МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота бакалавра

ДОДАТОК ДЛЯ БРОНЮВАННЯ НАВЧАЛЬНИХ АУДИТОРІЙ СУМСЬКОГО ДЕРЖАВНОГО УНІВЕРСИТЕТУ.

Здобувач освіти гр. ІН – 82 Владислав САВЧЕНКО

Науковий керівник,
кандидат фізико-математичних наук,
доцент кафедри комп'ютерних наук Олена ПРОЦЕНКО

Завідувач кафедри
доктор технічних наук, професор Анатолій ДОВБИШ

СУМИ 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Затверджую_____
Зав. кафедрою Довбиш А.С.
"_____" _____2022 р.

ЗАВДАННЯ

до кваліфікаційної роботи

здобувача вищої освіти четвертого курсу, групи ІН-82 спеціальності «122 – Комп'ютерні науки» денної форми навчання Савченка Владислава Сергійовича.

Тема: «ДОДАТОК ДЛЯ БРОНЮВАННЯ НАВЧАЛЬНИХ АУДИТОРІЙ СУМСЬКОГО ДЕРЖАВНОГО УНІВЕРСИТЕТУ»

Затверджена наказом по СумДУ
№_____ від _____2022 р.

Зміст пояснювальної записки: 1) аналітичний огляд методів і тенденцій створення веб-додатків 2) проектування додатку для бронювання аудиторій; 3) реалізація веб-додатку.

Дата видачі завдання «_____»_____ 2022 р.
Керівник роботи _____Олена ПРОЦЕНКО
Завдання прийняв до виконання _____ Владислав САВЧЕНКО

РЕФЕРАТ

Записка: 61 стор., 44 рис., 3 табл., 1 додаток, 15 джерел.

Об’єкт дослідження – Додаток для бронювання навчальних аудиторій Сумського державного університету

Мета роботи — розробка односторінкового додатку для бронювання аудиторій в Сумському державному університеті.

Результати — розроблено програмне забезпечення у вигляді односторінокового веб-додатку, що дозволяє бронювати аудиторії в Сумському державному університеті. Додаток реалізовано за допомогою JavaScript бібліотеки React.js, мови програмування Java, з використання системи керування реляційними базами даних PostgreSQL.

ОДНОСТОРІНКОВИЙ ВЕБ-ДОДАТОК, АУДИТОРІЇ,
БРОНЮВАННЯ, REACT, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС,
БАЗИ ДАНИХ, JAVA.

ЗМІСТ

ВСТУП

Університети. Це величезні комплекси споруд з купою авдиторій, лабораторій, дослідницьких центрів тощо. Серед такої кількості відділень та кабінетів з легкістю можна загубитися і в реальному житті, не кажучи вже про інформаційне середовище, де вказується зайнятість того чи іншого кабінету в певний момент часу.

Напевно кожен, хто хотів провести зайняття або якийсь захід в певній, на його думку підходящій для цього авдиторії, стикався з проблемою не точного відображення актуального списку вільних кабінетів.

Причин для виникнення цієї проблеми може бути декілька. Від небажання зазначати, що та чи інша авдиторія буде зайнята в певний час до відсутності зручного і централізованого способу це зробити.

На допомогу у вирішенні цього питання можуть прийти сучасні віяння моди в розробці сайтів та веб-додатків. Наприклад так популярні сьогодні односторінкові веб-додатки.

Односторінковий веб-додаток – це сайт, що складається з однієї сторінки, яка складається з динамічних та статичних елементів.

Динамічні елементи оновлюються відповідно до намірів користувача. Наприклад, під час перевірки електронної пошти заголовок і бічна панель зазвичай залишаються незмінними, коли ви змінюєте папки вхідних, спам і надіслані. Таким чином, односторінкові програми уникають надсилання непотрібної інформації у ваш браузер і значно прискорюють час завантаження [1].

Перевага веб-додатку полягає у тому, що всі дії виконуються у браузері. Користувачеві не потрібно встановлювати додаток для смартфону або комп’ютеру.

Метою роботи є розробка односторінкового веб-додатку для бронювання авдиторій для проведення лекційних чи практичних зайнять або додаткових заходів викладачами і студентами.

Створений додаток буде універсальним і його легко буде використовувати на користь різних університетів просто підключаючи свою базу даних із заздалегідь визначеною структурою інформації, що зберігається.

1. ЛІТЕРАТУРНИЙ ОГЛЯД ЗА ОБРАНОЮ ТЕМАТИКОЮ РОБОТИ

1.1. Аналіз принципів побудови односторінкових веб-додатків

На сьогоднішній день веб-додатки поділяються на багатосторінкові (MPA) та односторінкові (SPA).

Відмінність MPA полягає у тому, що дані розкидані по багатьох сторінках і їх відображення відбувається разом з підвантаження нових сторінок, коли користувач виконує натискання на кнопку або переходить за гіперпосилання у браузері. Окрім зручності це також впливає на швидкодію додатку, бо кожного разу, коли треба відобразити нові дані, доводиться перезавантажувати цілу сторінку, тобто отримувати із серверу нову купу коду.

Натомість SPA позбавлений цієї проблеми. Він є достатньо швидким бо основний скелет додатку завантажується тільки при вході в додаток. Далі будь-які зміни у відображенні відбуваються точково, саме там, де це потрібно без перевантаження всієї сторінки.

Структуру односторінкового додатку можна поділити на три рівні: користувацький інтерфейс, бізнес логіка, сховище даних.

Користувацький інтерфейс (UI) – це клієнтська частина додатку, те що забезпечує відображення даних і взаємодію користувача з додатком.

Основним принципом в побудові користувацького інтерфейсу є використання так званих компонент. Це блоки коду за допомогою яких, як конструктор, складється увесь додаток.

Головним правилом для створення компонент є принцип багаторазового використання. Тобто бажано, щоб блоки створювались таким чином, щоб їх можна було використовувати в інших частинах додатку. Але це правило не виключає наявність спеціальних компонент, створених в одному екземплярі під певні потреби.

Бізнес логіка є своєрідним мостом між користувачем і даними. Вона забезпечує обробку дій користувача і отримання, обробку і передачу інформації до UI.

Сховищем даних в основному виступає база даних, де зберігається інформація якою користувач, за посередництва бізнес логіки, може оперувати. В залежності від обраних дій це може бути читання, додавання, редагування або видалення певної інформації.

1.2. Інструменти для розробки веб-додатків

У світі веб розробки точаться суперечки який набір технологій кращий і більш універсальний, але точно, що для розробки користувацької частини сайту чи додатку люди уже досить давно було виокремлено три, а віднедавна 2 фаворити. Це React.js, View.js та Angular (поступово відходить через вищий поріг входження, ніж в конкурентів).

Усі вони побудовані на базі JavaScript, мають свої особливості, перш за все пов’язані з тим бібліотека це, як у випадку з React чи фреймовр, як View. А також дещо відмінний синтаксис. Але спільним у них є компонентний підхід до побудови додатку.

Перевагою React над конкурентами є її невеликий розмір і більша гнучкість. Те, що це бібліотека, дає можливість не використовувати зайві модулі, натомість, за допомогою Node.js і вбудованого менеджера пакетів npm, додавати виключно те, що безпосередньо використовуватиметься при розробці додатку.

Вибір інструментів для організації взаємодії з сервером в більшій мірі залежить від характеру додатку та навантаження, яке очікується при його використанні.

Найбільш використовуваними на сьогоднішній день є Java з фреймоврком Spring, Python і його фреймоврк Django, .NET, Node.js та інші фреймоврки на базі JavaScript.

До прикладу Java в основному використовується для великих високонавантажених систем через її вміння гарно працювати з ресурсами та сувору типізацію даних.

