроях. Покращена SEO-оптимізація: Пошукові системи зазвичай надають перевагу веб-сторінкам із повним вмістом,

доступним відразу ж під час завантаження сторінки. SSR дає змогу пошуковим системам легко проіндексувати вміст сторінки, що може позитивно позначитися на SEO-рейтингу вашого веб-сайту. Краща підтримка соціальних медіа: Під час обміну посиланнями на веб-сторінки в соціальних мережах або месенджерах, попередній перегляд вмісту (наприклад, заголовка, опису та зображень) зазвичай використовує метадані, доступні в HTML-коді сторінки. SSR дає змогу надати повну інформацію про сторінку, що покращує візуальне відображення під час обміну посиланнями. Покращена доступність і SEO для користувачів з обмеженими можливостями: Повністю сформовані веб-сторінки, одержувані за допомогою SSR, полегшують доступність веб-сайту для користувачів з обмеженими можливостями, такими як слабкий зір або використання скрінрідерів. Зручність розробки: SSR забезпечує однаковість коду і логіки між серверною і клієнтською частинами програми. Це може спростити розробку і підтримку коду, оскільки немає необхідності дублювати логіку на сервері і в браузері. Краща безпека: При використанні SSR, сервер може контролювати доступ до даних і обробку запитів, що покращує безпеку програми. Це дає змогу обробляти конфіденційні дані на сервері, мінімізуючи ризики витоку даних на клієнтській стороні. Однак, слід зазначити, що SSR також має свої недоліки і вимагає додаткових ресурсів на сервері для виконання рендерингу сторінок. Це працює таким чином, що кожна сторінка завантажується у вигляді HTML та CSS, далі “гідрується” за допомогою відповідного чанку JS і вже тоді віддається на клієнт, тобто я не вантажу прикінцевій пристрій користувача непотрібним JavaScript кодом, а суто зосереджений на видачі лише необхідного контенту. Визначившись зі стеком технологій переходжу до розробки.

РОЗДІЛ 4. РОЗРОБКА ДОДАТКУ ДЛЯ ВИРАХУВАННЯ ПОТОЧНОЇ ТА НАСТУПНОЇ НАЙБЛИЖЧОЇ ПАРИ

Перший сервіс представляє собою фуллстек додаток, написаний на Next.js в якості фронтенд частини та кастомного серверу Express і NodeJS в якості беку (рис 4.1):

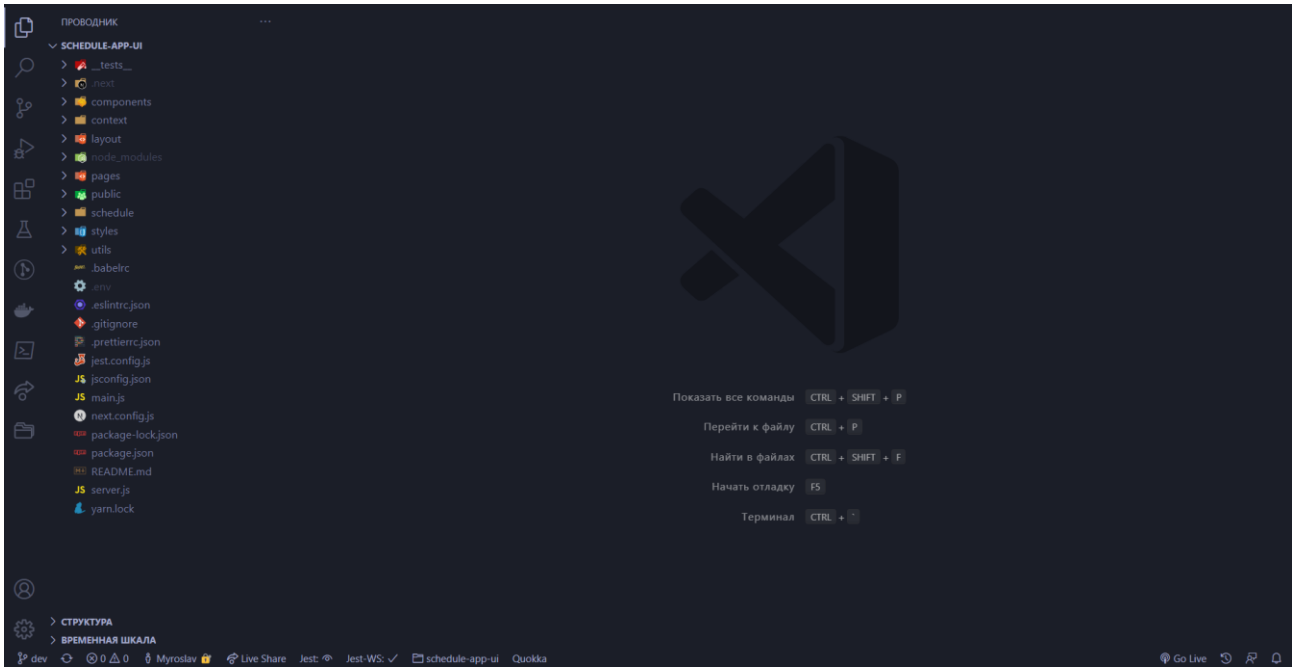
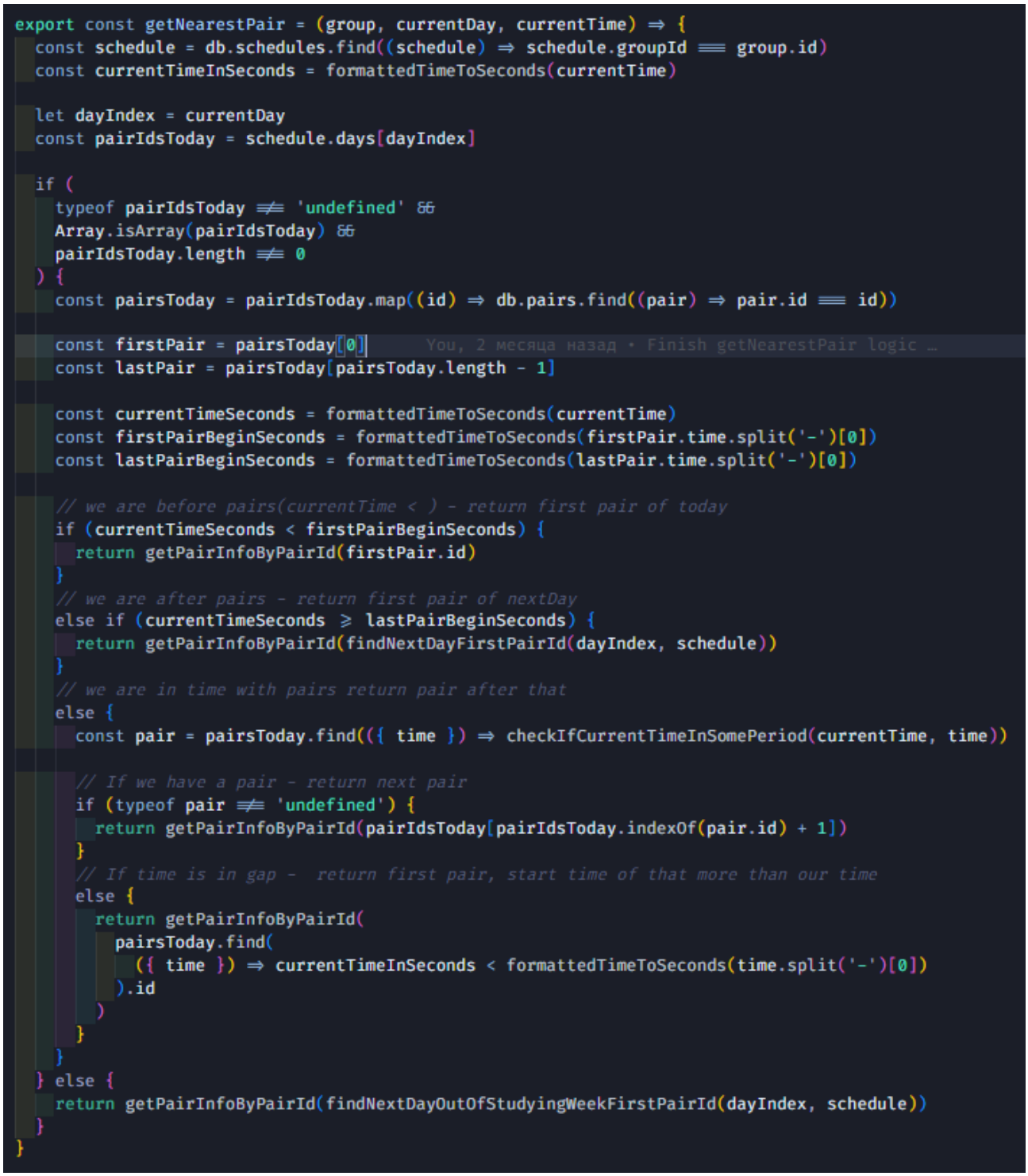


Рисунок 4.1 — Структура проекту Scheduler

Тут покладено core нашої програми для учнів, а саме - власноруч винайдений алгоритм підбору найближчої та поточної пари (рис 4.2 і рис 4.3):

```
export const getCurrentPair = (group, currentDay, currentTime) => {
  const schedule = db.schedules.find((schedule) => schedule.groupId === group.id)
  const pairIdsToday = schedule.days[currentDay]

  if (pairIdsToday.length === 0) throw new ScheduleError('На цей день не заплановано жодної пари')

  const pairsToday = pairIdsToday.map((pairId) => db.pairs.find(({ id }) => pairId === id))
  const pairNow = pairsToday.find((pair) => checkIfCurrentTimeInSomePeriod(currentTime, pair.time))

  if (typeof pairNow === 'undefined') throw new ScheduleError('Зараз пара не йде')

  return getPairInfoByPairId(pairNow.id)
}
```

