Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Курсова робота

освітній ступінь – бакалавр

на тему: «**РОЗРОБКА ВЕБ-ЗАСТОСУНКУ ДЛЯ СКЛАДАННЯ НАВЧАЛЬНОГО РОЗКЛАДУ**»

Виконав: студент 3-го року навчання,

Спеціальності
122 Комп’ютерні науки

Орел Даниїл Миколайович

Керівник Бабич Т.А., _____
магістр комп’ютерних наук, асистент
«____» _____ 20____ р.

Київ – 2021

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Освітній ступінь бакалавр

Спеціальність 122 Комп’ютерні науки

Освітня програма бакалавр

ЗАТВЕРДЖУЮ

Завідувач кафедри інформатики

Гороховський С. С.

“____” _____20___року

ЗАВДАННЯ

ДЛЯ КУРСОВОЇ РОБОТИ СТУДЕНТУ

Орлу Даниїлу Миколайовичу

1. Тема роботи Розробка веб-застосунку для складання навчального розкладу, керівник роботи Бабич Трохим Анатолійович, магістр комп’ютерних наук, асистент

2. Строк подання студентом роботи 17.05.2021

3. Анотація

Вступ

Розділ 1. Дослідження та аналіз предметної області

1. Опис предметної області
2. Аналіз баз даних
3. Аналіз серверної реалізації рішень
4. Аналіз клієнтської реалізації рішень

Розділ 2. Проектування та розробка системи

1. Проектування бази даних
2. Розробка клієнт-серверної частини

Висновки

Список використаних джерел

Додатки

ГРАФІК ПІДГОТОВКИ КУРСОВОЇ РОБОТИ ДО ЗАХИСТУ

№ з/п	ПЕРЕЛІК РОБІТ	Термін виконання	Дата ознайомлення наукового керівника	Підпис наукового керівника	Примітки
1.	Вибір теми, затвердження її на засіданні кафедри та закріплення наукового керівника Узгодження календарного графіка підготовки кваліфікаційної роботи. Ознайомлення студента з критеріями оцінювання кваліфікаційної роботи (п. 8.5).	21 жовтня			
2.	Вивчення джерел літератури, матеріалів архівів, періодичних видань, збір та узагальнення фактів, даних	30 жовтня – 14 листопада			
3.	Складання плану каліф. роботи та узгодження з науковим керівником	16 листопада			
4.	Написання розділів роботи *або* Постановка експерименту, аналіз отриманих результатів наукового дослідження	20 листопада – 10 березня			
5.	Проміжний контроль виконання роботи	17 лютого			
6.	Написання кваліфікаційної роботи в цілому, ознайомлення з її першим варіантом наукового керівника	4 січня – 27 березня			
	Розділ 1 (постановка проблеми, теоретичні основи, огляд літературних джерел)	13 квітня			
	Розділ 2 (аналітично-дослідницька частина) (експериментальна частина для природничих і біологічних наук)	29 квітня			
	Розділ 3 (проектно-рекомендаційна частина) (аналіз результатів експерименту для природничих і біологічних наук)	4 травня			
7.	Повне завершення написання кваліфікаційної роботи, оформлення її згідно з вимогами й подання на відгук науковому керівнику	13 квітня – 3 травня			
8.	Подання кваліфікаційної роботи для перевірки письмових робіт студентів НаУКМА на відповідність вимогам академічної доброчесності,	17 травня			
9.	Публічний захист кваліфікаційної роботи перед екзаменаційною комісією	згідно з розкладом роботи ЕК			

Графік узгоджено «21» жовтня 2020р.

Науковий керівник Бабич Трохим Анатолійович

Виконавець курсової роботи Орел Даниїл Миколайович

Зміст

Анотація **8**

Вступ **9**

1.1. Опис предметної області 10

1.2. Аналіз баз даних 10

1.3. Аналіз серверної реалізації рішень 14

1.4. Аналіз клієнтської реалізації рішень 17

Розділ 2. Проектування та розробка системи **22**

2.1. Проектування бази даних 22

2.2 Розробка клієнт-серверної частини 29

Висновки **42**

Список використаних джерел **43**

Додатки **46**

Додаток А 46

Додаток Б 47

Додаток В 48

Додаток Г 49

Додаток Д 50

Додаток Е 51

Додаток Є 52

Додаток Ж 53

Додаток З 54

Додаток И 55

Додаток І 56

Додаток Ї 57

Додаток К 58

Додаток Л 59

Додаток М 60

Додаток Н 61

Анотація

Дана робота присвячена розробці веб-застосунку, який надає можливість методистам факультету інформатики, університету “Києво-Могилянської академії”, ефективно створювати та підтримувати навчальний розклад.

У роботі описується створення такої системи, включно з архітектурою застосунку, ER-моделлю бази даних, описом використаних технологій, оцінкою їх переваг і доцільності.

У висновках було проведено аналіз розробленого рішення, запропоновано певні зміни та вдосконалення до побудованої системи.

Вступ

Необхідність розкладу у наш час є очевидною для усіх структур, які мають певну систематизацію: для компаній, для великих та маленьких організацій, в тому числі й для вищих навчальних закладів. Розклад - першочергово слугує для структуризації та часової оптимізації одночасно для багатьох людей, які об’єднуються в одну структуру та в них є необхідність керуванням часу.

У роботі було розглянуто вузьку предметну область - університет “Києво-Могилянської академії”, факультет інформатики, та було виявлено таку проблему - методистам університету складно справитися з задачею складання розкладу.

Існує багато обмежень, з якими вони зіштовхуються, а саме: накладання пар, вимоги викладачів щодо проведення пар, тощо. Побудоване рішення допоможе методистам боротись з вище названими проблемами.

Платформа слугує для зручної підтримки навчального розкладу, вона є необхідною для методистів не тільки факультету інформатики, а й в цілому для університету.

Розділ 1. Дослідження та аналіз предметної області

1.1.Опис предметної області

Потреба у наявності зручного середовища для керування, підтримкою навчального розкладу завжди була актуальною для методистів закладів вищої освіти, в тому числі й для методистів НаУКМА. Вони щоразу зіштовхуються з несистематичним та неавтоматизованим процесом зі складання навчального розкладу, що значно ускладнює цю задачу.

В основі цієї задачі лежить математична проблема - задача виконання обмежень. Нехай існує певна сукупність об’єктів, стан яких має задовольняти певні обмеження. Формально, цю задачу можна представити у вигляді трійки <X, D, C>, де X - множинна змінних, D - область значень, а C - множина обмежень. Кожне обмеження у свою чергу є парою <t, R>, де t - кортеж, що складається з n змінних, а R - n-місне відношення D.

Оцінка змінної - функція, що відображає множину змінних на область значень D. Це можна записати у такій формі: f: X → D.

Розв’язком цієї задачі буде оцінка, що задовольняє всім обмеженням, що присутні у сукупності об'єктів.

У цій курсовій роботі не було розглянуто вирішення такого типу задач у контексті навчального розкладу. Для початку, було обрано вирішити першу проблему та розробити автоматизовану систему для керуванням розкладу у факультеті інформатики.

1.2. Аналіз баз даних

Перед проектуванням архітектури всього застосунку була дилема між обранням бази даних, яку систему управління базами даних слід обрати для застосунку: з реляційною чи нереляційною структурою.

Було проведено аналіз обидвох типів структур, і виявлено наведені нижче особливості.

1.2.1. Аналіз SQL баз даних

Основними перевагами [2] реляційних баз даних є:

- Висока швидкість: Запити з реляційної структури бази даних можна використовувати для отримання великої кількості записів.
- “Вертикальне” масштабування: при збільшенні об’єму інформації у нашій базі, швидкість запитів залишається й досі на високому рівні.
- Чітка визначеність стандартів. Усі реляційні бази даних використовують структуровану мову запитів - SQL, що слугує для створення запитів.
- Реляційні бази даних є сильно типізованими, використовують великий спектр різноманітних примітивних типів даних.
- Висока надійність: Більшість з реляційних баз даних підтримують принцип ACID - властивість транзакцій запитів.
- Велика підтримка: реляційні бази даних існують вже досить тривалий час, навколо них побудовано багато

Головними недоліками [2] цього типу структур баз даних є:

- Реалізація залежить від платформи, яка хостить базу даних.
- Запити з реаляціями може писати вже досвідчений користувач або програміст, які знає всі тонкощі роботи з мовою.
- Побудова правильної архітектури бази даних вимагає дуже глибокого розуміння не тільки предметної області, а й теоретичних засад самої мови SQL.

1.2.2. Аналіз NoSQL баз даних

Основними особливостями [1] нереляційних баз даних є:

- “Горизонтальне” масштабування: у будь-якому документі, в будь-який момент часу є можливість додати нове поле, без внесення змін до всіх екземплярів певної сутності у базі даних.
- Слабка типізація: у моделях можна легко змінювати типи даних без застосування значних змін у архітектурі бази даних.
- Динамічні структури даних. Нереляційні бази даних пропонують декілька варіантів, у якій формі ми можемо зберігати наші дані: у документах, у графовій структурі, орієнтовано по колонках, за допомогою пар “ключ-значення”.

Головними недоліками [1] цього типу структур баз даних є:

- Відсутність всіх можливих обмежень до атрибутів бази, а саме: унікальність, перевірка на порожні значення, невизначеність зовнішніх ключів та операцій над ними.

1.2.3. Вимоги до бази даних у застосунку

На основі предметної області та для вирішення поставленої задачі, було вирішено обрати реляційну базу даних, оскільки:

- Застосунку важлива висока швидкість. Застосунок не передбачає “горизонтального” масштабування, орієнтується на “вертикальне”.
- Потрібно реалізувати високу надійність. У предметній області атрибути сутностей зв’язків між ними потребують чітких обмежень та чітко визначених типів даних.
- Важлива підтримка транзакцій.

Визначившись з структурою бази даних, тепер можемо перейти до вибору системи управління базами даних (далі СУБД).

Світ SQL містить багато різних СУБД, які мають свої характерні особливості поміж інших. Проте вибір дуже звузився через такі ключові фактори:

- Надійність та частота оновлень. СУБД повинна йти у крок за новими технологіями, розробками, алгоритмами.
- Велика підтримка з боку громади IT спеціалістів. Цей фактор дозволяє вивчати більш детально та глибинно розглядати обрану СУБД, отримати якомога більший спектр прикладів та застосувань. Також великі корпорації вибудовують інфраструктуру навколо цих систем та публікують цікаві теоретичні, наукові статті.
- Гарна документація.