Також ще використовують PHP, але з приходом розуміння, що сувора типізація все ж краще, його позиції похитнулись.

Велике розмаїття інструментів для роботи з базами даних викликає певні ускладнення при проектуванні додатку.

Зараз існує два види баз даних: реляційні та нереляційні. Їх відмінність полягає у тому, що реляційні бази зберігають інформацію у вигляді структурованих таблиць, а нереляційні виглядають як набір даних без строгої структури.

Реляційні бази даних є більш безпечними адже дотримуються принципу ACID (атомарність, непротирічність, ізольованість, довговічність). Відповідність до них гарантує цілісність та збереження даних, а також передбачуваність роботи бази даних [2].

Нереляційні ж бази не використовують у своїй роботі цей принцип, але натомість дають швидкий і гнучкий доступ до великого об’єму даних, що може бути корисним для таких сфер, як Big Data.

Серед систем адміністрування реляційних баз даних найбільшої популярності на даний момент набули OracleDB, PostgreSQL, MariaDB, MySQL, SQLite. Всі вони в основі містять синтаксис SQL зі своїми вкрапленнями.

Керування нереляційними базами даних в основному відбувається за допомогою MongoDB.

1.3. Постановка задачі

Метою даного проєкту є створення програмного продукту, односторінкового додатку для бронювання аудиторій.

Для реалізації проєкту у повному обсязі потрібно виконати наступні задачі:

1. провести порівняння доступний на сьогоднішній день інструментів для розробки веб-додатків та обрати найбільш підходящі для вирішення проблем цього проєкту;
2. розробити максимально доступний і зручний користувацький інтерфейс;
3. спроектувати структуру бази даних;
4. виконати проектування бізнес логіки додатку для забезпечення оптимальної роботи додатку з даними
5. перевірити працездатність додатку на основі отриманих результатів зробити висновки про виконання роботи.

Про успішну реалізацію додатку свідчитиме можливість за допомогою нього відслідковувати зайнятість авдиторій та бронювати їх під свої потреби.

2. МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

2.1. Проектування веб-додатку

Розроблятися веб-додаток буде за допомогою JavaScript бібліотеки React.js для користувацького інтерфейсу, Java для серверної частини та PostgreSQL для керування базами даних.

React.js досить легка бібліотека і не має у собі великої кількості передвстановлених пакетів, які часто можуть навіть не використовуватись при розробці додатків. Натомість є можливість встановлювати виключно потрібні пакети за допомогою менеджера пакетів Node.js.

Java використовуватиметься для написання логіки роботи додатку з базою даних та обробки запитів користувача на сервері. Ця мова програмування створена на основі C++, слідує всім принципам ООП, має сувору типізацію даних та зручний і достатньо популярний на сьогоднішній день фрейморк для створення веб-додатків Spring.

PostgreSQL є відкритою системою для управління реляційними базами даних. Працює на основі синтаксису SQL. Має зручну панель адміністрування баз та пакет для легшої побудови ієрархічних відношень між даними.

Для полегшення користування додатком сторінки у ньому будуть виконані у стилі мінімалізм з чітко визначеною колірною палітрою та мінімальним набором елементів на сторінці, щоб не заплутати користувача в процесі роботи із сайтом.

Додаток складатиметься з декількох збірних компонент та компонент по-менше, що входитимуть до складу збірних.

Однією з основних компонент буде так званий Header. Він міститиме логотип та назву додатку а також кнопку початку роботи з ним. Хедер буде постійно закріплений на сторінці незалежно від етапу, який проходить користувач в даний момент.

При запуску додатку під хедером розміщуватиметься інформаційний блок. Він надаватиме інформацію про час роботи університету, посилання на сайт, адресу та пошту. На рисунку 2.1 представлено схематичне зображення стартового екрану додатку.

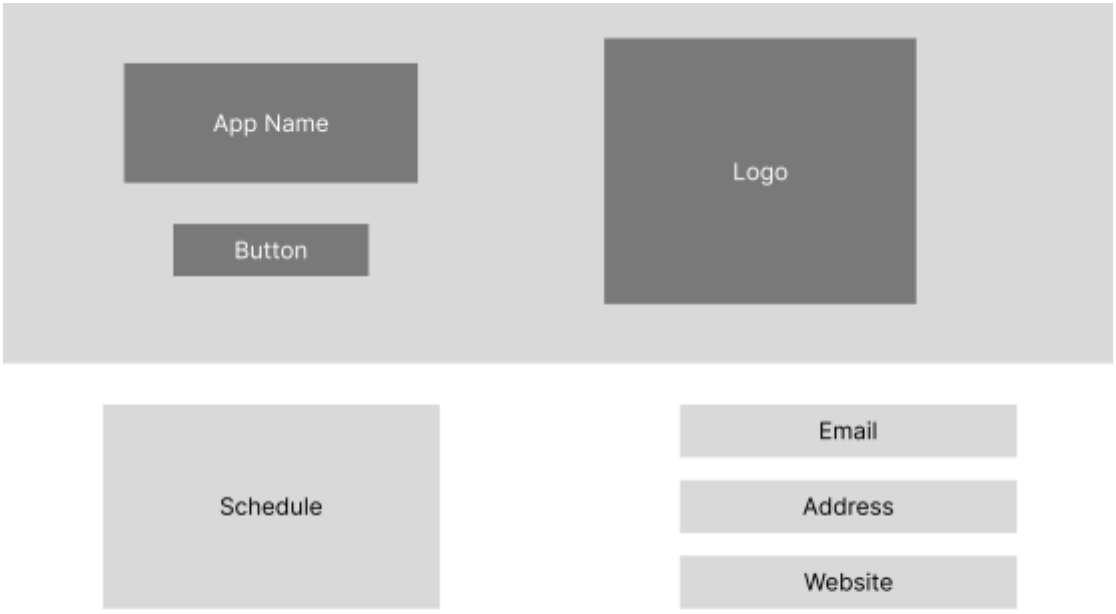

Рисунок 2.1 – Стартовий екран додатку

Після натискання користувачем кнопки в хедері розпочнеться процес бронювання аудиторії.

Під час бронювання користувачеві потрібно буде пройти декілька етапів на кожному з яких треба буде обрати потрібний варіант типу авдиторії, корпусу університету, номеру кабінету тощо. Варіанти будуть представлені у вигляді карток з кнопкою вибору.

Рисунок 2.2 – Схематичне зображення картки

Рисунок 2.2 відображає макет карток, що використовуватимуться для представлення варіантів вибору користувачеві.

Вибір, де буде представлено багато варіантів, наприклад аудиторії, буде виконаний за допомогою слайдеру. Керування слайдером буде реалізовано за допомогою перемикачів під картками або миші, де треба буде потягнути в потрібному напрямку, щоб «перегорнути» елементи слайдеру. Схематичне зображення сторінки зі слайдером зображено на рисунку 2.3.