Рисунок 4.2 — Функція підбору поточної пари



```
export const getNearestPair = (group, currentDay, currentTime) => {
  const schedule = db.schedules.find((schedule) => schedule.groupId === group.id)
  const currentTimeInSeconds = formattedTimeToSeconds(currentTime)

  let dayIndex = currentDay
  const pairIdsToday = schedule.days[dayIndex]

  if (
    typeof pairIdsToday !== 'undefined' &&
    Array.isArray(pairIdsToday) &&
    pairIdsToday.length !== 0
  ) {
    const pairsToday = pairIdsToday.map((id) => db.pairs.find((pair) => pair.id === id))

    const firstPair = pairsToday[0]      You, 2 месяца назад • Finish getNearestPair logic …
    const lastPair = pairsToday[pairsToday.length - 1]

    const currentTimeSeconds = formattedTimeToSeconds(currentTime)
    const firstPairBeginSeconds = formattedTimeToSeconds(firstPair.time.split('-')[0])
    const lastPairBeginSeconds = formattedTimeToSeconds(lastPair.time.split('-')[0])

    // we are before pairs(currentTime < ) - return first pair of today
    if (currentTimeSeconds < firstPairBeginSeconds) {
      return getPairInfoByPairId(firstPair.id)
    }
    // we are after pairs - return first pair of nextDay
    else if (currentTimeSeconds >= lastPairBeginSeconds) {
      return getPairInfoByPairId(findNextDayFirstPairId(dayIndex, schedule))
    }
    // we are in time with pairs return pair after that
    else {
      const pair = pairsToday.find(({ time }) => checkIfCurrentTimeInSomePeriod(currentTime, time))

      // If we have a pair - return next pair
      if (typeof pair !== 'undefined') {
        return getPairInfoByPairId(pairIdsToday[pairIdsToday.indexOf(pair.id) + 1])
      }
      // If time is in gap -  return first pair, start time of that more than our time
      else {
        return getPairInfoByPairId(
          pairsToday.find(
            ({ time }) => currentTimeInSeconds < formattedTimeToSeconds(time.split('-')[0])
          ).id
        )
      }
    }
  } else {
    return getPairInfoByPairId(findNextDayOutOfStudyingWeekFirstPairId(dayIndex, schedule))
  }
}
```

Рисунок 4.3 — Функція підбору наступної пари

Далі бачимо деплой на Vercel, деплоймент налаштовано із використанням технології webhook (див. рис. 4.4):

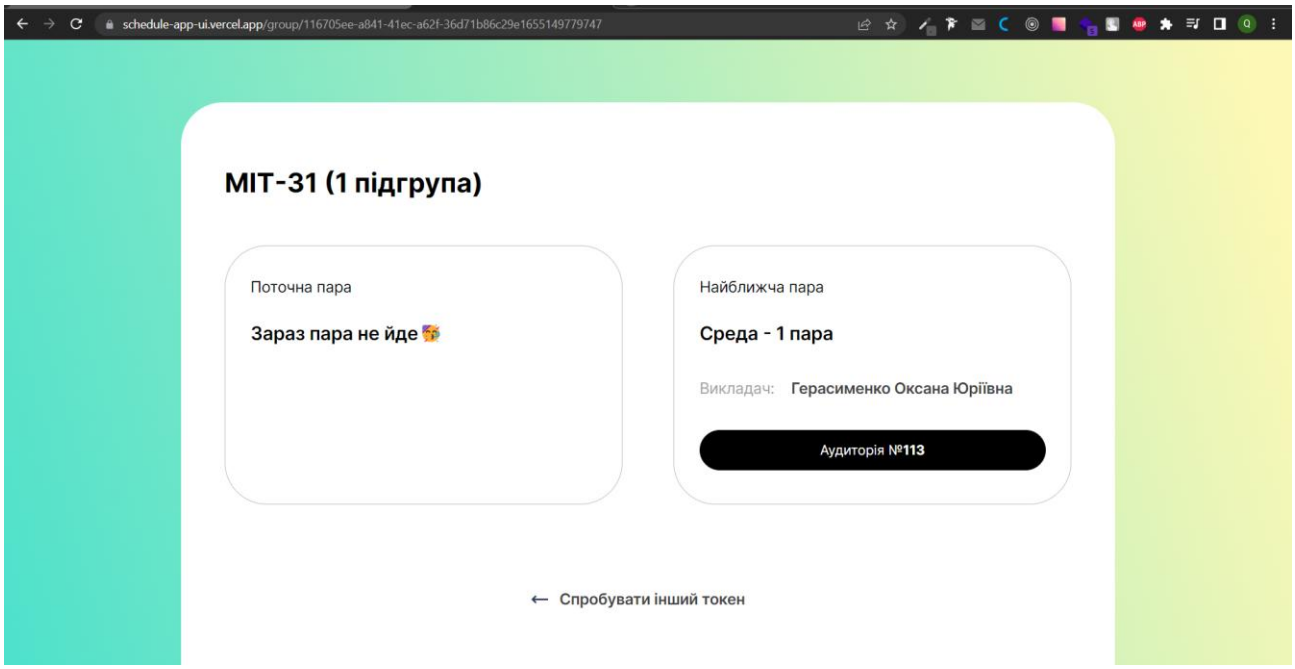


Рисунок 4.4 — Інтерфейс програми Scheduler

А також надам тести з бібліотеки JEST із поясненням (див. рис. 4.5):

Рисунок 4.5 — Пройдені тести функцій додатку

РОЗДІЛ 5. РОЗРОБКА API-СЕРВІСУ З НАБОРОМ НЕОБХІДНИХ CRUD ОПЕРАЦІЙ

5.1 РОЗРОБКА PYTHON ДОДАТКУ

Другий етап, база даних і CRUD, для максимально оперативної взаємодії хапаємо Redis DB, далі візьмемо python з фреймворком FLASK для забезпечення нас RESTful API (рис 5.1), врешті-решт, починаємо оголошувати сутності:

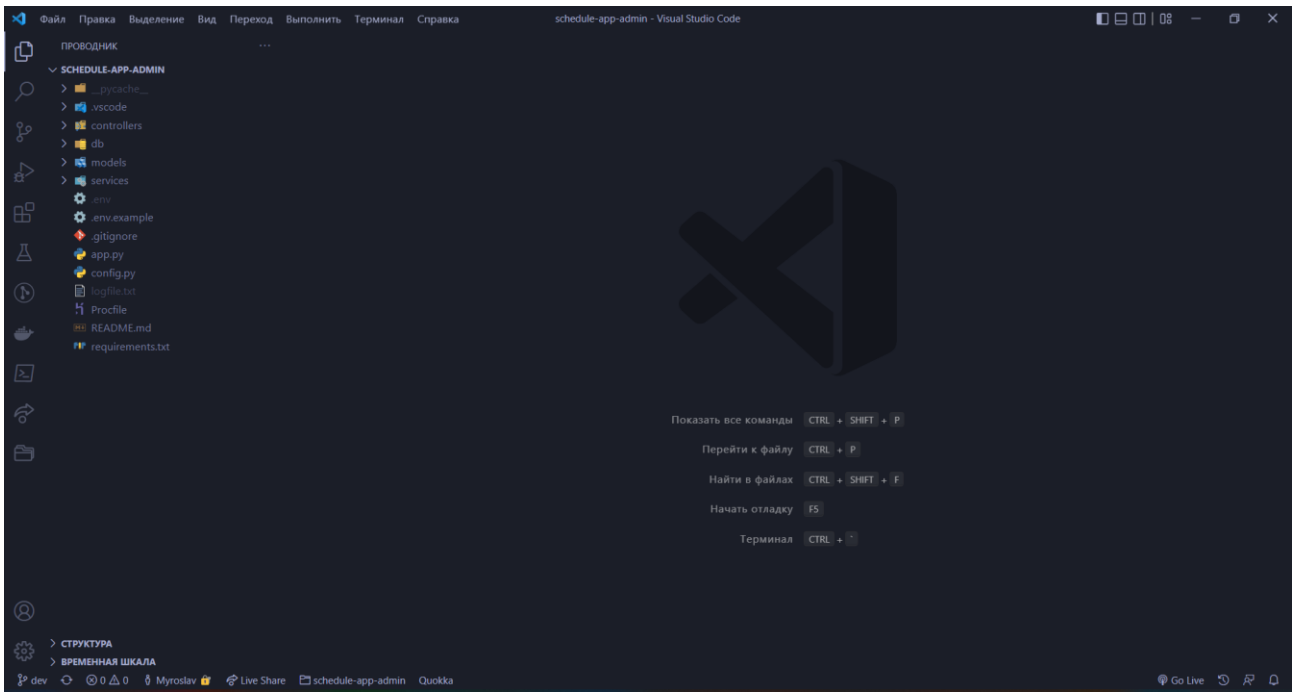


Рисунок 5.1 — Структура проекту schedule-app-admin

Буде наявно 6 сутностей:

- **Teacher** (Викладач, який має повне ім’я)
- **Group** (Група з назвою name і id запису у розкладі)
- **Pair** (Пара з певним часом і предметом для проведення)
- **Schedule** (Запис у розкладі за днем і кількістю пар)
- **Subject** (Певний предмет із назвою, поле name і id викладача)
- **Day** (День тижня з понеділка по неділю, має поле name)

Також дотримуючись правил нормалізації, базу було відформатовано відповідно до них. Правила нормалізації у таблицях - для ефективності та уникнення

надмірності даних. Основні правила нормалізації включають:

Перша нормальна форма (1NF): Усі атрибути (стовпці) в таблиці повинні містити тільки атомарні значення, тобто кожна комірка повинна містити тільки одне значення. Також має бути унікальний ідентифікатор (ключ) для кожного запису.