Після фільтрації СУБД по описаним вище чинникам, було звужено вибір до PostgreSQL[6] та MySQL [5], але в результаті було вирішено саме PostgreSQL. Деталі описані у наступному розділі.

1.2.4. Ключові переваги обраної системи управління базами даних PostgreSQL

Перш за все, він дозволяє використовувати [6] об’єктно-реляційні моделі зі складними структурами та широким спектром вбудованих, примітивних типів даних.

По-друге, PostgreSQL [6] прагне відповідати стандарту ANSI-SQL:2008, який в свою чергу відповідає вимогам ACID (атомарність, узгодженість, ізольованість та надійність) і відомий своєю транзакційною цілісністю. Велика кількість обмежень над первинними та зовнішніми ключами, обмеження на унікальність, ненульові значення, валідація атрибутів й багато інших функцій та можливостей дають впевненість, що база міститиме тільки коректні дані.

1.3. Аналіз серверної реалізації рішень

У цій секції буде розглянутий аналіз серверної реалізації застосунку: пошук найоптимальніших архітектурних рішень, визначення зі стеком для написання програми.

1.3.1. Аналіз бекенд рішень. Пошук найоптимальнішого середовища розробки

Перш за все, побудуємо вимоги до бекенд рішення, які обов’язково повинні бути у нашій предметній області:

- Швидкість.
- Надійність: всі використовувані дані у ході виконання програми повинні бути строго типізовані. Потрібне використання примітивних типів даних, що могли б мінімізувати використання пам’яті у застосунку.
- Об’єктно-орієнтованість. Сутності, які представлені у предметній області легко співставляються з реальним життям.

Відфільтрувавши мови за описаними вище критеріями, можна одразу виокремити лише одну мова програмування, яка надає таку підтримку, а саме - Java [3][4].

Вона також є дуже простою у вивченні. [3][4] Простоста суміщається одразу з популярністю мови, з її широким спектром використання та з крос-платформеністю.

1.3.2. Ключові особливості Java фреймворку - Spring Boot

Було проведено аналіз переваг та недоліків цього фреймворку [11], які підтвердились у подальшій розробці застосунку.

До переваг фреймворку [11] можна віднести:

- Уникнення ручної роботи з написання повторюваного коду за допомогою анотацій.
- Проста інтеграція Spring Boot з екосистемою Spring. Вона включає у себе повний набір утиліт для роботи з веб застосунками, такі як: Spring Data, Spring Security, Spring ORM та Spring JDBC.
- Легке підключення до систем управління базами даних та черг, таких як: PostgreSQL, MySQL, MongoDB, Rabbit MQ та багатьох інших.

Його ж недоліками [11] виявились такі фактори:

- Повільний цикл розробки: складно працювати у середовищі, у якому важко відслідковувати помилки (не виключенням є й NullPointerException-и, які є доволі частим явищем у цій мові - їх дуже складно запобігати).
- Побудова проекту займає дуже багато часу, особливо зі збільшенням кількості залежностей Spring Boot.
- Відсутність технології “Hot-Reload”. Це більше пов’язано з особливостями самої мови, Java, оскільки компілятор повинен завжди компілювати увесь проект перед початком програми.

Незважаючи на недоліки Spring Boot та й Java в цілому, швидкість розробки балансується [11] лаконічністю і простотою у написанні базових, та доволі монотонних функцій за допомогою допоміжних бібліотек, що входять у пакет Spring.

1.3.3. Комунікація між сервісами

У роботі проведений аналіз поміж різних способів комунікації [5] між сервісами, а саме:

- REST

- SOAP
- GraphQL

Усі вони - підходи до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів.

Який з них найоптимальніший? [5] Будемо йти методом від супротивного - які підходи точно не спрацюють при проектуванні застосунку:

SOAP: Використовує стандарт [5] XML для комунікації. Саме тому це призводить до надмірної кількості даних, що передаються по мережі як у запитах, так і у відповідях. Саме ця проблема може створити надто велике навантаження на мережу та вплинути на продуктивність програми. В застосунку ця проблема є критичною, оскільки по мережі передаватиметься великий об’єм даних про розклад всього факультету, а в подальшому й університету, що може значно уповільнити роботу сервісу.

GraphQL: Є доволі новим підходом [5] для проектування, тому там відсутні деякі важливі функції, а саме: механізм кешування запитів, лімітування кількості запитів за певний проміжок часу. Також присутня й складність у реалізації самих запитів.

REST: Важливим недоліком [5] є надмірне або недостатнє отримання даних з запиту. Малий набір типів типів даних, а відповідно слабка типізація, що може спричинити проблеми у спілкуванні між програмами, навіть тими, що підтримують REST API.

Проте для реалізації застосунку, найбільш зручним у використанні та найбільш ефективним виявився REST API [5], за допомогою якого можна швидко та ефективно будувати сервіс, з можливість швидкої передачі даних по мережі.

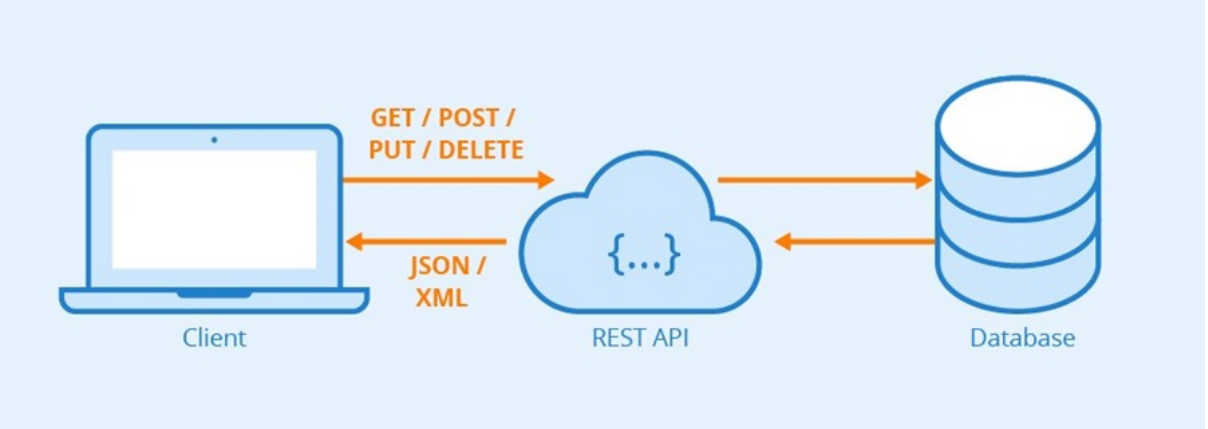


Рис 1.1. - REST API. Приклад комунікації на рівні сервісів

1.4. Аналіз клієнтської реалізації рішень

У цій секції буде розглянутий аналіз клієнтської реалізації застосунку: пошук найоптимальніших архітектурних рішень, визначення зі стеком для написання програми.

1.4.1. Пошук найоптимальнішого архітектурного рішення

Для побудови великих повноцінних веб-застосунків та продуктів використовуються три найбільш популярні бібліотеки, що дозволяють створювати інтерактивні інтерфейси для користувачів, а саме: React.js, Angular.js, Vue.js.

Розглянемо більш детально кожен з них, та вирішимо, який найбільш підпадає під наші потреби.

1.4.1.1. Огляд React.js

Його ключовими перевагами [9] [10] є:

- Високий рівень гнучкості.
- Віртуальна DOM (об'єктна модель документа), що дозволяє впорядковувати документи у форматах HTML, XHTML або XML у

дерево, з якого веб-браузери є більш прийнятним при аналізі різних елементів веб-програми.

- Легко працює з високим навантаженням.
- Бібліотека JavaScript з відкритим вихідним кодом, який отримує багато щоденних оновлень та вдосконалень відповідно до внесків розробників з усього світу.
- Підтримка сучасних стандартів мови JavaScript та найбільша розбудована екосистема для веб застосунків.

З недоліків [9] [10] можна виокремити такі пункти:

- Необхідно багато часу, щоб якісно засвоїти основи React.js. Він також вимагає глибоких знань, як саме проводиться інтеграція користувальницького інтерфейсу у структуру MVC.
- React.js має невизначеність у оптимальності рішень. Існує дуже багато способів як вирішити конкретну задачу, про те не зрозуміло який з них є найоптимальнішим.
- Одностороння прив'язка даних, що збільшує ризики на виникнення можливих помилок.

1.4.1.2. Огляд Vue.js

Найбільш характерними особливостями [9] [10] фреймворку є:

- Економія у пам’яті [9][10]. Vue.js може займати близько 20 КБ, зберігаючи свою швидкість і гнучкість, що дозволяє досягати значно кращої продуктивності в порівнянні з іншими фреймворками.
- Чудова інтеграція [9][10]. Може використовуватися як для створення односторінкових додатків, так і для більш складних веб-застосунків.
- Детальна документація [9][10]. Існує однозначність у підходах до вирішення задач.

Щодо недоліків [9] [10], то можна виділити такі особливості мови:

- Брак ресурсів [9][10]. Vue.js все ще має досить невелику частку ринку в порівнянні з іншими фреймворками, що означає, що обмін знаннями в цій структурі все ще знаходиться на початковій фазі. Відповідно слабо й розвинута екосистема навколо самого фреймворку.
- Ризик надмірної гнучкості. Іноді у Vue.js можуть виникати проблеми при інтеграції у великі проекти [9][10]. Компанії не використовують Vue.js для проектування великих рішень.

1.4.1.3. Огляд Angular.js

У цьому фреймворці можна виділити такі особливості [9] [10]:

- Двостороння прив'язка даних, що забезпечує особливу поведінку програми, яка мінімізує ризики можливих помилок.
- MVVM (Model-View-ViewModel), що дозволяє розробникам працювати окремо над тим самим розділом програми, використовуючи той самий набір даних.
- Детальна документація.

До недоліків [9] [10] розробки на цьому фреймворці можна віднести такі особливості:

- Проблеми з міграцією, що часто виникають під час переходу від однієї версії до останньої.
- Складний синтаксис, що походить від першої версії Angular.

1.4.2. Прийняття рішення по фреймворку.

Отже, було проведено детальний аналіз кожного з найактуальніших веб фреймворків. Для предметної області, та для практичного застосування було вирішено обрати React.js як фреймворк для застосунку, тому що:

- Фреймворк найкраще [9] [10] справляється з великим об’ємом даних та легко працює з високим навантаженням.
- Він має найбільшу підтримку як з боку IT громади, так з боку компаній, які допомагають розвивати мову. Фреймворк займає найбільшу частку ринку створення веб застосунків.