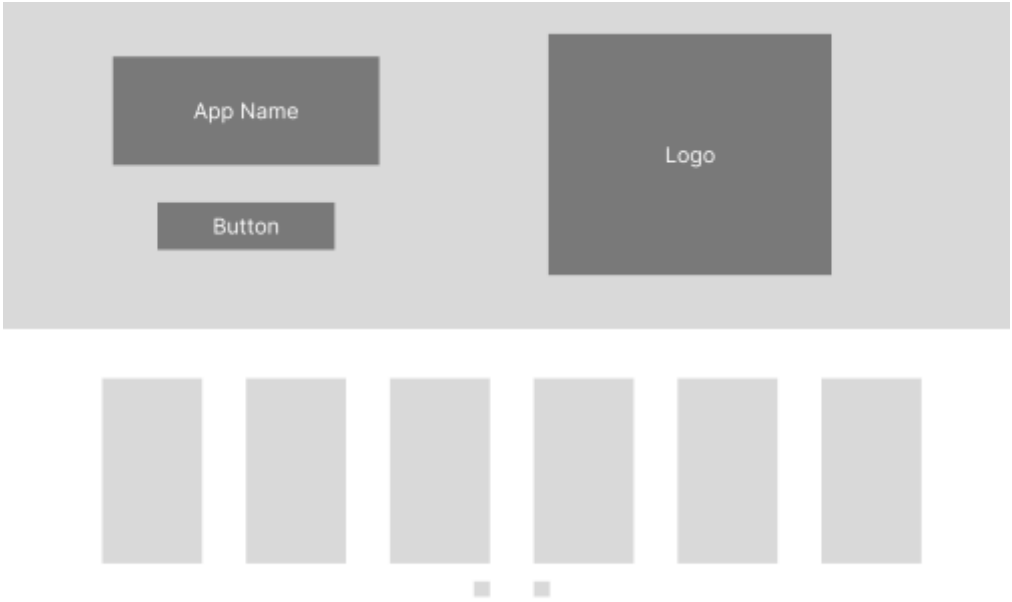

Рисунок 2.4 – Схематичне зображення слайдеру

Два останні блоки додатку – це блок для вибору дати і часу та блок підтвердження бронювання. Сторінка підтвердження міститиме поля для вводу імені та адреси електронної пошти користувача, а також інформацію про вибрану авдиторію. Як це зображено на рисунку 2.4.

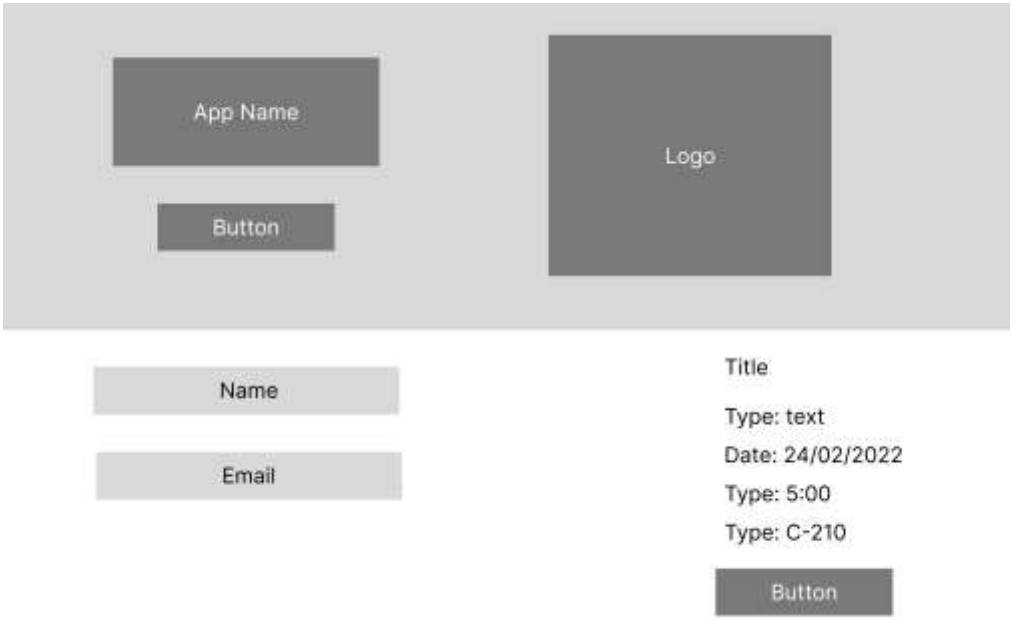

Рисунок 2.4 – Сторінка підтвердження

Завершення роботи з додатком супроводжуватиметься натискання кнопки «Підтвердити» та переадресацією на стартову сторінку.

2.2. Проектування бази даних

База даних додатку складатиметься з декількох таблиць. Таблиці з корпусами університету, таблиці з аудиторіями та таблиці із забронюваними аудиторіями. Таблиця з аудиторіями буде пов’язана з таблицею з корпусами зовнішнім ключем. Ключем виступатиме сам корпус.

Таблиця з корпусами матиме дві колонки. Перша – це id, друга – назва корпусу.

Таблиця 2.1 – Корпуси університету

Name	**Type**	**Unique**	**Required**	**Primary key**	**Foreign key**
id	number	true	true	true	false
campus	varchar2	true	true	false	true

З таблиці 2.1 бачимо, що поля id і campus є унікальними, щоб виключити можливість створення однакових записів та обов'язковими. Також видно, що id є первинним ключем у цій таблиці, а campus слугуватиме зовнішнім ключем для зв'язку з аудиторіями.

Таблиця 2.2 – Аудиторії

Name	**Type**	**Unique**	**Required**	**Primary key**	**Foreign key**
id	number	true	true	true	false
number	varchar2	true	true	false	false
campus	varchar2	false	true	false	true
type	varchar2	false	true	false	false

Таблиця 2.2 зберігатиме дані про аудиторії, що є в університеті, а саме: номер аудиторії у форматі перша літера корпусу і номер, назву корпусу, де вона знаходиться, та тип, лекційна чи практична.

Таблиця 2.3 – Забронювані авдиторії

Name	**Type**	**Unique**	**Required**	**Primary key**	**Foreign key**
id	number	true	true	true	false
number	varchar2	false	true	false	false

campus	varchar2	false	true	false	true
type	varchar2	false	true	false	false
date	date	false	true	false	false
startTime	varchar2	false	true	false	false
endTime	varchar2	false	true	false	false
name	varchar2	false	true	false	false
email	varchar2	false	true	false	false

У таблицю із забронюваними аудиторіями (таб.2.3) буде поміщатися інформація, що надходитиме від користувача після підтвердження замовлення: дані аудиторії (такі ж, що і в таблиці аудиторій), обрана дата та час, поділений на час початку та час кінця, для полегшення роботи з ним в коді. Ім’я та пошта користувача, що забронював аудиторію.

Після того, як час пройде помітку endTime запис автоматично видалятиметься.

2.3. Redux для керування станом

Стан — це вбудований об’єкт React, який використовується для зберігання даних або інформації про компонент. Стан компонента може змінюватися з часом; щоразу, коли він змінюється, компонент повторно відмальовується. Зміна стану може відбутися як відповідь на дії користувача або системні події, і ці зміни визначають поведінку компонента та те, як він буде відображатися [3].

Оскільки цей проєкт не матиме звичних для сайту сторінок, то звичайну навігацію за допомогою гіперпосилань зробити не вдасться. Тому було прийнято рішення змінювати блоки додатку за допомогою зміни стану додатку. Це дозволить легко відмальовувати потрібний блок у відповідь на дії користувача.

Керування станом у проєкті не є чимось специфічним. Його можна організувати власноруч навіть стандартними засобами чистого JavaScript. Також React має власні інструменти для роботи зі станом додатку, але зараз на ринку представлено багато бібліотек сконфігурованих саме під задачі управління станом додатку та обробки подій, пов’язаних з його змінами. Серед найбільш відомих Redux, MobX та Overmind.

Для керування станом в проекті було обрано Redux, адже додаток не об’ємний, обробки складних змін стану та подій пов’язаних з ними, як, наприклад, це відбувається в Overmind, не потребує, а тому потрібна невелика бібліотека, яка вирішить питання відмальовки компонент і не збільшуватиме розмір додатку своєю присутністю. Також React вже перевірена часом бібліотека, що має велику користувацьку підтримку.