Друга нормальна форма (2NF): Має бути усунена часткова залежність атрибутів від складеного первинного ключа. Якщо атрибути залежать від тільки частини складеного ключа, вони мають бути винесені в окрему таблицю.

Третя нормальна форма (3NF): Має бути усунена транзитивна залежність атрибутів. Якщо атрибути залежать один від одного, вони мають бути розділені на окремі таблиці.

Четверта нормальна форма (4NF): Має бути усунена багатозначна залежність. Якщо є залежність між неключовими атрибутами через інші неключові атрибути, вони мають бути виділені в окремі таблиці.

П'ята нормальна форма (5NF): Має бути усунена залежність стосовно неключових атрибутів через виключно ключові атрибути. Якщо є залежності між атрибутами, де один атрибут може бути виведений з іншого через кілька шляхів, вони мають бути розділені на окремі таблиці.

Це основні правила нормалізації, але також існують більш просунуті форми нормалізації, як-от шоста нормальна форма (6NF) і вищі форми. Вибір правил нормалізації залежить від конкретної ситуації та вимог бази даних. Мета нормалізації - мінімізувати надмірність даних, забезпечити цілісність і полегшити виконання операцій з даними.

Тестуємо запити у програмі Postman (див. рис 5.3):

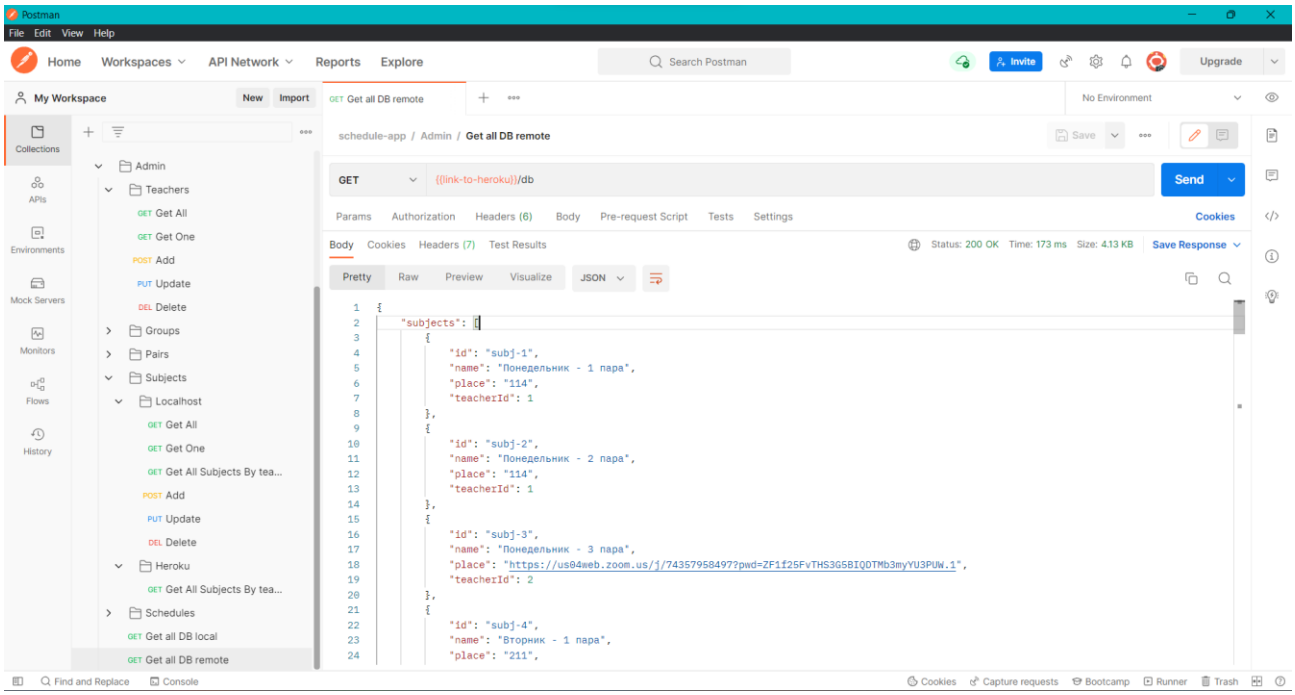


Рисунок 5.3 — Вікно програми Postman із усіма наявними запитами

Лінк на хероку приховано через конфіденційність, адже у базі будуть зберігатися записи про реальні пари групи на тиждень/місяць/рік.

5.2 ТРОХИ ПРО МІГРАЦІЮ З PYTHON НА C#

В подальшому нашу частину АПІ-додатку було мігровано на C#. І ось кілька переваг ASP.NET порівняно з Flask:

Мова програмування: ASP.NET використовує C#, яка є статично типізованою мовою програмування з багатими можливостями ООП. Вона забезпечує більш сувору типізацію і більш надійну перевірку помилок на етапі компіляції. У той час як Flask використовує Python, який є динамічно типізованою мовою з більш вільним синтаксисом.

Інтеграція з платформою: ASP.NET розроблений і підтримується компанією Microsoft, що означає наявність широкого спектра інструментів, бібліотек і ресурсів для розробників. ASP.NET також повністю інтегрований з екосистемою Microsoft, включно з IIS (Internet Information Services), SQL Server та іншими продуктами, що спрощує розгортання та інтеграцію в наявну інфраструктуру.

Масштабованість: ASP.NET пропонує різні можливості для масштабування додатків. Наприклад, ASP.NET підтримує розподілені обчислення за допомогою технології ASP.NET Core і можливості горизонтального масштабування з використанням технології Azure. Це особливо важливо для великих і високонавантажених проєктів.

Безпека: ASP.NET пропонує потужні механізми безпеки, включно з вбудованою автентифікацією та авторизацією, обробкою вхідних даних і захистом від атак. Він також інтегрується з Identity Framework, який надає функціональність управління користувачами, ролями та правами доступу.

Керування станом: ASP.NET надає різні механізми для керування станом застосунку, як-от сеанси, кешування та інші інструменти для зберігання й обміну даними між запитами. Це полегшує розробку додатків, яким потрібне збереження стану між запитами. Самий додаток написано за допомогою MVVM патерну (Model-View-ViewModel) - це архітектурний шаблон проектування, що використовується в розробці програмного забезпечення. Він є варіантом патерну MVC (Model-View-Controller), оптимізованим для створення користувацького інтерфейсу.патерн. Далі представлено програмний код одного з контролерів (див. рис 5.5):

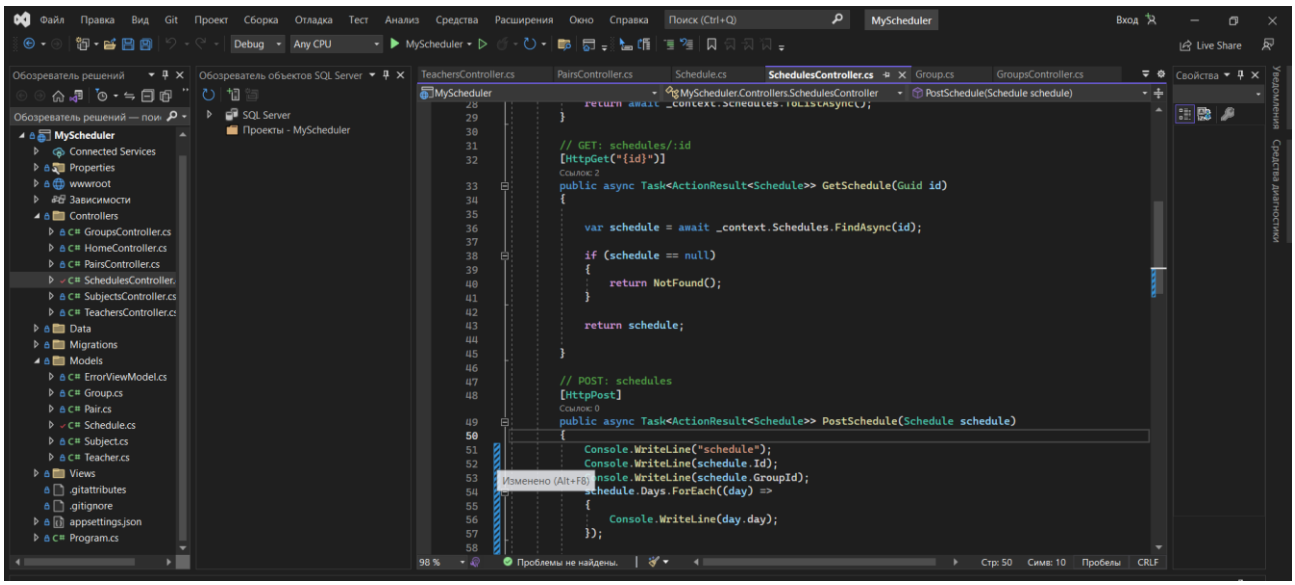


Рисунок 5.5 — Контролер розкладів schedules додатку Скедьюлер Admin

Далі лишилося повторити все те ж саме але вже для інших сутностей, тому було створено ще 4 контролери і в них теж вкладено по набору з п’яти операцій.