Окрім того, було також розглянуто переваги безстанових (Stateless) компонентів при створенні програми за допомогою фреймворку Redux [12]:

- Існує єдиний стан всього застосунку. Дані до компонентів легко додаються, за необхідності, оскільки існує один стан та багато компонентів мають одночасний доступ до одного ресурсу. У компонентах з локальним станом, ця проблема вирішується дублікатом та перевизначенням вже існуючого стану, що є проблемою надлишковості.
- Реалізація логіки для застосунку відокремлена від її візуальної частини, що дає велику гнучкість у розробці.

На основі цих факторів, було вирішено, що Redux [12] буде гарним фреймворком у React.js для побудови великого та надійного застосунку.

1.4.3. Typescript чи Javascript для написання програми? [13]

Звичайний React.js без використання Typescript є доволі гнучким інструментом розробки, проте виникають багато різних помилок та ситуацій, у яких Typescript є очевидно необхідним у застосунку:

- Використання типів [13] на етапі компіляції програми дозволяє швидше виявляти помилки у програмах, надавати гарантію на маніпулювання змінними у функціях.

- Він також надає можливість [13] використовувати інтерфейси, які задають поведінку для наших об’єктів у клієнтській частині та задавати контракти для реалізації методів у класах.
- Typescript своєю природою [13] документує код, аби була можливість потім повернутись до нього та легко зорієнтуватися.

Отже, розглянувши переваги Typescript під час реалізації React.js коду, було вирішено підтримувати програму за допомогою цієї мови.

Розділ 2. Проектування та розробка системи

2.1. Проектування бази даних

2.1.1. Entity-Relationship модель застосунку

Оскільки було обрано реляційно базу даних, прийнято описувати сутності, відношення між ними за допомогою Entity-Relationship моделі (далі ER моделі).

Для самого опису ER моделі застосунку було обрано платформу draw.io. Вона надає повний функціонал для проектування Entity-Relationship моделі.

Предметну область застосунку можна виокремити такими основними сутностями, а саме:

- Методист.
- Запис.
- Викладач.
- Предмет.
- Група.

Опишемо кожну з цих сутностей більш детально, та зобразимо зв’язки між кожною з наведених сутностей.

Перш за все, розглянемо центруючу сутність, навколо якої вибудовується уся архітектура застосунку, а саме - “Розклад”.

Як можна переконатись, сутність пов'язана з декількома допоміжними сутностями. Опишемо більш детально самі зв’язки між сутностями та їхні обмеження:

- “Пара”. Атрибутами сутності слугують номер пари, та відповідно години, у які дана пара триває. Зв’язок необов’язковий, тому що “Пара” може існувати незалежно від сутності “Розклад”.
- “День”. Містить у собі поле, що зберігає назви днів, у які студенти та викладачі відвідують університет. Зв’язок необов’язковий, тому що “День” може існувати незалежно від сутності “День Запису”.
- “День Запису”. Реалізація зв’язку “багато-до-багатьох” між сутностями “День” та “Розклад”. Оскільки один запис у розкладі може містити багато днів, у які проводиться пара. Аналогічно й день міститься у багатьох записах в розкладі. Зв’язок обов’язковий, тому що “День Запису” не може існувати незалежно від сутності “День” та “Розклад”.
- “Тиждень”. Відповідає за збереження номерів тижнів, у які навчаються студенти та читають викладачі. Зв’язок необов’язковий, тому що “Тиждень” може існувати незалежно від сутності “Тиждень Запису”.
- “Тиждень Запису”. Реалізація зв’язку “багато-до-багатьох” між сутностями “Тиждень” та “Розклад”. Оскільки один запис у розкладі може містити багато тижнів, у які проводиться пара. Аналогічно й тиждень міститься у багатьох записах в розкладі. Зв’язок обов’язковий, тому що “Тиждень Запису” не може існувати незалежно від сутності “Тиждень” та “Розклад”.
- “Аудиторія”. Слугує для збереження назв/номерів кабінетів. Зв’язок необов’язковий, тому що “Аудиторія” може існувати незалежно від сутності “Розклад”.
- “Корпус”. Зв’язана з “Аудиторією” сутність, що слугує для збереження номерів корпусів. Оскільки “Корпус” може містити багато “Аудиторій”, відповідно було встановлено зв’язок “один-до-багатьох”. Зв’язок

необов’язковий, тому що “Копус” може існувати незалежно від сутності “Аудиторія”.

Як можна зрозуміти з опису даних, вони визначають часову інформацію, інформацію за місцезнаходженням пар, що першочергово є найважливішою інформацією для складання розкладу.

Дана частина моделі була першочергово зведена до 4нф (див. додаток Б) для забезпечення цілісності даних та усуненню аномалій при запитах на зміну даних. Проте після роздумів над доцільністю даної форми, було спрощено модель до попередньої форми (див. додаток А).

Розглянемо, на прикладі, модель “Пара”. Очевидно, що атрибути сутності “Пара”, а саме: початок та кінець пар - не потребують 4нф, оскільки їх значення залишатимуться незмінними. Перша пара як проводилась об 8:30 ранку, так і далі проводитиметься, а тривалість пар навряд чи зміниться.

Аналогічно, було проведено роздуми й над іншими сутностями. В результаті було отримано результуючу модель (див. додаток А).

Наступний блок, який необхідно охопити стосується пов’язуючих з розкладом сутностей, а саме: “Група”, “Предмет” та “Викладач” (див. додаток В).

Опис даних визначає інформативну складову нашого застосунку про кожну з залучених сутностей. Звідси ми дізнаємось ім’я та посаду викладача; назву предмету, спеціальність та курс, на якому він викладається; групу, до якої належать викладач та предмет а також її номер.

Опишемо більш детально самі зв’язки між сутностями та їхні обмеження:

- “Предмет”. Зв’язок необов’язковий, тому що “Пара” може існувати незалежно від сутності “Група”. Зв’язок “один-до-багатьох”, тому що один предмет може читатись одночасно у багатьох групах.
- “Спеціальність”. Зв’язок необов’язковий, тому що “Спеціальність” може існувати незалежно від сутності “Предмет”. Зв’язок “один-до-багатьох”, тому що одна спеціальність може належати одночасно багатьом предметам.
- “Курс”. Зв’язок необов’язковий, тому що “Курс” може існувати незалежно від сутності “Предмет”. Зв’язок “один-до-багатьох”, тому що один курс може належати одночасно багатьом предметам.
- “Викладач”. Зв’язок необов’язковий, тому що “Викладач” може існувати незалежно від сутності “Група”. Зв’язок “один-до-багатьох”, тому що один викладач може читати одночасно у багатьох групах.
- “Посада Викладача”. Зв’язок необов’язковий, тому що “Посада Викладача” може існувати незалежно від сутності “Викладач”. Зв’язок “один-до-багатьох”, тому що одна посада може належати одночасно багатьом викладачам.
- “Група”. Зв’язок обов’язковий, тому що “Група” не може існувати незалежно від сутності “Розклад”. Зв’язок “один-до-одного”, тому що розклад створюється лише для однієї групи.

2.1.2. З’єднання з базою даних у Spring Boot. Hibernate та HQL

Для реалізації побудованої моделі було обрано фреймворк Hibernate. Фреймворк надає легкий для використання каркас для відображення між об’єктно-орієнтованою моделлю даних та традиційною реляційною.

У програмі було реалізовано класове налаштування конфігурацій Hibernate для використання PostgreSQL СУБД у застосунку. (див. додаток Г)

Оскільки було використано фреймворк Hibernate, процес написання SQL запитів значно стростився, оскільки він надає можливість використовувати Hibernate Query Language (далі HQL) [16]. Переваги використання HQL є очевидними, оскільки вони значно полегшують інтеграцію Boot Spring з PostgreSQL. Детальніше про переваги:

- HQL забезпечує повну підтримку реляційних операцій [16]. Надається можливість представляти SQL-запити у вигляді об'єктів, які використовує Boot Spring. Він використовує класи та їхні властивості замість таблиць та стовпців.
- Результати запитів мають форму об’єктів [16], а не простого тексту, як у звичайному SQL. Ці об'єкти можна використовувати для маніпулювання або отримання даних у подальшому в програмі. Це позбавляє від необхідності явного створення об'єктів та заповнення ітеративних структур даних із набору результатів, отриманих під час виконання запиту.
- HQL підтримує [16] значно більше розширених функцій у порівнянні з SQL, таких як: пагінація, об'єднання за допомогою динамічного профілювання (ініціалізація асоціацій та збір значень з їхніми батьківськими об'єктами може бути здійснена за допомогою одного оператора SELECT), внутрішні / зовнішні / повні об'єднання та декартовий добуток. HQL також підтримує [16] стандартні функції агрегації, як: прогнозування, агрегування (max., avg.), групування, упорядкування, підзапити та виклики функцій SQL.

Саме ці особливості були використані у роботі над застосунком. Приклад з програми - запит на пошук за ім’ям аудиторії (див. додаток Д).

Можемо прослідкувати, як використовуються вищеназвані особливості мови запитів Hibernate: усе що потрібно для його успішного виконання - це використовувати класи, що визначають таблиці для наших схем бази даних як об’єкти. Також, HQL надає “синтаксичний цукор” [16] у вигляді оминання деяких ключових слів, як SELECT, оскільки розуміє що на виході потрібно побудувати результуючий об’єкт.

2.1.3. Визначення таблиць бази даних у Spring Boot. Persistence та Lombok

Для визначення всіх сутностей у базі було використано бібліотеку Java Persistence API. Вона є Java ORM стандартом для управління, зберігання та створення доступу до об'єктів [11] Java у реляційних базах даних.

Бібліотеку Lombok було використано для визначення базових операцій над створеними класами, які відповідають за об’єкти у реляційних базах даних. У основі бібліотеки лежить лаконічна ініціалізація стандартних, базових методів, конструкторів, сеттерів та геттерів класу за допомогою декораторів. У роботі було використано такі декоратори:

- “@AllArgsConstructor”. Відповідає за автоматичну генерацію конструктору зі всіма параметрами для цільового класу.
- “@NoArgsConstructor”. Відповідає за генерацію конструктора без параметрів для цільового класу.
- “@Data”. Відповідає за створення геттерів та сеттерів для усіх властивостей у цільовому класі. Також він відповідає за створення методу toString(), hashCode() та equals(), що використовуються для приведення об’єкта до звичайного тексту, створення хешу екземплярові класу та порівняння екземплярів класу відповідно.