До React-проекту ця бібліотека додається за допомогою менеджера пакетів. Для цього в консолі потрібно прописати npm install react-redux, бібліотека автоматично встановиться і додасться до залежностей проєкту, щоб вразі зміни пристрою на якому ведеться розробка розробник зміг відновити всі встановлені бібліотеки.

2.4. Організація стилів в додатку

На початку розвитку інтернету перші сайти являли собою статичну HTML верстку, яка просто виводила потрібну інформацію користувачу. Але з плином часу з’явилася потреба цю інформацію організувати та подати в більш зручному на гарному вигляді.

Для вирішення цієї проблеми було створено каскадні таблиці стилю або ж CSS.

Каскадні таблиці стилю (CSS) – це простий механізм для додавання стилів (шрифтів, кольорів, відступів тощо) до Веб документу [4].

Щоб додати стилі на сайт варто лише підключити файл з розширенням .css у якому прописані потрібні властивості до html-документу.

Зараз же варіантів роботи зі стилями більше. Особливо якщо брати до уваги, що більшість сайтів і веб-додатків створюються за допомогою фреймворків.

На сьогоднішній день ринок пропонує використовувати готові бібліотеки зі стилями (Bootstrap, MaterialUI), препроцесори (SASS, LESS) або прописувати стилі безпосередньо у файлі де відбувається ініціалізація компонент, так звані styled-components.

Кожен із способів має свої недоліки і переваги. Бібліотеки стилей, наприклад, мають гарно скомпоновані стилі, що слідують сучасним трендам в розробці. Але вони є важкими, адже мають у собі прописані стилі для всіх можливих елементів і різних їх станів. Зазвичай використовується дуже обмежена її частина, а інше просто збільшує об’єм додатку.

Стилізовані компоненти або styled-components зручні з точки зору наочності. Розробник одразу бачить які стилі використовуються для цього компоненту і де вони задіяні. Виключається момент ходіння файловою структурою у пошуках потрібного файлу зі стилями.

Препроцесори – надлаштування над стандартним CSS, що привносять додаткові можливості, а також роблять написання коду стилів простішим і код більш придатним для читання, тобто привносять «синтаксичний цукор».

Основними нововведеннями, які приносять препроцесори, є можливість використовувати змінні, якщо однакове значення використовується у декількох місцях. Корисно, наприклад, записати кольорову палітру сайту в змінні і потім легко підставляти потрібні кольори в тому місці, де це потрібно. Також зручним є використання вкладеності. За допомогою вкладеності можна показати ієрархію елементів, а також приналежність псевдо елементів до певного класу у файлі.

Міксини є одним з головних нововведень, що привносять препроцесори. Міксин – це заздалегідь заготовлений шматочок коду, що може часто

використовуватись. І замість повторного прописування одного і того ж, підключається міксин, там де потрібно.

Виходячи з описаних вище переваг і недоліків різних способі організації стилів в додатку, для проєкту було обрано використання препроцесору. Хоча він і потребує наявність інтерпретатору, що буде перетворювати SCSS синтаксис на синтаксис звичайного CSS, але це не є проблемою, оскільки він підключається разом з SCSS-бібліотекою до проекту через менеджер пакетів.

React також приносить деякі нові можливості в організацію стилів, які варто використати, а саме модулі. Використання модулів стилів починається з того моменту, як в файловій системі додатку створюється файл з розширенням module.css (module.scss). Ці файли імпортуються в компоненти як і звичайні файли стилів, але не мають глобального характеру змін, а отже одні й ті ж класи можуть використовуватись від компоненти до компоненти, не впливаючи на роботу додатку.

За правилами оформлення структури додатку для кожної компоненти створюється свій модуль з аналогічною назвою, але його може не бути, якщо компонента не потребує власних стилів.

3. РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ

3.1. Процес розробки користувацького інтерфейсу

Розробка веб-додатку розпочинається зі створення користувацького інтерфейсу. Користувацький інтерфейс – одна з найважливіших частин додатку, адже за його допомогою відбувається взаємодія користувача із сервером і даними.

Перше, що бачитиме користувач при запуску додатку це header з назвою додатку, логотипом та кнопкою для початку роботи.

У файловій структурі проєкту header представлений одноіменною компонентою та модулем стилей для неї (рис.3.1).

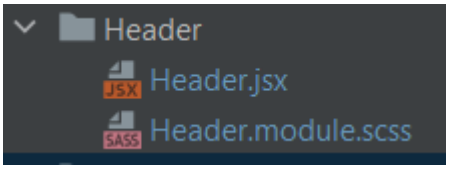

Рисунок 3.1 – Представлення Header у файловій структурі

Заголовок задається HTML елементом h1 з назвою додатку «Book It Yourself».

Кнопка «Book Now», що запускає процес бронювання стилізована градієнтом, який задається за допомогою властивості linear-gradient з указанням напрямку зміни кольорів і хекс-кодів самих кольорів. При наведенні курсору на елемент, програється анімація «збільшення».

Логотип являє собою годинник з календарем, що відсилає до прямого призначення сайту (рис.3.2).

Рисунок 3.2 – Логотип додатку

Також header задає основну колірну гаму додатку, що поєднує холодний відтінок блакитного для фону та жовтий і світло-коричневий для елементів взаємодії з користувачем.

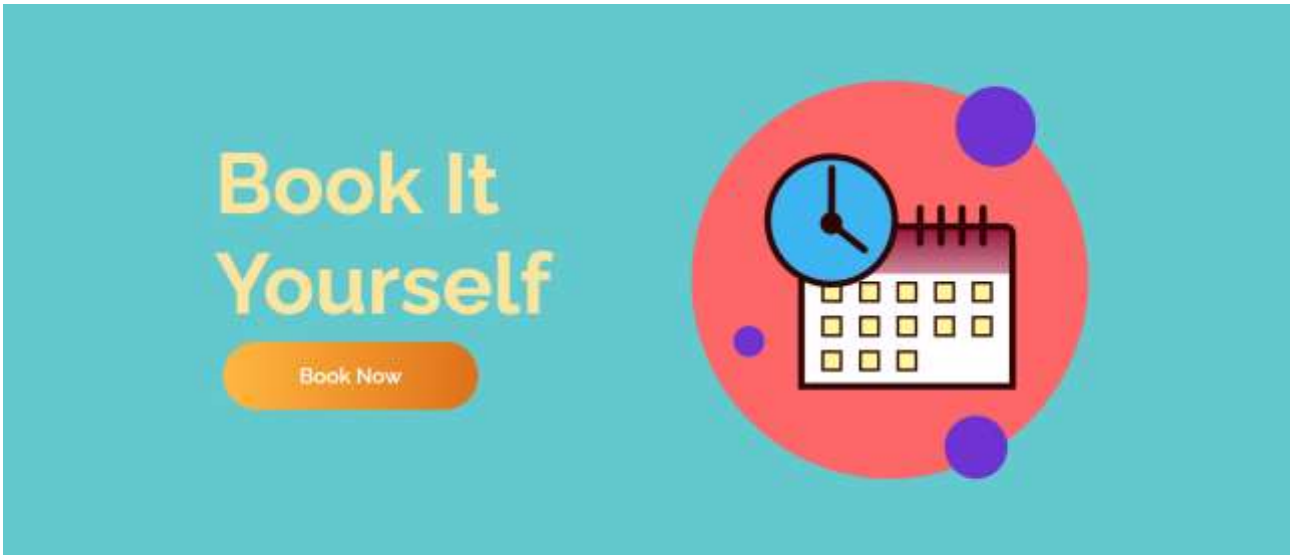

Рисунок 3.3 – Header

Наступною великою складовою частиною інтерфейсу додатку є компонента MainPage. Вона займає другу половину екрану і є контейнером для основних сторінок, де відбуватиметься процес бронювання.