Через GET запит я встановлюю відповідність між клієнтом та API і повертаю або цілий набір всіх сутностей (/{entity-name}) або одну сутність за унікальним ідентифікатором (/{entity-name}/:id) - це операція Read. Наступний - POST запит за роутом (/{entity-name}). Тут також необхідно додати в body тіло запиту щоб кожне поле відповідало вимогам валідації, а головне - моделі з папки Models. Насамперед важливо не тримати в базі даних недостовірної або неконсистентної інформації (broken data). Таким чином представлена операція Create. Після йде PUT - запит на оновлення інформації. Тут також треба згадати про PATCH, тому що він теж є доволі розповсюдженим способом оновлення даних у ВЕБ.

Різниця між запитами PATCH та PUT полягає в їхньому призначенні та способі оновлення ресурсу на сервері: PUT, мета: замінити цільовий ресурс повністю на передані дані або створити його, якщо він не існує. Ідемпотентність: PUT-запити ідемпотентні, що означає, що повторне виконання одного й того самого запиту не повинно мати побічних ефектів. Якщо я відправляю PUT-запит на /{entity-name}/1, це може означати, що я повністю замінюю інформацію про сутність з ідентифікатором 1 переданими даними.

PATCH-запит частково змінює цільовий ресурс застосовуючи передані зміни тільки до відповідних полів ресурсу, не зачіпаючи інші. Ідемпотентність: PATCH-запити зазвичай не ідемпотентні, тобто повторне виконання одного й того самого запиту може мати різні побічні ефекти. Якщо я надсилаю PATCH-запит на /{entity-name}/1, це може означати, що я хочу оновити тільки певні поля інформації про сутність з ідентифікатором 1, не торкаючись інших полів.

В обох же випадках сервер повинен обробляти запити відповідним чином відповідно до їхнього призначення та логіки програми. Вибір між PATCH і PUT залежить від специфіки вашого застосунку та вимог до оновлення

ресурсів. Це і є моя операція Update.

І, наостанок, видалення. Тут доволі просто, адже достатньо в контролері прописати роут /{entity-name}/:id за типом запиту DELETE, а в запиті до БД створити відповідно операцію, як просто звертаючись на цей роут ми цілковито видаляємо цю сутність і всі зв’язані з нею сутності. Бо коли потрібно видалити сутність із бази даних, але в неї є пов'язані з нею інші сутності, такі як розклади у моєму прикладі, це називається каскадне видалення (cascade deletion) або каскадне видалення залежностей (cascade delete). Каскадне видалення передбачає автоматичне видалення пов'язаних сутностей, щоб підтримувати цілісність даних. У контексті баз даних це означає, що під час видалення групи, всі пов'язані з нею розклади також буде видалено. Це остання частина (Delete) з абревіатури CRUD. Також нижче додано схему оновленого тестування за допомогою додатку Postman (див. рис. 5.6):

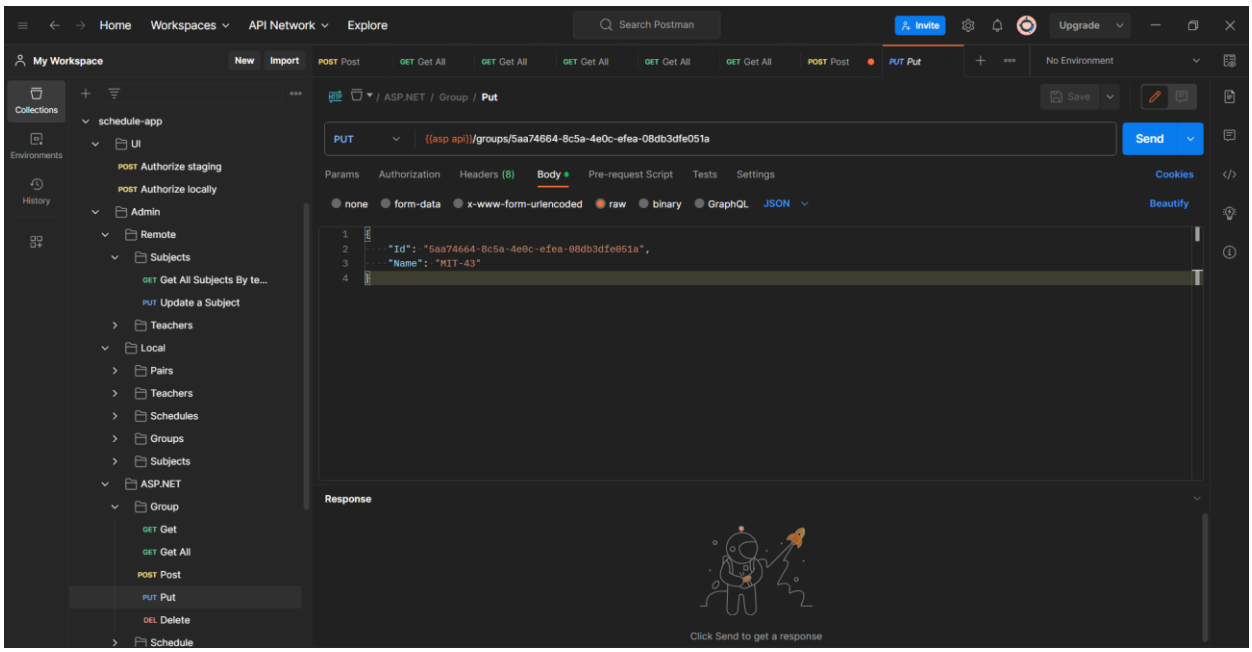


Рисунок 5.6 — Структура тест-запитів у додатку постман

Даний об’єм роботи було проведено щоб отримати на виході наступну структу JSON на ендпоінті для UI-додатку:

```
{
 "subjects": [
  {
   "id": "subj-1",
   "name": "Понедельник - 1 пара",
   "place": "https://us04web.zoom.us/j/74357958497?pwd=ZF1f25FvT”,
   "teacherId": 1
  }
 ],
 "teachers": [{ "id": 1, "name": "Миколайчук Роман Антонович" }],
 "groups": [{ "id": "djsaFnGDLk727uiogu3i710gDDA9GhdLIk", "name": "MIT-31
(1 підгрупа)" }],
  "schedules": [
  {
   "id": 1,
   "groupId": "djsaFnGDLk727uiogu3i710gDDA9GhdLIk",
   "days": [[1, 2, 3], [], [7, 8, 9], [10, 11, 12], [], [], []]
  }
 ],
 "pairs": [
  {
   "id": 1,
   "subjectId": "subj-1",
   "time": "09:00 - 10:20"
  }
 ],
```

```
  "days": [{ "id": 1, "name": "Понеділок" }]
}
```

РОЗДІЛ 6. РОЗРОБКА UI ЧАСТИНИ ДЛЯ АДМІНІСТРАТИВНОЇ СТОРОНИ СИСТЕМИ

Ось і дісталися до останнього сервісу, ним став Editor для наших дорогих викладачів. Суть сервісу полягає в зручному маніпулюванні розкладом та представленими в Розділі номер 5 сутностями. І тут на допомогу прийшов фреймворк Next.js зі своїми можливостями по роутингу, а в якості стейт менеджера постає Redux Saga, так як нам необхідно виконувати доволі багато асинхронних дій і redux-thunk, хоч і має таку можливість, але сага дає нам виграш у плані перфомансу, що при роботі з великим об’ємом даних є ледь не найголовнішою ознакою, яка вплине на UX в майбутньому (рис 6.1):

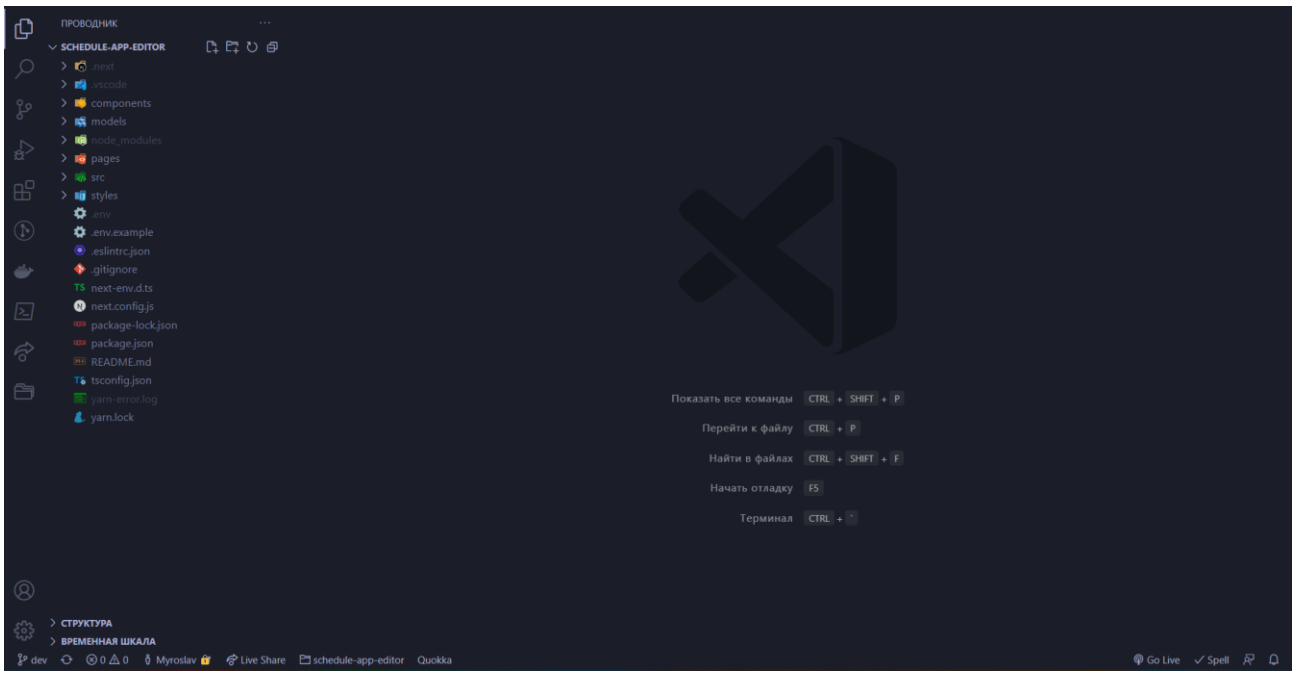


Рисунок 6.1 — Структура проекту schedule-app-editor

Запускаємо та вантажимо на Vercel (рис 6.2) Головна сторінка представляє собою селектор викладача, за яким буде здійснено запит до API:

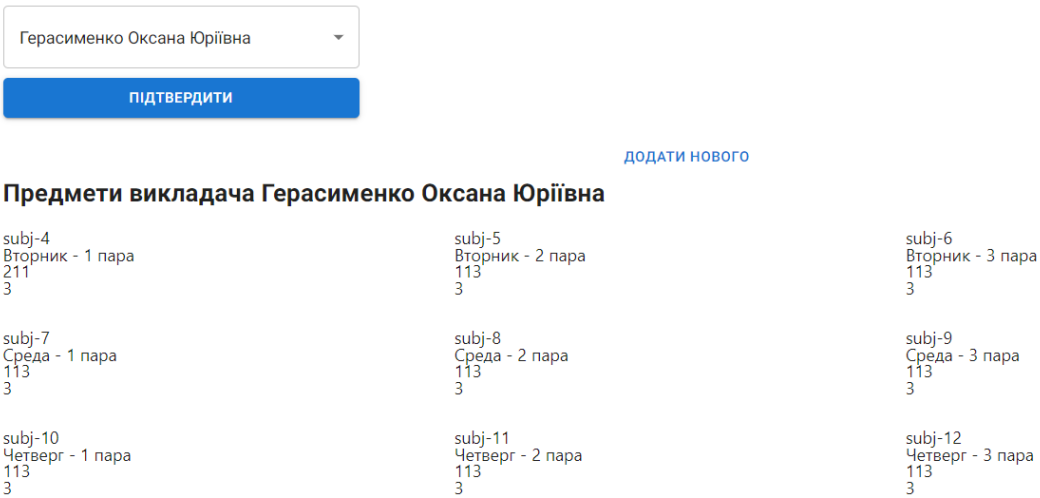


Рисунок 6.2 — Перша та не остаточна версія schedule-app-editor на Vercel

Після деякого проміжку часу інтерфейс програми було вдосконалено та перероблено за допомогою Figma на більш сучасний лад. Додано невелике оформлення, а також раціональніше розподілено місце на сторінці для зручності виконання операцій (див. рис. 6.3)

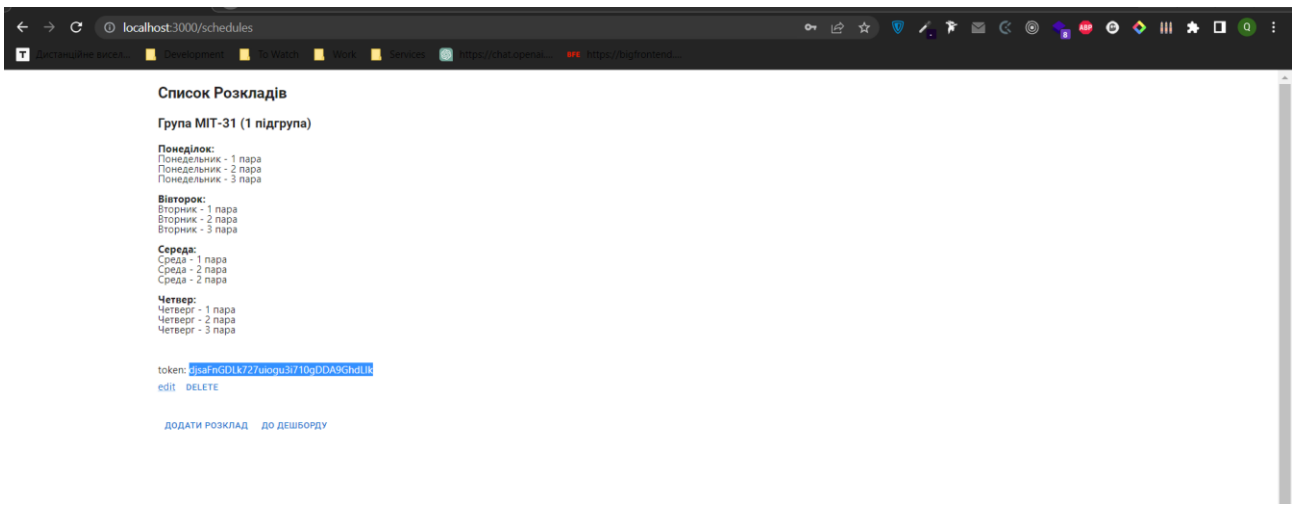


Рисунок 6.3 — Оптимізований дизайн програмного забезпечення Scheduler після другої ітерації

РОЗДІЛ 7. РЕФАКТОРИНГ КОДУ ТА ОПТИМІЗАЦІЯ СКЛАДНИХ ФУНКЦІЙ

Ні для кого не секрет, що будь-яке подвір'я мусить бути прибраним, а код програміста - це його подвір'я, яке необхідно прибирати. Тож, і мені довелося очистити деякі застарілі місця в коді.

Почнемо з першого сервісу, і тут одразу видно, що код, за допомогою якого ми дістаємо інформацію про пару за айді цієї пари проситься для винесення в окрему функцію (рис 7.1):

```
const getPairInfoByPairId = (pairId) => {
  const { subjectId } = db.pairs.find(({ id }) => id === pairId)
  const { name, place, teacherId } = db.subjects.find(({ id }) => id === subjectId)
  const teacherName = db.teachers.find(({ id }) => id === teacherId).name

  return { name, place, teacherName }
}
```

Рисунок 7.1 — Функція для отримання інформації про пару

Тему оптимізації хотілося б почати з введення до О великого. Нехай f, функція, яку потрібно оцінити, є дійсною або комплексною функцією, а g, функція порівняння, є дійсною функцією. Нехай обидві функції визначено на деякій необмеженій підмножині додатних дійсних чисел, і g(x) строго додатна для всіх достатньо великих значень x

$$f(x) = O(g(x)) \quad \text{as } x \to \infty. \tag{1}$$

якщо абсолютна величина від f(x) є щонайбільше додатною сталою величиною, кратною g(x) для всіх достатньо великих значень x. Тобто,

$$f(x) = O(g(x)), \tag{2}$$

якщо існує додатне дійсне число M і дійсне число x_0 таке, що

$$|f(x)| \leq Mg(x) \quad \text{for all } x \geq x_0. \tag{3}$$

У багатьох контекстах припущення, що нас цікавить темп зростання як змінна x прямує до нескінченності, залишається не висловленим, і пишуть простіше, що

$$f(x) = O(g(x)). \tag{4}$$

Нотація також може бути використана для опису поведінки f біля деякого дійсного числа a (часто, a=0): ми говоримо

$$f(x) = O\big(g(x)\big) \quad \text{as } x \to a. \tag{5}$$

Якщо існують додатні числа **δ** і M таким чином, що для всіх визначених x

з $$0 < |x - a| < \delta, \tag{6}$$

$$|f(x)| \le Mg(x). \tag{7}$$

Як g(x) вибирається строго додатною для таких значень x, обидва ці визначення можна об'єднати за допомогою граничного випередження:

$$f(x) = O\big(g(x)\big) \quad \text{as } x \to a, \tag{8}$$

якщо

$$\limsup_{x \to a} \frac{|f(x)|}{g(x)} < \infty. \tag{9}$$

І в обох цих визначеннях гранична точка a (незалежно від того, ∞ чи ні) є точкою кластеризації доменів f та g, тобто в кожній околиці точки a повинні мати нескінченно багато спільних точок. Більше того, як зазначено у статті про граничну нижчу та граничну вищу, у $\limsup_{x \to a}$(принаймні на розширеній дійсній числовій прямій) завжди існує.

У комп'ютерних науках поширене дещо більш обмежувальне визначення: нехай f та g є функціями від деякої необмеженої підмножини натуральних чисел до невід'ємних дійсних чисел; тоді

$$f(x) = O\big(g(x)\big). \tag{10}$$

якщо існують додатні цілі числа M та n_0 такі, що

$$f(n) \le Mg(n). \tag{11}$$

для всіх $n \ge n_0$ (див. рис 7.2):

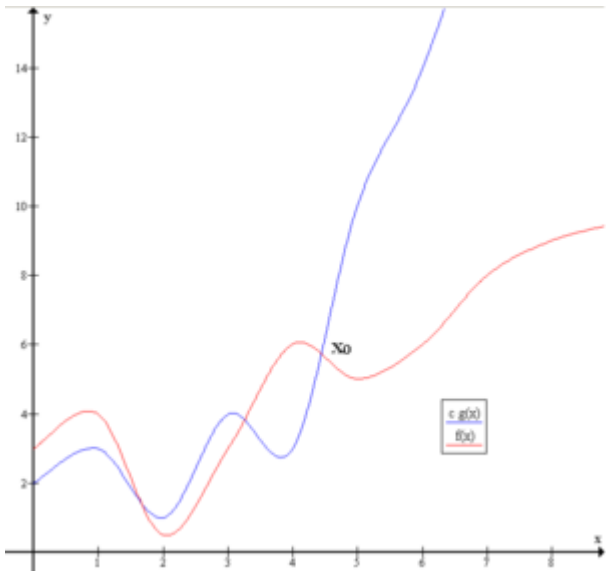


Рисунок 7.2 – Приклад нотації Big O

$f(x) \in O(g(x))$як існує M<0 (наприклад, M=1) і x_0 (наприклад, x_0=5) так, що $f(x) \leq Mg(x)$коли б то не було$x \geq x_0$.