Для того, щоб було більш наглядно як усі ці конструкції використовуються у проекті, візьмемо одну з моделей створеної бази даних, та розглянемо її більш детально, покроково (Додаток Е).

1. Визначення класу “Day” розпочинається з декораторів “@Entity” та “@Table” - саме вони сприяють синхронізації з нашою базою даних PostgreSQL, і визначають таблицю з іменем “Day”.

2. За допомогою декораторів бібліотеки Lombok [11] генеруються стандартні операції над сутністю “Day”, які були зазначені у описі можливостей цієї бібліотеки.

3. Наостанок, визначення атрибутів нашої цільової сутності.

3.1. Визначено первинний ключ з іменем “id” за допомогою Java Persistence [11] API декоратора “@Id”. Також, необхідно зазначити властивості первинного ключа:

- Визначено ім’я стовпця - “id”.
- Він не може бути невизначеним.
- Впродовж програми ми не можемо перевизначити первинний ключ.
- Первинний ключ повинен бути унікальним у всій таблиці.
- Для генерації значення первинного ключа використовується стратегія “Sequence”. Вона допомагає генерувати унікальні значення для первинних ключів інкрементним способом, але область видимості автоінкременту не лише сама таблиця, а вся база даних.

3.2. Було визначено звичайний атрибут таблиці, “name”, якому було надано такі властивості:

- Визначено ім’я стовпця - “name”.

- Зазначено, що цей атрибут буде визначений як тип даних “text”.
- Він не може бути невизначеним.

2.2 Розробка клієнт-серверної частини

Наступним етапом для побудови застосунку став пошук оптимальної для предметної області клієнт-серверної архітектури.

У роботі був проведений детальний огляд та аналіз різного стеку технологій для реалізації застосунку.

2.2.1. Розробка серверної частини

У цій секції буде продемонстровано побудоване середовище для розробки та підтримки застосунку, буде розглянуто архітектурну реалізацію бекенд рішення застосунку, а також огляд REST API для клієнт-серверної комунікації.

2.2.1.1. Середовище розробки

Перед проектуванням архітектури було вирішено, що потрібно поділити цикл розробки на певні проміжні етапи, які мають своє призначення, а саме: етап розробки, та етап розгортання рішення.

Вони між собою відрізняються концептуальними призначеннями та безпекою у реалізації.

Етап розробки мав такі властивості:

- Було налаштовано технологію “Hot-reload”, яка допомогла пришвидшити процес розробки шляхом уникнення ручного перезапуску проекту після внесення змін. Ця технологія дозволяє автоматично слідкувати за усіма використовуваними файлами у проекті та, відповідно, змінювати лише їх.

- Середовище було пристосовано до тестових даних системи.
- Сервіси, які будувались у проекті не були незахищені від зовнішніх атак.

Етап готового ж рішення мав наступні властивості:

- Насамперед, стояло питання безпеки, тому кожен з сервісів був надійно захищений. Комунікація могла відбуватись лише поміж сервісів, що знаходяться у одній мережі. У такому випадку “зовнішній світ” не має можливості спілкуватись з нашим рішенням.
- Було усунуто усі можливі “Hot-reload” імітації для того, щоб застосунок був повністю захищений і ми не мали впливу на нього після його розгортання на сервері.

2.2.1.2. Архітектура бекенд рішення

У даній роботі був використаний концепт упакування усіх складових застосунку у певні ізольовані, мікросистеми - сервіси [15].

У цьому разі, було використано Docker та Docker-compose.

Docker - це відкрита платформа для розробки, розгортання та запуску застосунків. Docker дозволяє ізолювати програми-складові від інфраструктури, щоб можна було швидко розгортати розроблене програмне забезпечення. Методологія Docker [15] слугує для швидкого циклу розробки та розгортання сервісу, тому він дозволив значно скоротити затримку між написанням коду та його запуском на сервері.

У попередньому розділі, було описано необхідність ізоляції не лише самих сервісів, а й циклів розробки також. Docker [15] вирішує усі названі проблеми у досить елегантний та простий спосіб.

На прикладі розглянуто як було реалізовано сервіс для адміністрування та керування базою даних PostgreSQL за допомогою сервісу PGAdmin (Додаток Є).

Як можна переконатись, сервіс, що зображений у додатку, налаштований для етапу локальної розробки, оскільки у ньому прослідковуються такі, характерні для нього, особливості:

- Restart [15] - відповідає за перевірку здоров'я сервісу та автоматичний перезапуск у тому разі, коли сервіс не запустився успішно.
- Ports [15] - відповідає за відкриття доступу до цільового сервісу у локальній мережі сервісів, та у зовнішньому світі.

У нашому випадку, порти у локальній розробці відкриті для того, щоб створити синхронізацію з локальною машиною, на якій розробляється застосунок та відобразити цільовий сервіс у неї.

На рисунку (див. додаток Ж) зображено ту ж саму реалізацію сервісу, проте з характерними особливостями готового до розгортання рішення, а саме:

- Expose [15] - використовується лише для відкриття доступу у локальній мережі сервісів. Зовнішній світ не матиме доступу до цільового сервісу.
- Restart - у цьому прикладі він відповідає лише за одноразовий й неповторний запуск сервісу.

Аналогічним способом було побудовано й інші сервіси, що використовуються у роботі, а саме:

- "Postgres"- сервіс для маніпулюванням базою даних PostgreSQL.

- “Pgadmin”- сервіс, який слугує адміністраційною сторінкою для бази PostgreSQL. У ньому присутній інтерфейс для швидкого внесення змін до бази.
- “Server”- сервіс, що слугує для створення логіки застосунку, для комунікації з іншими сервісами, які існують у мережі.
- “Client”- сервіс, що відповідає за візуальну складову кабінету методистів. Саме цим сервісом користуватимуться методисти для маніпулювання розкладом НаУКМА. У основі сервісу побудована логіка для відображення усіх компонентів, пов’язаних з внесенням інформації з розкладу, користувачам.
- “Admin”- сервіс, що відповідає за візуальну складову кабінету адміністрації. Саме у основі цього сервісу побудована логіка для відображення усіх компонентів, пов’язаних з внесенням інформації про самих користувачів-методистів, оновлення адміністраційних даних програми.
- “Nginx”- сервіс, що відповідає за налаштування комунікації між клієнтами-користувачами та серверною частиною застосунку.

У цій секції більш детально будуть розглянуті сервіси, що стосуються саме серверної частини, а у наступній - клієнтської.

2.2.1.3. Сервіси серверної частини застосунку

2.2.1.3.1. Сервіс “Nginx”

Розпочнемо з сервісу “Nginx” [14], оскільки з нього починається комунікація з сервісом в нашій програмі [14].

Як вже було зазначено раніше - Nginx є зворотною посередницькою проксі-службою [14], яка приймає на вході запит клієнта, та передає його

одному або декільком серверам. Після обробки запиту клієнту, проксі надсилає відповідь сервера/серверів назад клієнту.

Більшість сучасних застосунків можуть працювати як веб-сервер самостійно, проте веб-сервер “Nginx” забезпечує низку вдосконалених функцій, таких як:

- Балансування навантаження поміж серверів. [14] Можливість зваженого балансування, та розподілення трафіку по сервісам [14].
- Можливості та прискорення TLS / SSL [14], яких бракує у більшості спеціалізованих застосунків.

У програмі (див. Додаток З) реалізовано конфігурацію для серверу, яка описує наступну логіку:

- Запити приймаються за портами 80 та 443, що відповідають за HTTP, та HTTPS протоколи.
- Налаштовано SSL за допомогою HTTPS сертифікатів.
- Кожному запиту, який потрапляє у наш веб-проксі сервер присвоюється стандартні проксі хедери, які задають адресу, від кого прийшов запит, і в який сервіс потрібно його направити. Саме ці налаштування дозволяють спростити навігацію по мережі сервісів.
- У випадку, якщо користувач не наткнувся на HTTPS адресу веб-сервісу ”Nginx”, його автоматично перенаправить саме на безпечний протокол.
- І наостанок, за допомогою команди UPSTREAM можливо налаштувати направлення трафіку саме у докеризований сервіс, який буде обробляти запит користувача. У цьому випадку це “Server”.

2.2.1.3.2. Сервіси “Postgres” та “Pgadmin”

Наступні серверні сервіси є ключовими для інформативної роботи побудованої програми. Ці сервіси відповідають за збереження та маніпуляцію усіма вхідними даними у програмі, є сховищем для всієї інформації, що надходить як від користувачів-методистів застосунку, так й від адміністраторів.

Сервіс “Postgres” побудований на основі Docker відображенні Postgres:9.6.1 версії, та підтримує такі можливості [6]:

- Середовище логування. Усі повідомлення з різними рівнями логування записуються у файл на локальній машині, на якій розгорнуто сервіс “Postgres”.
- Авторизація до бази даних, початкова ініціалізація суперкористувача бази даних.
- Сервіс перевірки статусу здоров’я. Кожних 10 секунд перевіряється статус бази даних.

Сервіс “Pgadmin” побудований на основі Docker відображення Dpage/pgadmin4 версії, та підтримує такі можливості:

- Авторизація до сервісу контролю над базою даних.
- Авторизація до сервісу бази даних, для того щоб мати доступ до даних, які є у цільовій схемі.

2.2.1.3.1. Сервіс “Server”

Наступна секція присвячена огляду використаних пакетів та бібліотек у застосунку, побудові архітектури серверної частини програми, реалізації комунікації між сервісами.

Оскільки було використано Spring Boot [11] для написання бекенду програми, було використано пакет, який містить стандартний набір класів, за

допомогою якого можна побудувати основу для веб-застосунку, а саме - org.springframework.boot [11]. Цей пакет використовується для налаштування логування, тестування, для побудови веб-серверу, для побудови сутностей бази даних та налаштування середовища розробки у застосунку.

Усі ці допоміжні пакети детально описані у pom.xml файлі, який фіксує детальний опис та версійність зазначених вище пакетів.