Дизайн елементів для вибору представлений у вигляді карток із заокругленими кутами так само як і в кнопок. Шрифт теж використовується плавний, без гострих кутів і засічок. Це зроблено для того, щоб додаток здавався максимально товариським до користувача.

З цього місця в додатку відбувається розподілення уловлювачів подій, так званих хендлерів, що реагують на дії користувача і відображають потрібний блок додатку.

Оскільки сайт не має звичної розгалуженої структури, де відбувається зміна цілої сторінки у відповідь на дії користувача, процес зміни блоків відбувається за допомогою стану додатку.

Кожному «екрану» присвоєно власне значення властивості display і в залежності від виконаної дії це значення записується в глобальний об’єкт state. Це запускає процес ререндерінгу компоненти MainPage і в залежності від поточного значення display відмальовується потрібний слайд. Приклад даної організації показано на рисунку 3.4.

```
return (
    <div className={style.wrapper} >
        {props.display.display === "info" && <InfoPage />}
        {props.display.display === "campus" && <Campus handlers={props.handlers}/>}
        {props.display.display === "type" && <TypePage handlers={props.handlers}/>}
        {props.display.display === "class" && <Classes type={props.classType.classType} handlers={props.handlers}/>}
        {props.display.display === "calendar" && <Calendar handlers={props.handlers}/>}
        {props.display.display === "confirm" && <ConfirmPage handlers={props.handlers}/>}
    </div>
);
```

Рисунок 3.4 – Організація перемикання "екранів"

Першою користувача зустрічає сторінка із загальною інформацією. Цей блок складається із двох частин: графіку роботи університету і контактних даних. У проєкті ж інформаційна сторінка розбита на дві компоненти Schedule та Contacts для зручності розробки і стилізації.

Кожен день у графіку реалізований за допомогою HTML тегу, що позначає список ul із вкладеними в нього елементами li які відображають день, час початку і час завершення роботи університету (рис.3.5).

Рисунок 3.5 – Графік роботи

Контактні дані – гіперпосилання, що ведуть до пошти, сайту університета, та Google Map за адресою розташування закладу. При наведені на будь-який елемент, текст змінює колір і виділяється підкреслення. Зроблено це для полегшення взаємодії користувачам (рис.3.6).

Рисунок 3.6 – Зміна кольору при наведенні

Початок роботи з додатком і пов’язаний з цим перехід до другого екрану відбувається при натисканні кнопки «Book Now» в компоненті Header. Після натискання у свою чергу запускається процес перезапису state і відмальовки нового блоку.

Слайди, які на пряму стосуються бронювання вгорі мають свого роду заголовки. Вони додані для відображення поточної стадії в роботі користувача.

Рисунок 3.7 – Заголовок слайду

Приклад на рисунку 3.7 показує, що користувач зараз знаходить на екрані вибору корпусу.

Вибір корпусу виконано у вигляді слайдеру з картками. Картки відображають один з п’яти корпусів та складаються із фото будівлі, назви та кнопки «Select» (рис.3.8).

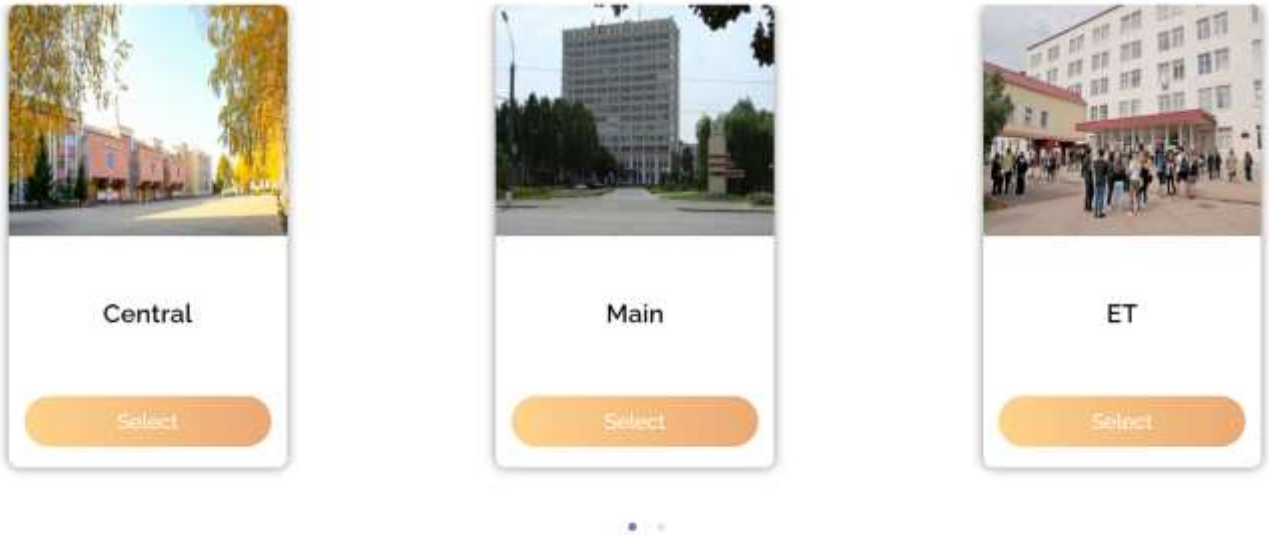

Рисунок 3.8 – Картки корпусів

Управління слайдером відбувається за допомогою маркерів знизу або свайпом за допомогою курсору.

Слайдер створений за допомогою зовнішньої бібліотеки «Alice carousel», що підключається до проєкту за допомогою пакетного менеджеру Node.js. Далі вона імпортується у потрібну компоненту через команду import і використовується як звичайна компонента.

Налаштування відбувається за допомогою властивостей, які прописуються в props тегу слайдера. Основною властивістю є items, з її допомогою передаються

елементи декомпонованого методом map масиву, що потім стануть картками. Приклад налаштування відображення показано на рисунку 3.9.

```
<AliceCarousel
    autoHeight
    mouseTracking
    items={classList}
    paddingLeft={100}
    responsive={responsive}
    controlsStrategy="responsive"
    disableButtonsControls={true}
/>
```

Рисунок 3.9 – Налаштування слайдера

Наступним кроком є розробка блоку з вибором типу аудиторії. Користувачу буде представлено на вибір два типи: лекційна та аудиторія для практичних занять.

Екран також представлений картками, але цього разу без слайдеру. Під назвою розташована піктограма будівлі і назва корпусу, обраного на минулому кроці.