Оптимізація (у математиці та інформатиці) - це задача знаходження екстремуму (мінімуму або максимуму) цільової функції в деякій області скінченновимірного векторного простору, обмеженої набором лінійних та/або нелінійних рівностей та/або нерівностей.

Теорію та методи розв'язання задачі оптимізації вивчає математичне програмування. Математичне програмування - це галузь математики, що розробляє теорію, чисельні методи розв'язання багатовимірних задач оптимізації з обмеженнями. Оптимізація - модифікація системи для поліпшення її ефективності. Система може бути одиночною комп'ютерною програмою, цифровим пристроєм, набором комп'ютерів або навіть цілою мережею.

Хоча метою оптимізації є отримання оптимальної системи, істинно оптимальна система в процесі оптимізації досягається далеко не завжди. Оптимізована система зазвичай є оптимальною тільки для однієї задачі або групи користувачів: десь може бути важливішим зменшення часу, необхідного програмі для виконання роботи, навіть ціною споживання більшого об'єму пам'яті; у додатках, де важливішою є пам'ять, можуть обирати повільніші

алгоритми з меншими запитами до пам'яті.

Ба більше, часто не існує універсального рішення (яке добре працює у всіх випадках), тому інженери використовують компромісні (англ. tradeoff) рішення для оптимізації тільки ключових параметрів. До того ж, зусилля, необхідні для досягнення повністю оптимальної програми, яку неможливо далі поліпшити, практично завжди перевищують вигоду, яка може бути від цього отримана, тому, як правило, процес оптимізації завершується до того, як досягається повна оптимальність. На щастя, у більшості випадків навіть при цьому досягаються помітні поліпшення.

Оптимізація має проводитися з обережністю. Тоні Хоар уперше вимовив, а Дональд Кнут згодом часто повторював відомий вислів: "Передчасна оптимізація - це корінь усіх бід". Дуже важливо мати для початку озвучений алгоритм і працюючий прототип.

Нотація Big O - математична нотація, що описує граничну поведінку функції, коли аргумент тяжіє до певного значення або нескінченності. Big O належить до сімейства позначень, винайдених Паулем Бахманом, Едмундом Ландау та іншими, які в сукупності називаються позначеннями Бахмана-Ландау або асимптотичними позначеннями. Літеру O Бахманн обрав для позначення Ordnung(порядок апроксимації).

Наступним кроком винесемо функцію, за допомогою якої ми знаходимо наступний день цього тижню (див. рис 7.3):

```
const findNextDayOutOfStudyingWeekFirstPairId = (dayIndex, schedule) => {
  dayIndex = (dayIndex + 1) % db.days.length

  while (
    typeof schedule.days[dayIndex] === 'undefined' ||
    typeof schedule.days[dayIndex][0] === 'undefined'
  ) {
    dayIndex = (dayIndex + 1) % db.days.length
  }

  return schedule.days[dayIndex][0]
}
```

Рисунок 7.3 — Функція для отримання наступного дня не на цьому тижні

Взагалі в інформатиці нотація big O використовується для класифікації алгоритмів відповідно до того, як їх час виконання або вимоги до простору зростають зі збільшенням розміру вхідного сигналу.[3] В аналітичній теорії чисел нотацію big O часто використовують для вираження обмеження різниці між арифметичною функцією та більш зрозумілою апроксимацією; відомим прикладом такої різниці є остаточний член в теоремі про просте число. Нотацію Big O також використовують у багатьох інших галузях для надання подібних оцінок.

Позначення Big O характеризує функції за темпами їхнього зростання: різні функції з однаковою асимптотичною швидкістю зростання можна представити, використовуючи одне й те саме позначення O. Літера O використовується тому, що темп зростання функції також називають порядком функції. Опис функції в термінах великої літери O зазвичай дає лише верхню межу швидкості росту функції.

Ну і завершальний крок у цьому додатку стане отримання першої пари завтрашнього дня (рис 7.4):

```
const findNextDayFirstPairId = (dayIndex, schedule) => {
  dayIndex = (dayIndex + 1) % schedule.days.length

  while (typeof schedule.days[dayIndex][0] === 'undefined') {
    dayIndex = (dayIndex + 1) % schedule.days.length
  }

  return schedule.days[dayIndex][0]
}
```

Рисунок 7.4 — Функція отримання першої пари завтрашнього дня

РОЗДІЛ 8. РОЗРОБКА АПІ НА ASP.NET ТА МІГРАЦІЯ З PYTHON

8.1 МІГРАЦІЯ ДАНИХ В СКБД MS SQL

Попередньо додаток schedule-app-admin було написано на Python Flask та Redis і це мало свої беззаперечні переваги:

Простота вивчення і використання: Python є однією з найпопулярніших і найпростіших у використанні мов програмування. Він має простий і зрозумілий синтаксис, що робить його ідеальним вибором для початківців програмістів. Flask, зі свого боку, є легковажним фреймворком, який також вирізняється простотою і мінімальністю, даючи змогу розробникам створювати веб-додатки з мінімальними витратами зусиль.

Гнучкість і модульність: Flask надає мінімалістичний набір інструментів, який дає змогу розробникам обирати та налаштовувати лише необхідні компоненти для свого проекту. Це дає можливість створювати гнучкі та модульні веб-додатки, які легко масштабувати та підтримувати.

Підтримка розширень: Flask має велике співтовариство та екосистему розширень, які забезпечують додаткові функціональні можливості та інтеграцію з іншими бібліотеками та інструментами. Наприклад, за допомогою розширень ви можете додати автентифікацію, базу даних, обробку форм, асинхронні запити та багато іншого, розширюючи функціональність вашого застосунку.

Швидкість розробки: Python і Flask дають змогу прискорити процес розробки завдяки своїй простоті та високій продуктивності. Flask пропонує багато вбудованих функцій і зручний маршрутизатор, що спрощує створення маршрутів і обробку HTTP-запитів. Python також відомий своєю ефективністю в написанні чистого і зрозумілого коду, що сприяє прискоренню розроблення та підтримуваності проєкту.

Широка підтримка: Python і Flask мають активне співтовариство розробників, готових допомогти і підтримувати один одного. Ви знайдете безліч

посібників, документації та ресурсів, які допоможуть вам розв'язати проблеми і навчитися нових речей. Крім того, Flask добре інтегрується з іншими інструментами та бібліотеками Python, що полегшує роботу.

А в зв’язці з Редіс ДБ ця зв’язка набуває ще більше плюсів для розробки веб-додатків. Ось лише деякі з них:

Швидкодія: Redis є швидкою і масштабованою системою зберігання даних, яку можна використовувати як кеш або сховище даних для вашого Flask-додатку. Він працює в оперативній пам'яті і має високу продуктивність завдяки своїй здатності обробляти дані в пам'яті, що робить його ідеальним вибором для завдань, які потребують швидкого доступу до даних.

Кешування: Redis надає можливість кешування даних у пам'яті, що дає змогу значно прискорити доступ до часто використовуваних даних. Ви можете використовувати Redis для кешування результатів складних запитів до бази даних або довгих обчислень, що знизить навантаження на сервер і поліпшить продуктивність вашого застосунку.

Сесії та зберігання стану: Redis може бути використаний для зберігання даних сесій користувачів у вашому Flask-додатку. Це дає змогу зберігати стан між запитами та забезпечувати автентифікацію й авторизацію користувачів. Використання Redis для зберігання сесій також забезпечує масштабованість, оскільки Redis може бути налаштований для роботи в кластері, забезпечуючи високу доступність і відмовостійкість (див. рис. 8.1) .

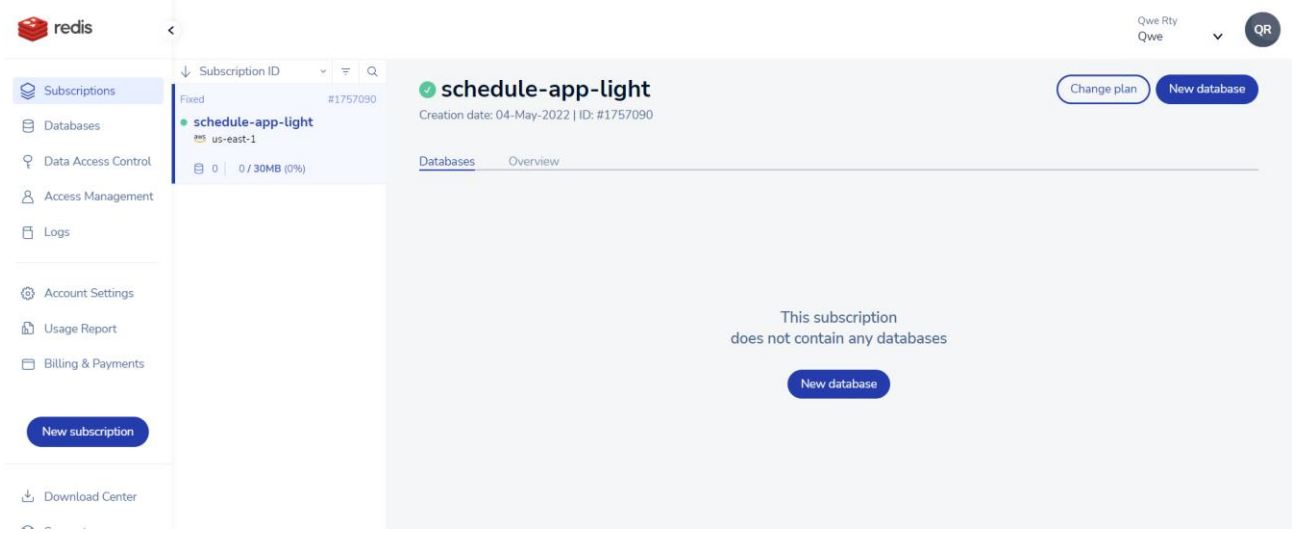


Рисунок 8.1 — Кластер Redis DB.