Розглянемо більш детально кожен з них. Першим описаною залежністю буде саме spring-boot-starter-web [11]. Пакет використовує вбудовані у бібліотеку функції, які дозволяють швидко розгортати сервер на машині, на якій запускається програма (Рис 1.1).

```
1   package com.danorel.scheduler;
2
3   import org.springframework.boot.SpringApplication;
4   import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6   @SpringBootApplication
7   public class SchedulerApplication {
8
9       public static void main(String[] args) {
10          SpringApplication.run(SchedulerApplication.class, args);
11      }
12  }
13
```

Рис 2.1. Початкова точка входу у сервер

Наступна бібліотека зі стандартного пакету, на яку варто звернути увагу - sprint-boot-starter-login [11]. Бібліотека синхронізується з усіма діями, що відбуваються у компонентах Spring пакету, і відповідно відбувається процес логування. Для створення логів, що призначені для відслідковування елементів програми, було використано пакет org.slf4j [11]. (Рис. 1.2.)

```
@Configuration
class DatabaseLoader {

    private static final Logger log = LoggerFactory.getLogger(DatabaseLoader.class);

    @Bean
    CommandLineRunner initTeacherTypeTable(TeacherTypeRepository repository) {

        return args -> {
            log.info("Preloading " + repository.save(new TeacherType( id: 1L,  name: "Старший викладач")));
            log.info("Preloading " + repository.save(new TeacherType( id: 2L,  name: "Викладач")));
            log.info("Preloading " + repository.save(new TeacherType( id: 3L,  name: "Викладач-стажист")));
            log.info("Preloading " + repository.save(new TeacherType( id: 4L,  name: "Доцент")));
            log.info("Preloading " + repository.save(new TeacherType( id: 5L,  name: "Декан факультету")));
            log.info("Preloading " + repository.save(new TeacherType( id: 6L,  name: "Завідувач кафедри")));
            log.info("Preloading " + repository.save(new TeacherType( id: 7L,  name: "Професор")));
        };
    }

```

Рис 2.2. Логування Spring Boot

Стандартний пакет також надає можливість використовувати середовище розробки для імітації “Hot-Reloading” для докеризованого сервісу “Server” під час етапу розробки. Використовуючи бібліотеку spring-boot-devtools, програма синхронізується з контейнером та автоматично перезапускається при внесенні змін до вихідного коду програми.

Останній пакет стандартної бібліотека це spring-boot-starter-data-jpa [11]. Саме він надає можливість синхронізувати Java Persistence API (далі JPA) з базою даних. Налаштування нижче (Рис 1.3) ініціалізує конфігурації PostgreSQL і пов’язує їх з JPA.

```
## Devtools standard config for reload
spring.devtools.restart.poll-interval=2s
spring.devtools.restart.quiet-period=1s

# Drivers for db
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.database=postgresql
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQL9Dialect
spring.jpa.properties.hibernate.enable_lazy_load_no_trans=true
```

Рис 2.3. - Підключення Java Persistence API

Для проектування бекенд архітектури та для налаштування комунікації між сервісами було використано паттерн проектування Model-View-ViewModel (далі MVVM). [10]

У якості Model - виступає база даних PostgreSQL, яка слугує для зберігання всіх сутностей, що існують у нашій предметній області.

У якості View - слугує React-івський клієнтський код, що відповідає за віддачу сторінок користувачам сервісу. У режимі розробки виступає як окремий сервіс, у режимі розгортання сервісу - будуються статичні файли клієнтського коду, які потім відображаються на серверній частині.

У якості ViewModel ми маємо сам по собі Spring Boot сервер, що відповідає за збереження та маніпулювання сутностями з бази даних. Насамперед, події обробляються сервером, отримуються з клієнтської частини, та надсилаються оброблені дані на клієнт. Оброблені дані зберігаються у нашій Model сутності, у нашому випадку - у базі даних PostgreSQL. Таким чином ViewModel виступає посередником між двома сутностями (Рис 1.4).

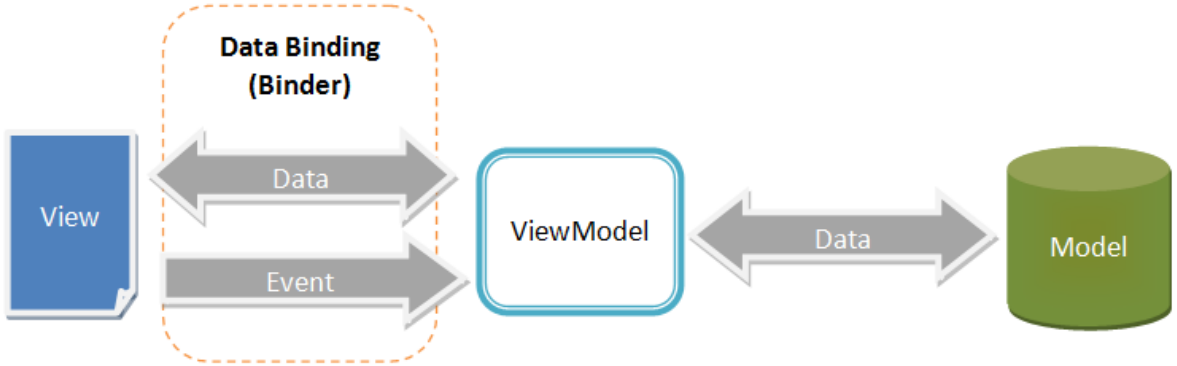


Рис 2.4. - Модель MVVM. Приклад комунікації. Абстракція

Для налаштування комунікації між сервісами в мережі було обрано REST API для виконання цього завдання.

Архітектура частини комунікації складає у себе такі сутності:

- Моделі/сутності - Визначають класи та їхні властивості, що використовуються у предметній області.
- Репозиторії - Визначають стандартний набір операцій над сутностями.
- Контроллери - Модуль, що відкриває шляхи для комунікації з сервером.

Моделі/сутності були у попередньому розділі.

Репозиторії задають інтерфейс для керування моделями у базі даних. Базовий набір операцій складає функції типу CRUD (Create, Read, Update, Delete, далі просто CRUD). Для того, щоб над кожною сутністю можна було провести усі базові операції, було визначено до кожної з них інтерфейс.

Для того, щоб Spring Boot мій розпізнавати інтерфейс класу як репозиторій, потрібно додати до нього декоратор @Repository [11] (див. додаток І)

Spring Boot також надає можливість коротко та лаконічно реалізувати [11] CRUD операції за допомогою шаблонного інтерфейсу CrudRepository<T, ID>, який приймає на вході два параметри:

- T - Це шаблон, що відповідає за сутність, над якою будуть проведені усі базові операції.
- ID - Тип ідентифікатору, первинний ключа (далі PK) сутності.

Саме таким чином було визначено базові операції над усіма сутностями бази даних. При необхідності додавання нових функцій до цільового репозиторію, Java надає можливість множинного успадкування та реалізації інтерфейсів.

Контроллери, що були використані у даній праці (див. додаток И), визначають “інтерфейс” для комунікації між сервісами. Оскільки було використано REST API [5], відповідно у контроллерах ми маємо можливість визначити усі базові операції, а саме: DELETE, POST, GET, PUT, OPTION. У цих методах було використано репозиторій, що й сам по собі реалізовує усі зазначені вище методи.

Для визначення контроллеру, було використано Spring Boot [11] декоратор, що має назву @RestController. Таким чином було визначено, що даний клас слугуватиме маршрутизатором для спілкування по протоколу HTTP.

Декоратор @RequestMapping - визначає, за яким шляхом буде визначений кінцевий шлях передачі повідомлень. У прикладі (див. додаток И) можна переконатись, що контроллер було визначено за шляхом “/api/teacher”. Усі вищезазначені REST API методи можна використовувати наступним чином:

- @GetMapping: “/api/teacher/” - Обирає усіх викладачів, які є створені у нашій базі. Декоратор визначив, що функція приймає лише GET запити.

- @PostMapping: “/api/teacher/” - Створює нового викладача у базі. Анотацією, було зазначено, що функція приймає лише POST запити.
- @PutMapping: “/api/teacher/{id}” - Змінює вже внесеного викладача у базу на нового, з новими атрибутами/властивостями. Анотацією, було зазначено, що функція приймає лише PUT запити, що призначені на зміну об’єктів.
- @DeleteMapping: “/api/teacher/{id}” - Видаляє викладача з бази. Декоратор визначив, що функція приймає лише DELETE запити.

Слід також зазначити, що було використано анотацію Spring Boot [11] @CrossOrigin. Саме вона дозволила надсилати запити до API кінцевих шляхів, та додала до заголовку HTTP-запиту механізм для завантаження ресурсів - CORS (Cross-Origin Resource Sharing). Саме цей механізм дозволяє оминати OPTION відповіді до REST API серверу та успішно повертати відповіді від цільових методів HTTP.

2.2.2 Розробка клієнтської частини

У цій секції буде розглянуто архітектурну реалізацію фронтенд рішення застосунку, продемонстровано створені інтерфейсні сторінки, що взаємодіють з користувачами застосунку.

Отже, в основі клієнтського коду лежить бібліотека, що підтримується гігантами технічних індустрій, а саме: компанією Facebook та Instagram - React.js [13]. Бібліотека розроблена з метою розробки інтерактивних інтерфейсів для користувачів веб-застосунків.

Фреймворк побудований навколо методології Single Page Application (далі SPA). Таким чином, дані протягом усього циклу програми мають змогу динамічно змінюватись, не перезавантажуючи сторінку знову і знову. Зі зміною сторінка перемальовується згідно нового, оновленого стану.

2.2.2.1. Стек фронтенд рішення

Повністю всю клієнтську частину програми було реалізовано за допомогою фреймворку Redux, концепція якого буде описана у наступному розділі.

Для того, щоб React.js став більш надійним та зручним у використанні, було вирішено використати Typescript [13] для вирішення проблем звичайного Javascript, що описані у першому розділі.

2.2.2.2. Архітектура фронтенд рішення

Оскільки було обрано фреймворк Redux [12], для керуванням загальним станом усього клієнтського застосунку, він пропонує дуже гарний шаблон проектування для всієї клієнтської програми, який називається “Ducks” (див. додаток К)

При проектуванні архітектури бази даних, відокремивши сутності бази даних у окремі структури, обєднавши їх за певними подібними властивостями та особливостями, одночасно було вирішено питання архітектури для клієнтського коду.

Кожна сутність має свій початковий стан, який змінюється протягом часу зі взаємодією з інтерфейсом користувача (див. додаток Л).

Для того щоб змінювати початковий стан, використовуються так звані Reducers, або ж редуктори. Усі операції над станом певної точкової сутності є іммутабельні, тобто дані у стані є незмінними після ініціалізації значенням і єдиним методом, як можна перевизначити дані це створити новий об’єкт (див. додаток М). В такому підході є багато переваг, а саме:

- Оскільки клієнтський застосунок є SAP, а він змінюється відразу після зміни самого стану сутності. Мутабельні типи у звичайному Javascript не дають змогу прослідкувати зміни у стані сутності і відповідно оновити

сторінку на React.js, проте іммутабельність, яка реалізована “під капотом” фреймворку Redux [12], має таку можливість. Саме тому, кожна зміна гарантовано відображається на сторінці клієнта.