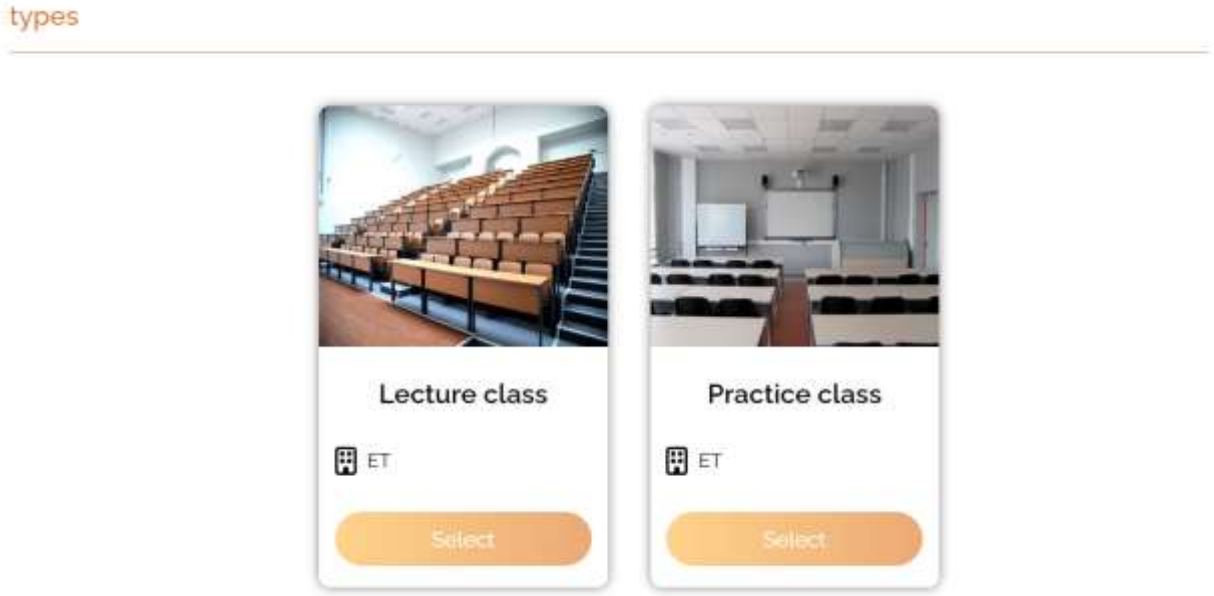

Рисунок 3.10 – Екран вибору типу аудиторії

Піктограми, як і слайдер, додаються за допомогою бібліотеки FontAwesome і подальшого додавання тегу FontAwesomeIcon з властивістю icon, де вказується назва і стиль відображення потрібної іконки.

При розробці цього додатку одразу стають зрозумілими переваги компонентного підходу до створення сайтів. Адже написану або підключену зовні один раз компоненту можна використовувати від сторінки до сторінки тільки коригуючи за допомогою props інформацію в ній. Це значно спростило і прискорило процес розробки екранів з вибором корпусів або аудиторій.

На рисунку 3.11 показано слайд з аудиторіями, створений за допомогою двох заздалегідь підготовлених компонентів: картки та слайдеру.

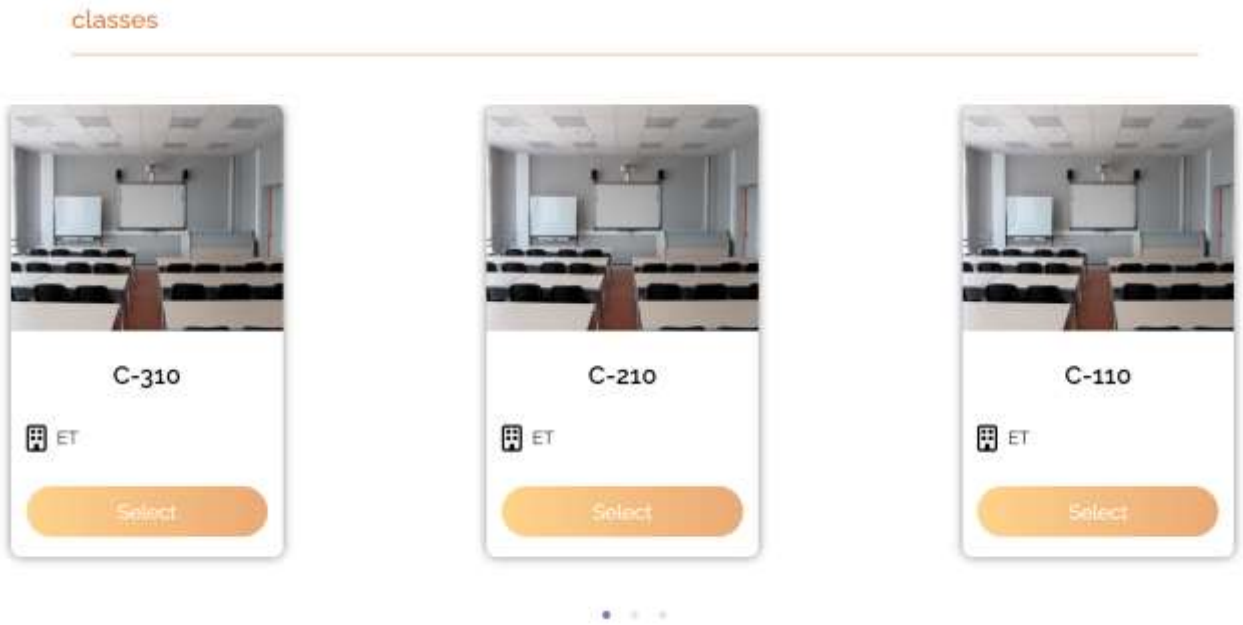

Рисунок 3.11 – Екран вибору аудиторії

Розробка календаря для вибору дати починається зі створення за допомогою HTML таблиці. Заголовком кожного стовпчика таблиці буде скорочена назва дня тижня. Субота і неділя додатково стилізовані іншим кольором, щоб показати, що у ці дні не можна виконати бронювання.

У комірках таблиці розміщені числа, що позначають дні місяця. При на веденні на число воно виділяється сірим овалом. Якщо натиснути на комірку

овал заповнюється градієнтом відповідно до глобального стилю додатку. Поточна дата в календарі позначається помаранчевим кольором. Вихідні дні або дні до поточної дати виділяються сірим кольором і недоступні для взаємодії. Для вимкнення усіх можливих типів взаємодії з об’єктом використовується CSS властивість pointer-events. У значенні none вона вимикає взаємодію з елементом.

Навігація між місяцями здійснюється за допомогою кнопок «Prev Month» для переключення на минулий місяць та «Next Month» – на наступний. Додатково до написів на кнопках додані також іконки, що позначають напрямок. Щоб підтримати концепцію мінімалізму в дизайні додатку, кнопки навігації представлені простими написами. Але при наведені спрацьовує виділення помаранчевим кольором. Загальний вигляд календаря представлено на рисунку 3.12.

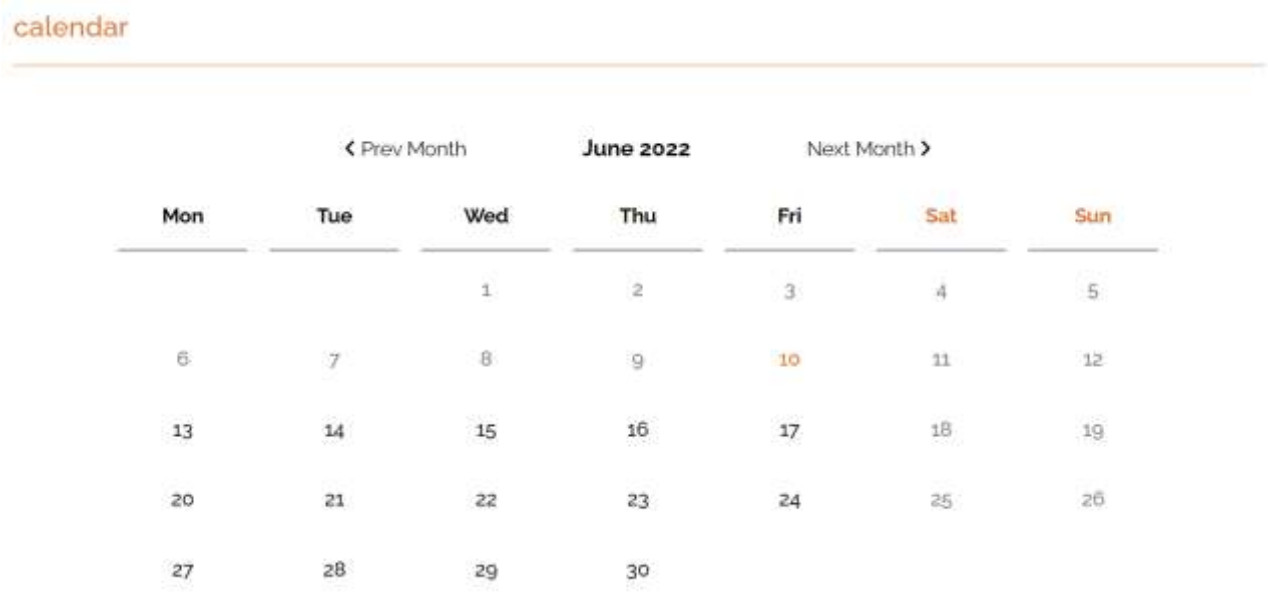

Рисунок 3.12 – Календар

Натискання на одну з комірок викличе відмальовку блоків з доступним для бронювання часом. Виконані вони у вигляді овалів з тонкою обводкою із вказаними всередині часом початку і закінчення. При наведенні обводка також зміню колір (рис.3.13).