Хоча зв'язка Python, Flask і Redis має безліч переваг, вона також має деякі недоліки, про які варто згадати:

Складність конфігурації: Налаштування та управління Redis може бути складним завданням для новачків. Потрібно правильно конфігурувати і масштабувати Redis-сервер, встановити відповідні параметри і обробити питання безпеки. Неправильна конфігурація Redis може призвести до вразливостей безпеки або неправильного функціонування вашої програми.

Втрата даних: Redis за замовчуванням працює в оперативній пам'яті, що означає, що дані зберігаються тільки в пам'яті та можуть бути втрачені під час збою живлення або перезавантаження сервера. Хоча Redis пропонує можливість зберігати дані на диск з використанням механізму снапшотів і журналу транзакцій, це додає складність у налаштуванні та управлінні.

Складність масштабування: У разі необхідності масштабування Redis для обробки великого обсягу даних або високого навантаження, потрібна складніша конфігурація і розгортання Redis-кластера. Кластеризація Redis може зажадати додаткових зусиль і компетенцій для налаштування та управління кластером.

Додатковий шар складності: Впровадження Redis у зв'язку Python і Flask додає додатковий шар складності до проекту. Розробнику необхідно бути знайомим із синтаксисом Python, основами Flask і Redis-специфічними

функціями та операціями. Це може вимагати додаткового навчання і призвести до більш високого порогу входження для новачків.

Керування станом: Використання Redis для зберігання сесій або стану застосунку може бути складним для масштабування та оновлення під час розробки або обслуговування. Коректне управління станом у Redis може вимагати додаткових зусиль і уваги до деталей.

Важливо зазначити, що багато з цих недоліків можна подолати з правильною конфігурацією, плануванням і розумінням особливостей Redis. Більш досвідчені розробники можуть ефективно використовувати зв'язку Python

Тож через деякий час було вирішено мігрувати існуючий python application (див. рис. 8.2) на ASP.NET.

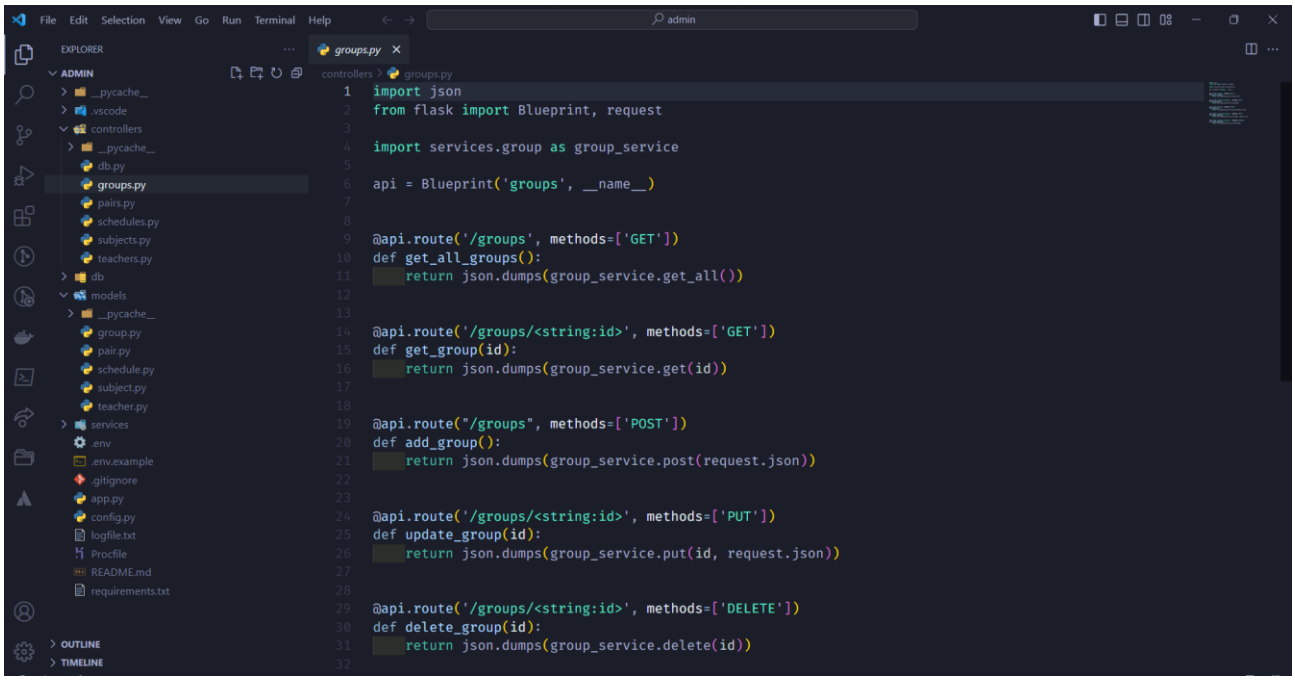


Рисунок 8.2 — Виконаний на мові Python schedule-app-admin

Асп (ASP.NET) і MSSQL (Microsoft SQL Server) є технологіями, які можуть запропонувати свої переваги, як порівняти з Python і Redis. Ось деякі з них: Переваги ASP.NET і MSSQL:

Інтеграція з Windows: ASP.NET і MSSQL розроблені Microsoft і мають глибоку інтеграцію з операційною системою Windows. Це означає, що я

нарешті зможу використовувати їх у середовищі Windows, повністю взаємодіючи з іншими продуктами та сервісами Microsoft.

Широка підтримка інструментів і ресурсів: ASP.NET і MSSQL мають велике співтовариство розробників і велику екосистему інструментів і ресурсів для розробки. Microsoft надає різноманітні інструменти, IDE (інтегроване середовище розробки) і бібліотеки, які спрощують розробку і підтримку додатків.

Потужні можливості розроблення: ASP.NET надає широкий спектр можливостей для розроблення веб-додатків, як-от інтеграція з Windows-службами, авторизація та автентифікація, масштабування тощо. MSSQL, з іншого боку, є потужною і розширюваною реляційною базою даних, що пропонує широкий спектр можливостей для управління даними.

Промисловий стандарт і надійність: ASP.NET і MSSQL є промисловими стандартами, які отримали широке визнання і використання в корпоративних середовищах. Вони добре підтримуються і мають високий ступінь надійності, що важливо для критично важливих систем і проектів.

Для старту було використано шаблон проєктування MVC накладений на реалізацію під C# та ASP.NET (див. рис. 8.3)

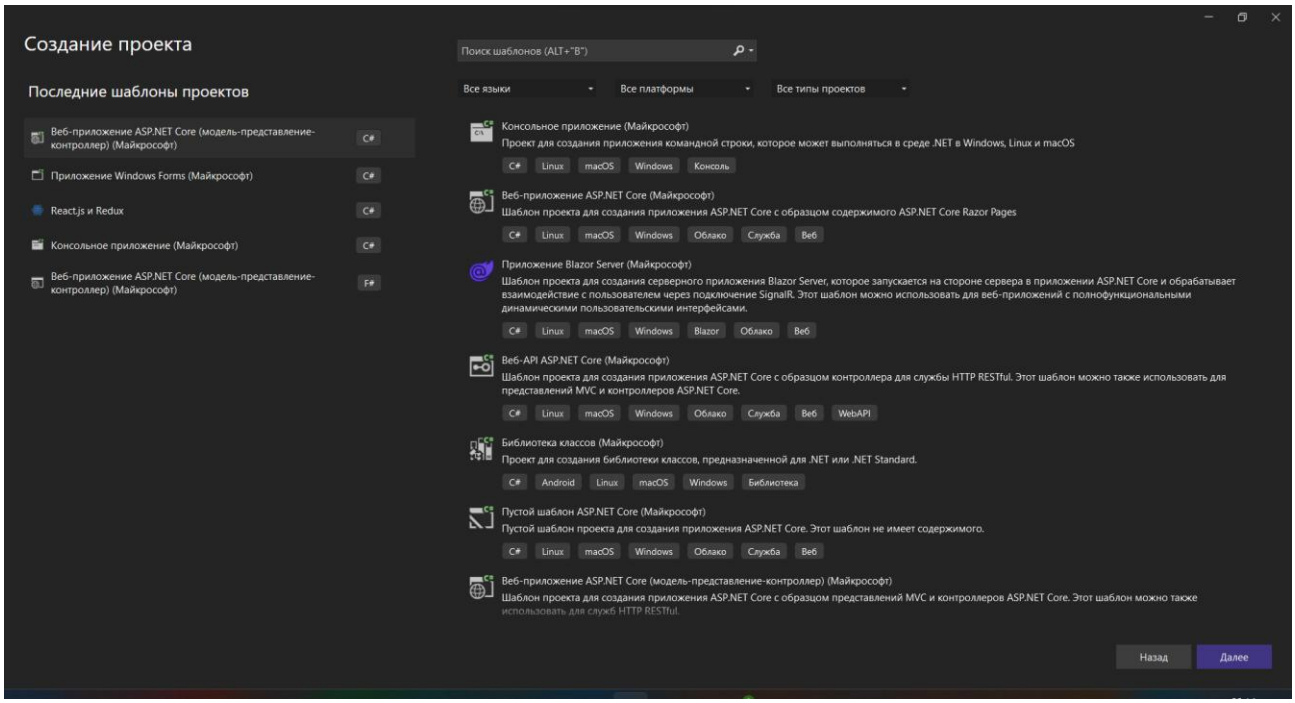


Рисунок 8.3 — Шаблон проєктів на ASP.NET Core

8.2 ОКРЕМЕ АПІ ДЛЯ ФОРМУВАННЯ ЗВЕРТАЛЬНОГО ВІДМІНКУ

Якщо добре пригадати, то на головній сторінці дашборду представлено повне ім’я викладача, його хотілося б провідміняти та зробити відповідне звертання за прізвищем, ім’ям та по-батькові. В інтернеті було знайдено цікавий сайт, https://slovnyk.ua. Там є маршрут, який може нам надати необхідний функціонал, а саме трансформацію ім’я та прізвища в звертальну форму. Тому тут було вирішено написати функцію для того, щоб парсити сторінку по запиту на ім’я конкретного авторизованого викладача та відображати його (див. рис. 8.4).