- Крім того, дослідження показало швидкодійну перевагу використання іммутабельних типів даних над мутабельными.

2.2.2.3. Комунікація з REST API

Для того, щоб інтерфейс був справді інтерактивним для користувачів, необхідно створити сервіс, який буде отримувати та надсилати дані з бекенд частини. У проєкті було використано асинхронні функції-запити за допомогою бібліотеки axios (див. додаток Л).

Перед впровадженням REST API [5], для повноцінної імітації запитів на клієнтській частині слугували JSON “моки”, які дозволяли імітувати дані, що приходитимуть після надсилання запитів у REST API.

Після успішної реалізації бекенд частини застосунку, бібліотека axios дозволила використовувати стандартний набір функцій REST API [5] запитів для комунікації, та JSON формат для утворення параметризованих запитів.

Крім того, було використано додатковий проміжний крок для створення самих запитів за допомогою бібліотеки @reduxjs/toolik, функції createAsyncThunk. Саме ці функції слугують інтерфейсом для виклику асинхронних запитів з клієнтської частини програми.

Висновки

Головною метою роботи було проведення якісної побудови системи для платформи навчального розкладу. Було проведено повноцінне та детальне дослідження щодо вибору стеку для побудови платформи такого типу. Було створено прототип конструктору-розкладу, який був першочергово запланований у роботі. А також, було проведено роботи-демонстрації застосунку з методистом кафедри інформатики.

У подальшому можливі деякі вектори для вдосконалення застосунку, а саме:

- Синхронізація розкладу поміж багатьох методистів, що будуть користуватись застосунком.
- Автогенерація розкладу за допомогою вирішення задачі виконання обмежень.
- Розширення розкладу для усіх факультетів в університеті.

Список використаних джерел

1. Lauren S. SQL vs NoSQL [Електронний ресурс] / Schaefer Lauren // MongoDB Developer Advocate. – 2021. – Режим доступу до ресурсу: https://www.mongodb.com/nosql-explained/nosql-vs-sql.
2. simranjenny84. SQL advantages [Електронний ресурс] / simranjenny84. – 2020. – Режим доступу до ресурсу: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-sql/.
3. Aran D. Why Should You Use Java for Your Backend Infrastructure? [Електронний ресурс] / Davies Aran. – 2021. – Режим доступу до ресурсу: https://www.devteam.space/blog/why-should-you-use-java-for-your-backend-infrastructure/.
4. Oliya S. The Good and the Bad of Java Programming [Електронний ресурс] / Smith Oliya. – 2021. – Режим доступу до ресурсу: https://www.altexsoft.com/blog/engineering/pros-and-cons-of-java-programming/.
5. Paul L. REST, SOAP, GraphQL — Gesundheit! [Електронний ресурс] / Logan Paul. – 2018. – Режим доступу до ресурсу: https://medium.com/better-practices/rest-soap-graphql-gesundheit-6544053f65cf.
6. Priya P. What is PostgreSQL? [Електронний ресурс] / Pedamkar Priya. – 2021. – Режим доступу до ресурсу: https://www.educba.com/what-is-postgresql/.
7. Kent A. Advantage & Disadvantage of MySQL [Електронний ресурс] / Anderson Kent // MongoDB Developer Advocate. – 2019. – Режим доступу до ресурсу:

https://www.slideshare.net/KentAnderson43/advantage-disadvantage-of-mysql.

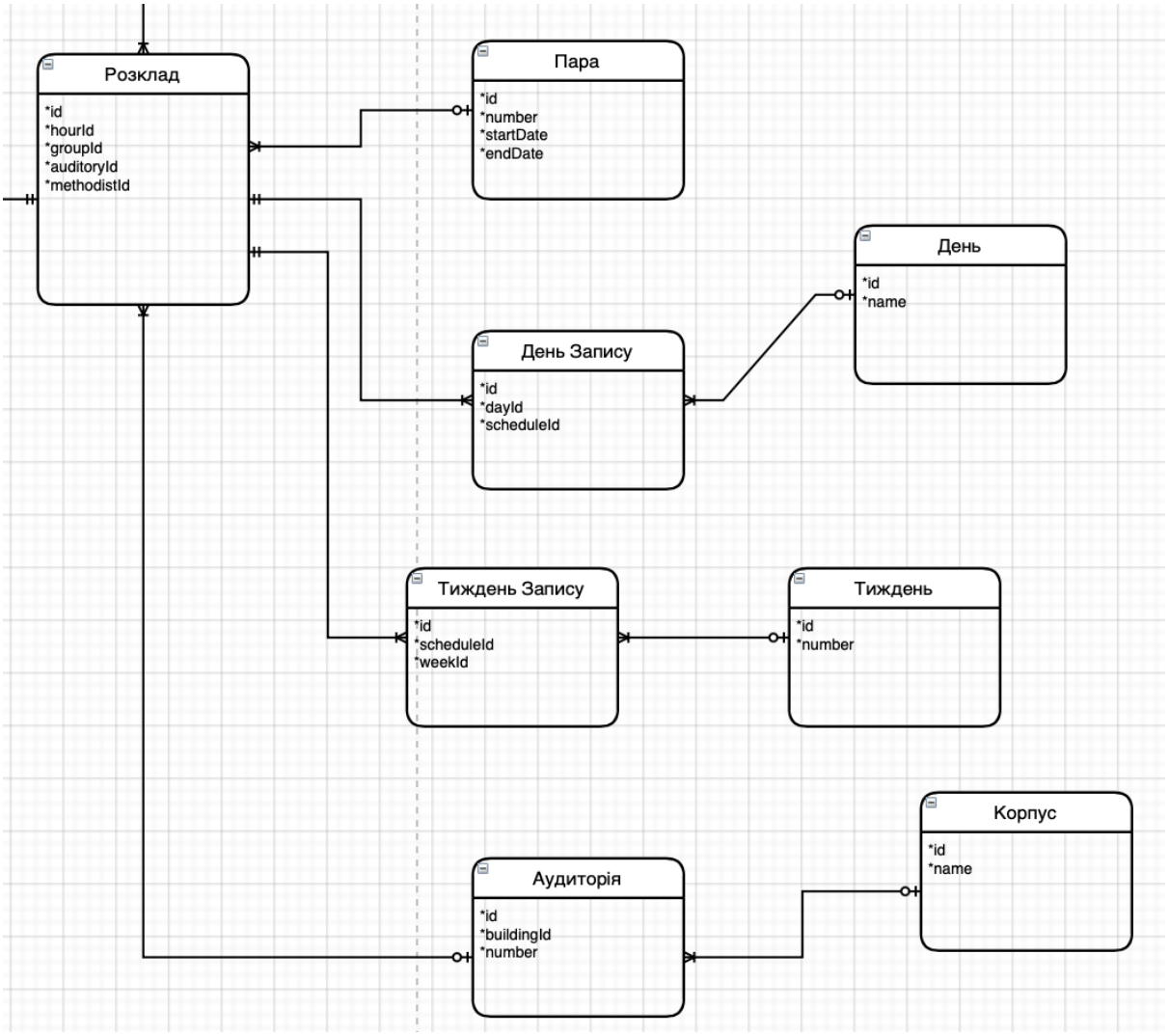
8. Victoria P. Why Use Java for Back-end Development? [Електронний ресурс] / Puzhevich Victoria. – 2020. – Режим доступу до ресурсу: https://scand.com/company/blog/why-use-java-for-back-end-development/.
9. Karol K. Angular vs React vs Vue: Which Framework to Choose in 2021 [Електронний ресурс] / Karol. – 2021. – Режим доступу до ресурсу: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/.
10. Romana K. React vs Angular vs Vue.js — What Is the Best Choice in 2021? [Електронний ресурс] / Kuts Romana. – 2021. – Режим доступу до ресурсу: https://blog.techmagic.co/reactjs-vs-angular-vs-vuejs-what-to-choose-in-2020/.
11. Pramod K. S. Understanding the Basics of Spring vs. Spring Boot [Електронний ресурс] / Kumar Srivastava Pramod. – 2018. – Режим доступу до ресурсу: https://dzone.com/articles/understanding-the-basics-of-spring-vs-spring-boot.
12. Antoine C. React/Redux: pitfalls and best practices [Електронний ресурс] / Caron Antoine. – 2020. – Режим доступу до ресурсу: https://slashgear.github.io/react-redux-pitfalls-and-best-pratices/.
13. Malesh H. How to use TypeScript in your React App? [Електронний ресурс] / Haldar Malesh. – 2019. – Режим доступу до ресурсу: https://blog.bitsrc.io/why-and-how-use-typescript-in-your-react-app-60e8987be8de.
14. Matthew C. What are benefits of using NGINX unit? [Електронний ресурс] / Casperson Matthew. – 2019. – Режим доступу до ресурсу: https://octopus.com/blog/why-use-nginx-unit.

15. Nick P. Docker Containers: The Pros and Cons of Docker [Електронний ресурс] / Puzhevich Nick. – 2020. – Режим доступу до ресурсу: https://blog.iron.io/docker-containers-the-pros-and-cons-of-docker/.
16. Sivateja K. HQL | Introduction [Електронний ресурс] / Kandula Sivateja. – 2019. – Режим доступу до ресурсу: https://www.geeksforgeeks.org/hql-introduction/.