Рисунок 3.13 – Функція вибору часу

Якщо на якийсь із доступних інтервалів часу вже є бронь, то блок з цим часом не з’явиться в переліку.

Кожен елемент цього переліку слугує також кнопкою для переходу на останній екран.

При запуску додатку ініціалізується пустий об’єкт. Потрібен він для передачі інформації про бронювання до бази даних. Після кожного користувацького вибору викликався метод, що додавав у цей об’єкт нову пару ключ значення. Де ключем є назва поля, а значення те, що обрав користувач. Наприклад після вибору аудиторії формується пара class: «M-100»

Сторінка підтвердження бронювання. На цій сторінці розміщено два поля для введення input з типом текст. Одне поле для імені та прізвища, інше для адреси електронної пошти. Обидва поля мають плейсхолдери з прикладами заповнення. При фокусі на одному з полів лінія під ним підсвічується. Зняття ж фокусу з поля викликає функцію, яка передає значення, введене в поле, до об’єкту, що буде переданий в базу даних.

Справа від інпутів знаходиться заключна інформація з типом обраної авдиторії, датою і часом на коли буде зроблена бронь і номером кабінету.

Під цим блоком знаходить кнопка підтвердження «Confirm booking». Натискання цієї кнопки запускає процес передачі обраної інформації до бази даних. Якщо ж користувач на даному етапі закриє додаток не натиснувши кнопку бронювання не буде збережено а інформація не буде записана до бази (рис.3.14).

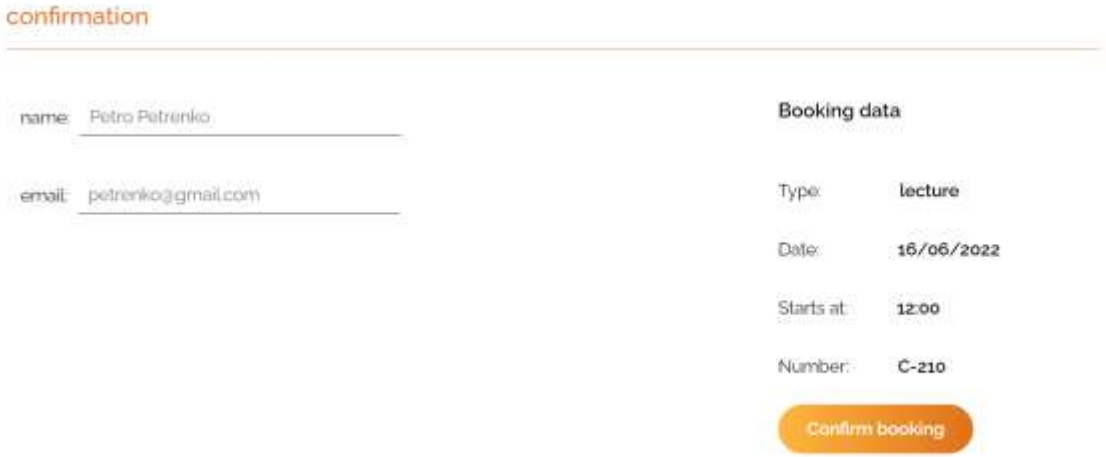

Рисунок 3.14 – Сторінка підтвердження

3.2. Розробка бази даних

Для керування базами даних додатку було обрано систему управління реляційними базами даних PostgreSQL. Головними причинами такого вибору стали вільне розповсюдження цього ПЗ та зручний і широкий набір інструментів для створення редагування та адміністрування баз даних та інформації, що вони зберігають.

Рисунок 3.15 –Логотип PostgreSQL

Для початку роботи з таблицями даних спершу потрібно на локальному сервері, що запускається при вході в додаток для керування, створити базу даних. Для цього обирається серед списку серверів, стандартно він один, потрібний, правою кнопкою миші викликається контекстне меню і обирається пункт Create > Database (рис.3.16).

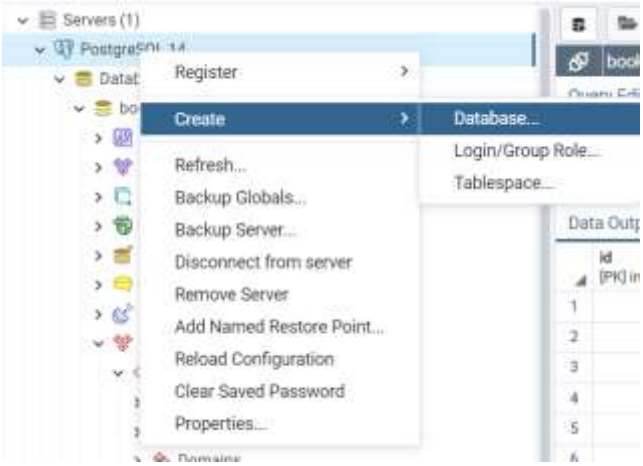

Рисунок 3.16 – Створення бази даних

Майже за таким же алгоритмом створюється в базі даних схема, де пізніше будуть розміщені таблиці з даними. Викликається контекстне меню і серед запропонованих варіантів для створення обирається Schema (рис.3.17).

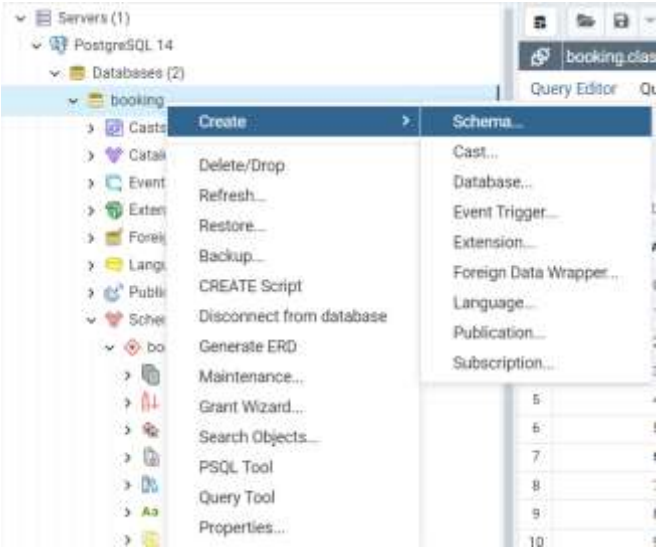

Рисунок 3.17 – Створення схеми

Далі в схемі обирається список з таблицями Tables і за допомогою контекстного меню створюється три таблиці (рис.3.18).

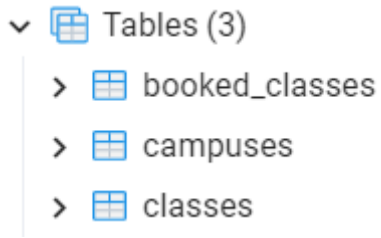

Рисунок 3.18 – Таблиці бази даних

Перша таблиця складається з дев’яти колонок. У неї записуватиметься інформація про заброньовані аудиторії, а саме: номер кабінету, корпус університету, де вона знаходиться, тип авдиторії, обрану дату, час початку, час закінчення, ім’я користувача та його пошта (рис.3.19). Спочатку таблиця пуста і буде заповнюватись даними по мірі використання додатку.