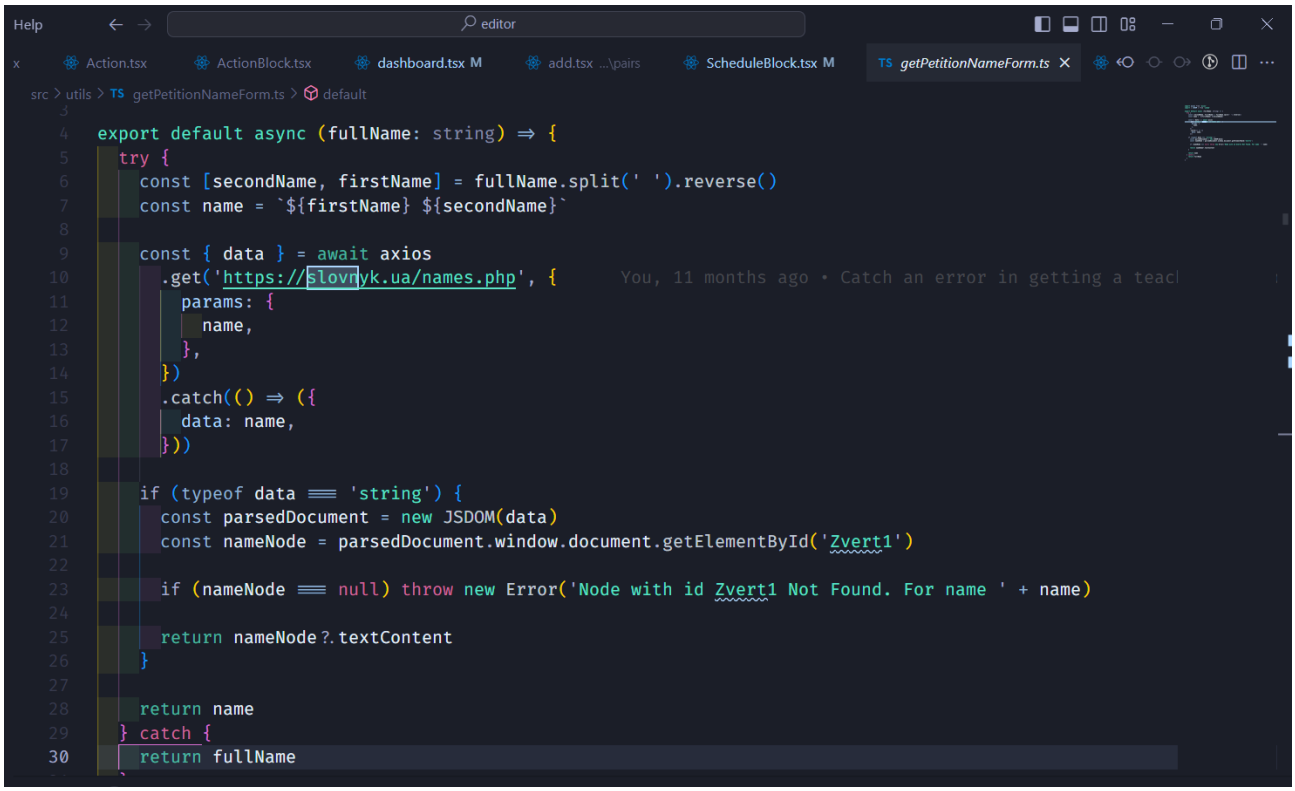


Рисунок 8.4 — Функція трансформації імені в звертальний відмінок

ВИСНОВКИ

Завершено роботу над масштабним додатком Scheduler, цей шлях був дійсно грандіозним і в завершенні хочеться зробити висновок про зроблену роботу: Було досліджено цілу низку наведених у розділах технологій, створено та оптимізовано алгоритм підбору пар, налаштовано з’єднання сервісів по протоколу HTTP, завантажено 3 сервіси на різні хостинги, підключено базу даних зі збереженими сутностями, а також протестовано та перевірено студентами групи МІТ-31 МІТ-41.

Schedule-app-ui - суто студентський додаток для перегляду поточної та наступної пар. Schedule-app-admin - API, що надає нам ендпоінти на доступ до розкладу. Schedule-app-editor - зручний викладацький інтерфейс для роботи з API адмін частини. Завершивши аналіз найкращих практик веб-розробки та використавши певну структуру на основі рекомендованого темплейту T3-Стек було побудовано два інтерфейсі систем Schedule-app-ui & Schedule-app-editor.

ПЕРЕЛІК ПОСИЛАНЬ

1. Eloquent JavaScript 3rd edition (2018) Marijn Haverbeke *(книга один автор)*
2. Patterns for scalable JavaScript-applications (2011) Eddie Osmani *(книга один автор)*
3. "You Don't Know JS" book series (2019-2022) Kyle Simpson *(книга один автор)*
4. Computer Science Distilled. Learn the art of solving computational problems (2018) Wladston Ferreira Filho *(книга один автор)*
5. Clean Code: A Handbook of Agile Software Craftsmanship 1st Edition (2018) Robert C. Martin *(книга один автор)*
6. Clean Architecture: A Craftsman's Guide to Software Structure and Design 1st Edition (2018) Robert C. Martin *(книга один автор)*
7. Code Complete (1993) by Steve McConnell *(книга один автор)*
8. Introduction to Algorithms, 3rd Edition (The MIT Press) 3rd Edition (2009) by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein *(книга чотири автори)*
9. Web Scalability for Startup Engineers 1st Edition (2015) Artur Ejsmont *(книга один автор)*
10. Refactoring: Improving the Design of Existing Code 1st Edition (1999) Martin Fowler *(книга один автор)*
11. Design Patterns: Elements of Reusable Object-Oriented Software 1st Edition, Kindle Edition by Gamma Erich (Author), Helm Richard (Author), Johnson Ralph (Author), Vlissides John (Author), Grady Booch (Foreword) *(книга один автор)*
12. https://refactoring.guru/ru/design-patterns *(електронний підручник один автор)*
13. https://ru.reactjs.org/docs/getting-started.html *(документація)*
14. https://www.typescriptlang.org/ *(документація)*

15. https://blog.mariano.io/common-css-mistakes-and-how-to-fix-them-8ee0f5e88d64#.ucpni71wf *(блог один автор)*
16. https://github.com/trekhleb/javascript-algorithms *(репозиторій один автор)*
17. https://github.com/you-dont-need/You-Dont-Need-JavaScript *(книга один автор)*
18. https://habr.com/ru/company/mvideo/blog/597653/ *(блог один автор)*
19. https://developer.mozilla.org/ru/docs/Learn/JavaScript/Client-side_web_APIs/Introduction *(документація)*
20. https://habr.com/ru/company/dcmiran/blog/521718/ *(блог один автор)*
21. https://dev.to/andrewbaisden/moving-from-javascript-to-typescript-40ac *(блог один автор)*

ДОДАТОК

Об’єкт дослідження: способи оптимізації створення та редагування розкладу занять, технології Next.js; Vercel; Flask; Redis; Redis Cloud; Redis CLI; Docker; протокол HTTP/HTTPS.

Мета роботи (проекту): розробка веб-додатку для зручного перегляду, зміни, видалення та створення учбового розкладу занять, та полегшення цього циклу процесів як для учнів, так і для викладачів.

Методика дослідження: комплексне використання апаратних та програмних можливостей технологій Веб-розробки та протоколу HTTPS, а також програмного забезпечення Vercel, Heroku, Docker та Postman і технології Webhook.

В роботі запропоновано та практично реалізовано: програмна частина інформаційної системи для встановлення поточної та найближчої пари з розкладом за токеном групи, створення записів та занесення їх до бази даних створення та обробка запитів використовуючи протокол HTTP виконано в повному обсязі. За час випробувань кількома студентам групи МІТ-31 при виконанні представлених вище функцій, дослідницький зразок виконує їх всі.

Посилання на GitHub:

UI - https://github.com/Miroslavetsh/schedule-app-ui

Admin - https://github.com/Miroslavetsh/schedule-app-admin

Editor - https://github.com/Miroslavetsh/schedule-app-editor

Посилання на Vercel:

UI - https://schedule-app-ui.vercel.app

Editor - https://schedule-app-editor.vercel.app