Додатки

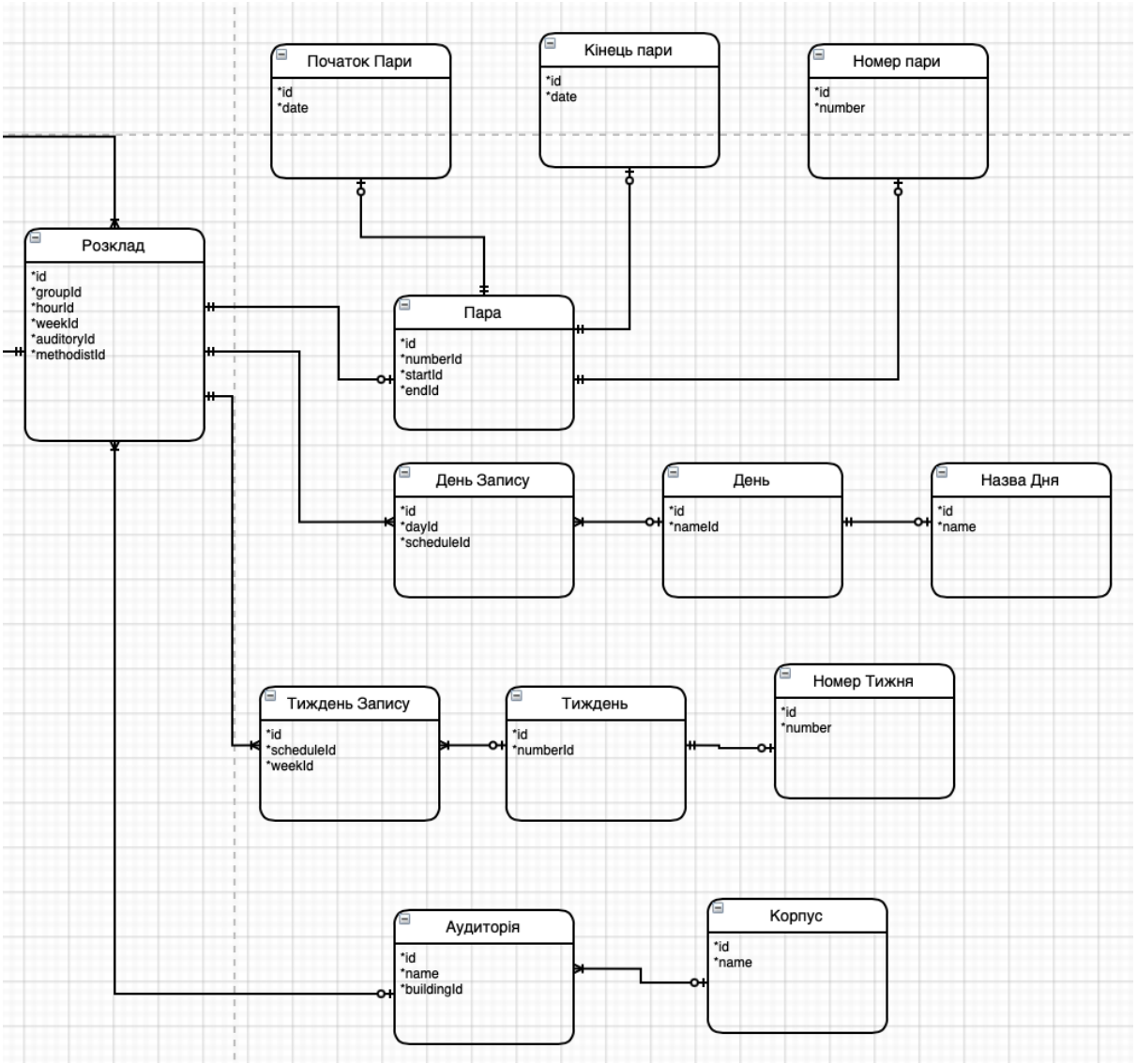
Додаток А

ER сутність “Розклад” та її допоміжні сутності



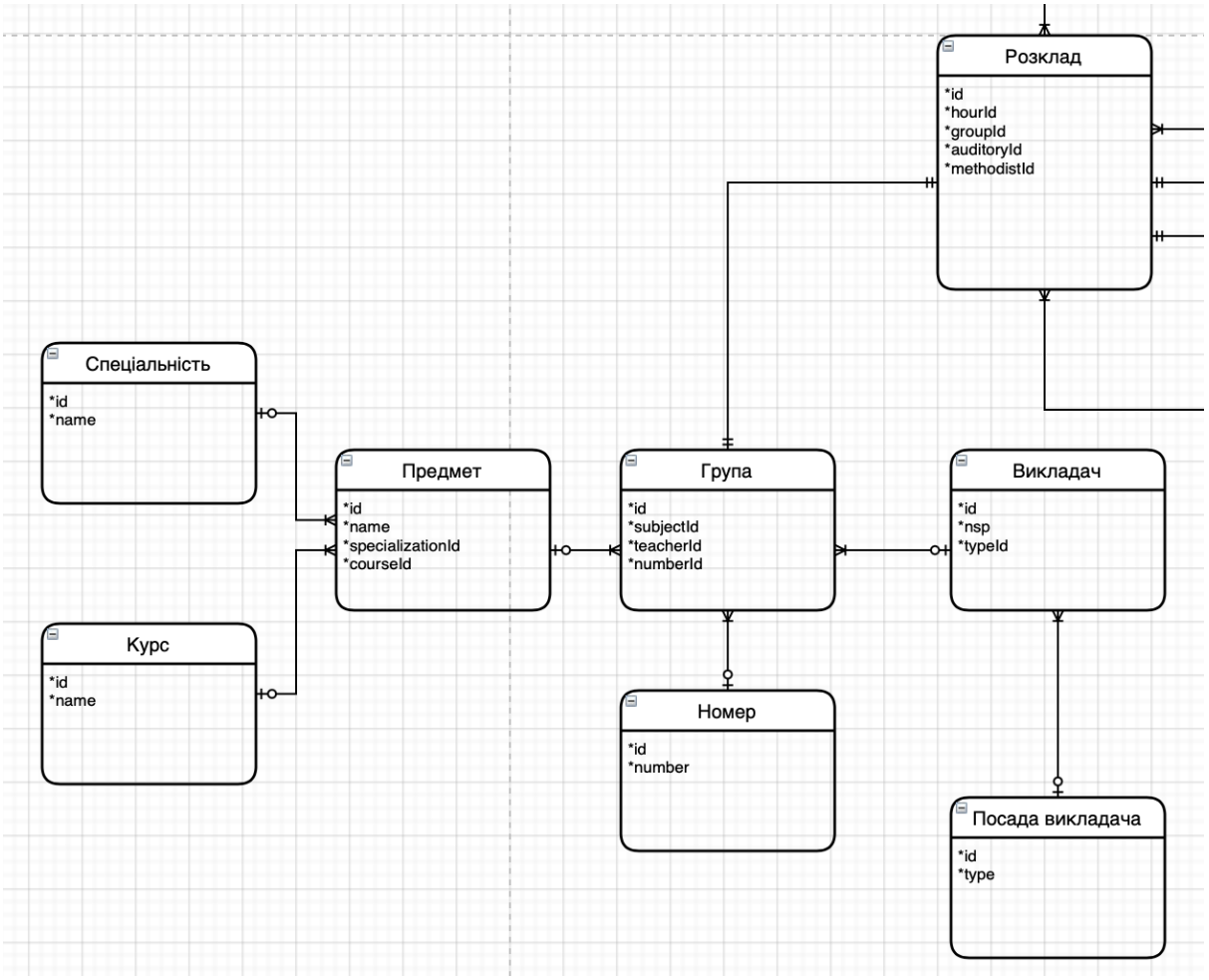
Додаток Б

ER сутність “Розклад” та її допоміжні сутності перед спрощенням. 4нф



Додаток В

ER сутності “Розклад”-”Група” та їхні допоміжні, інформативні сутності



Додаток Г

Налаштування Hibernate

```
// Create registry builder
StandardServiceRegistryBuilder registryBuilder = new StandardServiceRegistryBuilder();

Properties props = new Properties();

props.setProperty(Environment.DRIVER, "org.postgresql.Driver");
props.setProperty(Environment.URL, System.getenv( name: "SPRING_DATASOURCE_URL"));
props.setProperty(Environment.USER, System.getenv( name: "SPRING_DATASOURCE_USERNAME"));
props.setProperty(Environment.PASS, System.getenv( name: "SPRING_DATASOURCE_PASSWORD"));
props.setProperty(Environment.DIALECT, "org.hibernate.dialect.PostgreSQL9Dialect");
props.setProperty(Environment.ENABLE_LAZY_LOAD_NO_TRANS, "true");

Configuration cfg = new Configuration()
        .setProperties(props);

// Apply settings and return registry
registry = registryBuilder
        .applySettings(cfg.getProperties())
        .build();

// Create MetadataSources
MetadataSources sources = new MetadataSources(registry);
```

Додаток Д

HQL у дії. Запит на знаходження корпусу аудиторії за ім’ям

```
default AuditoryBuilding findByName(String name) {

    try {
        Session session = HibernateUtil
                .getSessionFactory()
                .openSession();

        session.beginTransaction();

        Query<AuditoryBuilding> query = session
                .createQuery( queryString: "FROM AuditoryBuilding WHERE AuditoryBuilding.name = :name",
                        AuditoryBuilding.class);
        query.setParameter( name: "name", name);

        List<AuditoryBuilding> auditoryBuildings = query.getResultList();

        session.close();

        if (auditoryBuildings.size() > 0)
            return auditoryBuildings.get(0);
```

Додаток Е

Визначення таблиць. Persistence та Lombok у дії

```java
import javax.persistence.*;
import java.io.Serializable;

@Entity(name = Day.PERSISTANCE_NAME)
@Table(name = Day.PERSISTANCE_NAME)
@AllArgsConstructor
@NoArgsConstructor
@Data
@JsonIgnoreProperties({"hibernateLazyInitializer"})
public class Day implements Serializable {

    static final String PERSISTANCE_NAME = "Day";

    @Id
    @Column(name = "id", unique = true, nullable = false, updatable = false)
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long id;

    @Column(name = "name", columnDefinition = "text", nullable = false)
    private String name;
}
```

Додаток Є

Реалізація сервісу для керування базою даних PostgreSQL на етапі розробки

```
pgadmin:
    image: dpage/pgadmin4
    container_name: pgadmin
    restart: unless-stopped
    environment:
        - PGADMIN_LISTEN_PORT=${PGADMIN_PORT}
        - PGADMIN_DEFAULT_EMAIL=${PGADMIN_DEFAULT_EMAIL}
        - PGADMIN_DEFAULT_PASSWORD=${PGADMIN_DEFAULT_PASSWORD}
    volumes:
        - pgadmin:/var/lib/pgadmin
    ports:
        - ${PGADMIN_PORT}:${PGADMIN_PORT}
        - "80:80"
    networks:
        - db
```

Додаток Ж

Реалізація сервісу для керування базою даних PostgreSQL на етапі розгортання

```
pgadmin:
    image: dpage/pgadmin4
    container_name: pgadmin
    restart: no
    environment:
        - PGADMIN_LISTEN_PORT=${PGADMIN_PORT}
        - PGADMIN_DEFAULT_EMAIL=${PGADMIN_DEFAULT_EMAIL}
        - PGADMIN_DEFAULT_PASSWORD=${PGADMIN_DEFAULT_PASSWORD}
    volumes:
        - pgadmin:/var/lib/pgadmin
    expose:
        - ${PGADMIN_PORT}
    networks:
        - db
```

Додаток З

Налаштування серверу за допомогою зворотнього проксі-серверу Nginx.

```
upstream server {
    server server:${SERVER_PORT};
}

server {
    listen 80;
    listen 443 ssl;

    ssl_certificate         /etc/nginx/conf.d/certs/server/fullchain.pem;
    ssl_certificate_key     /etc/nginx/conf.d/certs/server/privkey.pem;

    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;

    proxy_buffering off;
    proxy_intercept_errors on;
    proxy_request_buffering off;

    if ($scheme != "https") {
        return 301 https://$host$request_uri;
```

Додаток И

Контроллери для налаштування комунікації між сервісами.

```java
@RestController
@CrossOrigin(origins = "http://localhost:1024")
@RequestMapping("/api/teacher")
public class TeacherController {

    private final TeacherRepository repository;

    public TeacherController(TeacherRepository repository) { this.repository = repository; }

    @GetMapping("/")
    public @ResponseBody
    ResponseEntity<Object> getAll() {
        ArrayList<Teacher> foundTeachers = (ArrayList<Teacher>) repository.findAll();
        return new ResponseEntity<>(foundTeachers, HttpStatus.OK);
    }

    @GetMapping("/{id}")
    public ResponseEntity<Teacher> addSubject(@PathVariable("id") Long id) {
        Optional<Teacher> foundTeacher = repository.findById(id);
        return foundTeacher
                .map(ResponseEntity::ok)
                .orElseGet(() -> ResponseEntity.notFound().build());
```

Додаток І

Визначення CRUD операцій над сутністю “Викладач”.

```java
package com.danorel.scheduler.repositories.teacher;

import org.springframework.stereotype.Repository;
import org.springframework.data.repository.CrudRepository;

import com.danorel.scheduler.domain.entities.teacher.Teacher;

@Repository
public interface TeacherRepository extends CrudRepository<Teacher, Long> {}

```

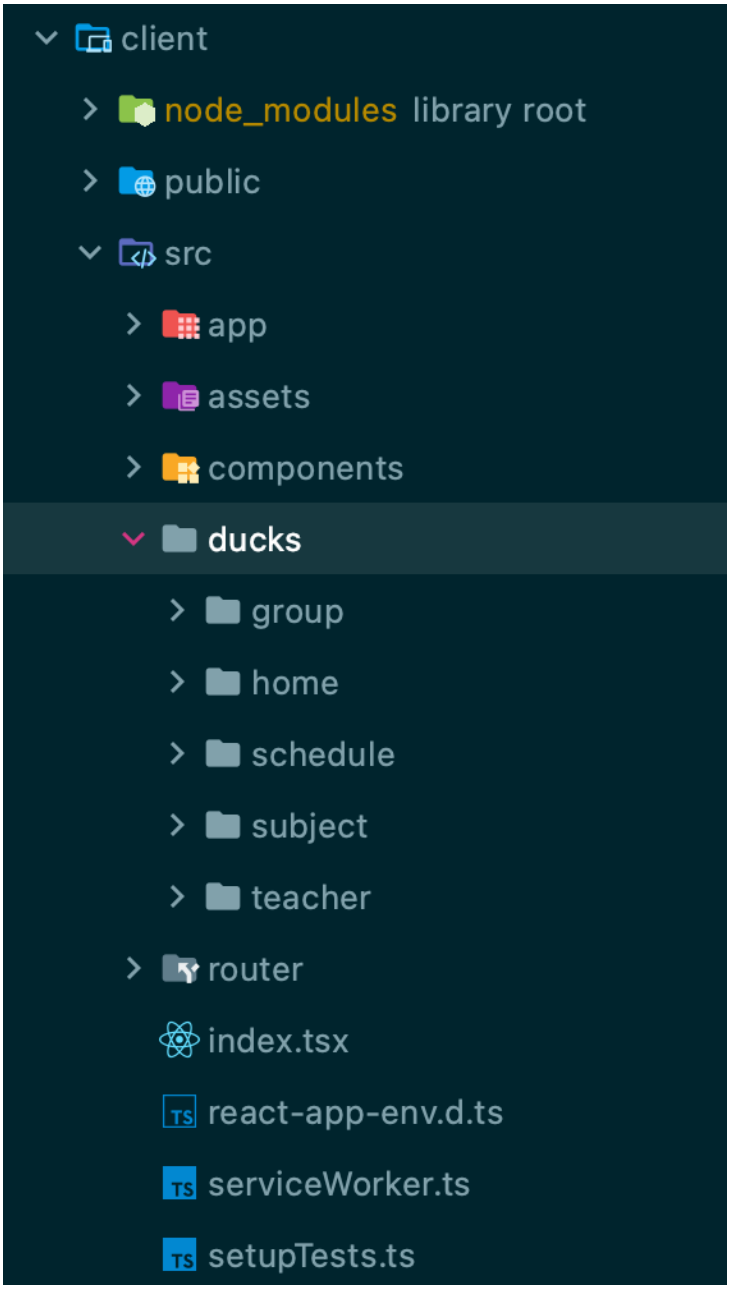
Додаток Ї

Функція відображення сторінки. Підключення фреймворку Redux

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';

import App from './router/App';
import { store } from './app/store';

import * as serviceWorker from './serviceWorker';

ReactDOM.render(
    <React.StrictMode>
        <Provider store={store}>
            <App/>
        </Provider>
    </React.StrictMode>,
    document.getElementById( elementId: 'root')
);

// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

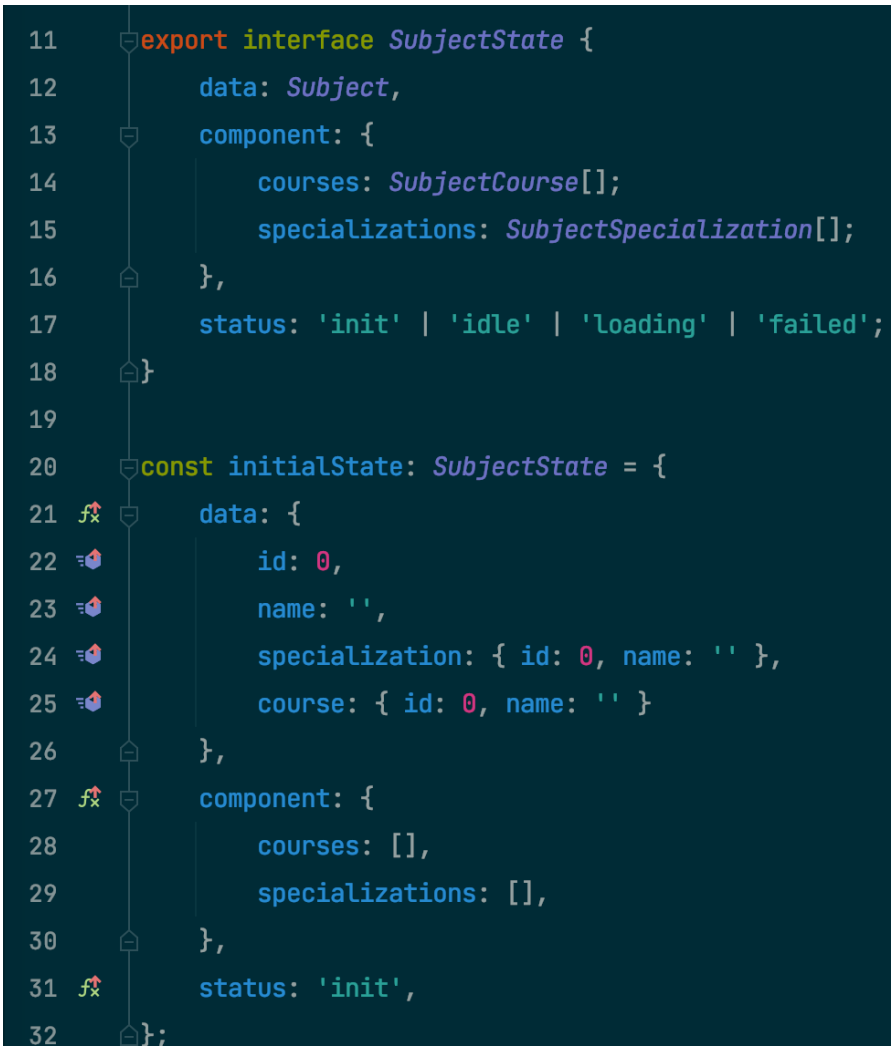
Додаток К

Файлова структура у клієнтському коді



Додаток Л

Початковий стан сутності “Subject” при проектуванні її за допомогою Redux шаблону “Ducks”



```
export interface SubjectState {
    data: Subject,
    component: {
        courses: SubjectCourse[];
        specializations: SubjectSpecialization[];
    },
    status: 'init' | 'idle' | 'loading' | 'failed';
}

const initialState: SubjectState = {
    data: {
        id: 0,
        name: '',
        specialization: { id: 0, name: '' },
        course: { id: 0, name: '' }
    },
    component: {
        courses: [],
        specializations: [],
    },
    status: 'init',
};
```

Додаток М

Зміна стану сутності “Subject” після отримання відповіді запиту до REST API

Додаток Н

Створення запитів до бекенд частини застосунку, нормалізація отриманих з REST API даних

```
/**
 * Courses GET API call.
 */

export type FetchCoursesResponse = SubjectCourse[];

export function normalizeFetchCourses(data: any): FetchCoursesResponse {
    return data.map((course: any) => ({
        id: String(course.id),
        name: String(course.name)
    }));
}

export function fetchCourses() {
    return new Promise<{ data: FetchCoursesResponse }>( executor: async (resolve) => {
        const response = await axios.get( url: 'http://localhost:4096/api/subject/course/');
        resolve( value: {data: normalizeFetchCourses(response.data)});
    });
}
```

```
import { createAsyncThunk } from "@reduxjs/toolkit";
import { postSubject, fetchCourses, fetchSpecializations } from "./service";

import { Subject } from 'Entities';

export const postSubjectAsync = createAsyncThunk(
    typePrefix: 'subject/postSubjectAsync',
    payloadCreator: async (subject: Subject) => {
        const response = await postSubject(subject);
        return response.data;
    }
)

export const fetchCoursesAsync = createAsyncThunk(
    typePrefix: 'subject/fetchCoursesAsync',
    payloadCreator: async () => {
        const response = await fetchCourses();
        return response.data;
    },
);
```