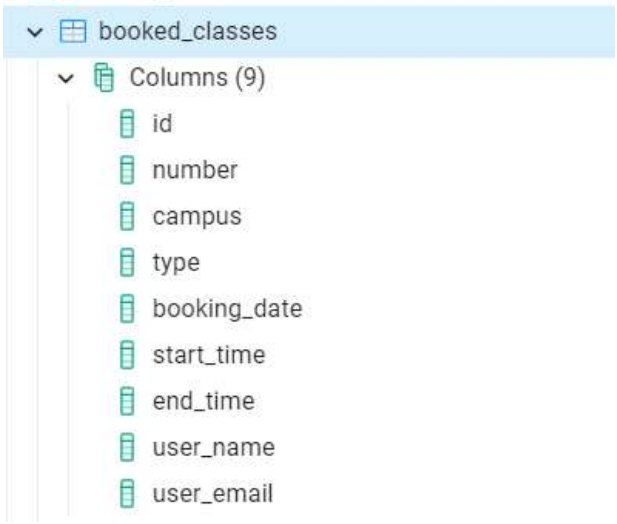

Рисунок 3.19 – Склад колонок в таблиці booked_classes

Таблиця campuses містить назви корпусів університету, де буде проходити бронювання. Складається вона всього з двох колонок id та назви корпусу.

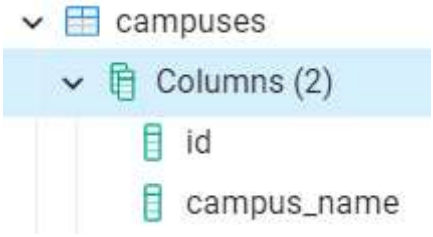

Рисунок 3.20 – Склад таблиці campuses

Остання таблиця це таблиця з аудиторіями. У ній зберігаються всі аудиторії, які можна забронювати у тих корпусах, що занесені до таблиці campuses. Інформація про аудиторії занесена у вигляді номеру, корпусу та типу, лекційна чи для практичних робіт.

	id [PK] integer	number character (10)	campus character (25)	type character (25)
1	1	C-204	Central	practice
2	2	C-208	Central	practice
3	3	C-209	Cenral	practice
4	4	C-215	Central	practice
5	5	C-216	Central	practice
6	6	C-217	Central	lecture
7	7	C-219	Central	lecture
8	8	C-220	Central	practice
9	20	ET-105	ET	lecture
10	21	ET-122	ET	lecture
11	0	C-201	Central	practice
12	9	C-221	Central	practice

Рисунок 3.21 – Таблиця classes

На рисунку 3.21 показана структура інформації, що зберігається в таблиці classes.

3.3. Розробка серверної частини

Призначення серверної частини – організація взаємодії між користувацьким інтерфейсом та базою даних.

Для реалізації бекенду було обрано мову програмування Java та Spring, фреймворк для створення веб-додатків на основі Java Enterprise. Набір інструментів для реалізації взаємодії із сервером і базою даних, що пропонує даний фреймворк значно прискорюють розробку додатку. Наприклад базові запити до таблиць даних можна описати за допомогою Java синтаксису без використання мови запитів SQL.

Починається розробка серверної частини із створення директорії із Java проєктом. Ініціалізація розпочинається з вибору бібліотеки за допомогою якої буде створюватись додаток та мови програмування (Java, Kotlin, Groovy) (рис.3.22).

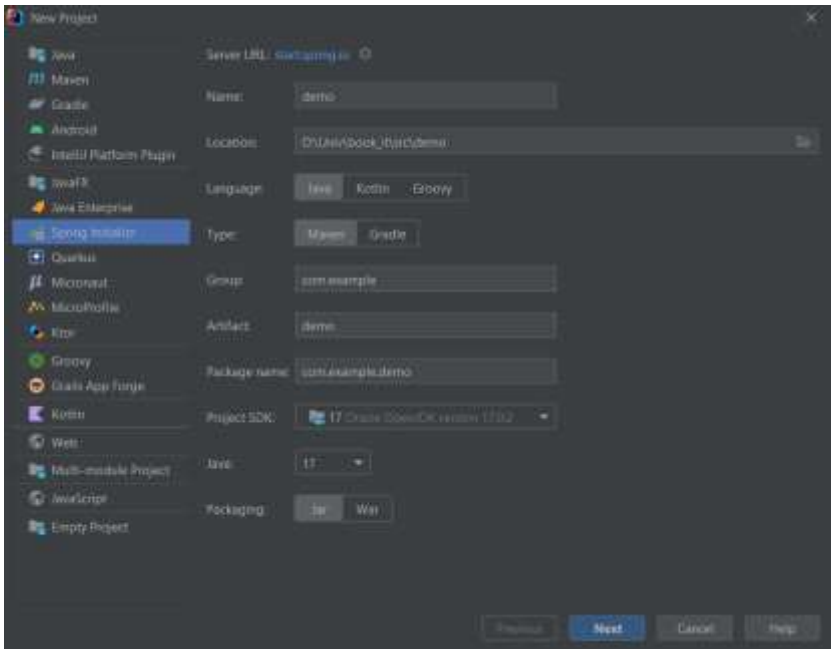

Рисунок 3.22 – Створення проекту

Наступним кроком є підключення залежностей, що використовуватимуться Spring бібліотекою. Для даного додатку обирається Spring Web, Spring Data JPA, Spring Boot DevTools, PostgreSQL Driver (рис.3.23).

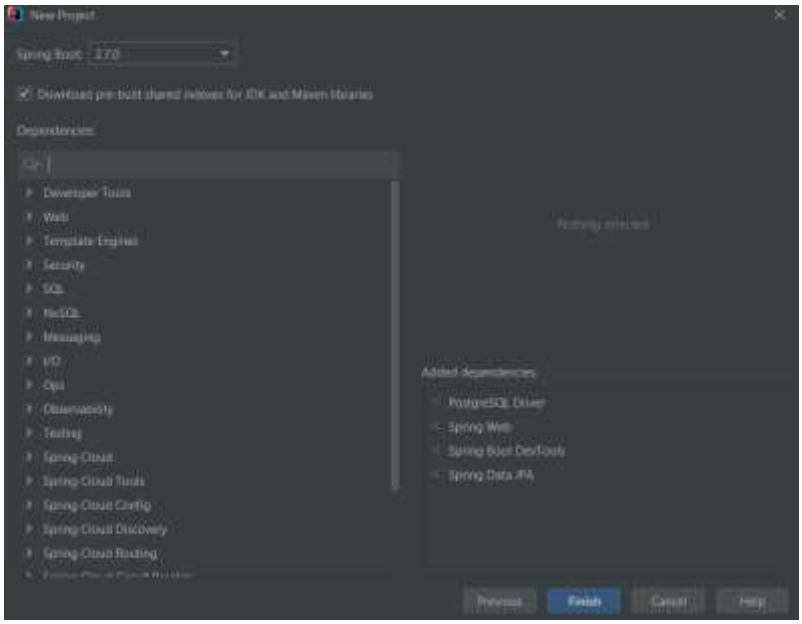

Рисунок 3.23 – Вікно підключення залежностей

Spring Web – містить у собі набір інструментів для розробки додатків з використання REST і використовує модель MVC (Model, View, Controller).

Spring Data JPA – бібліотека, що забезпечує спрощене налаштування взаємодії Java та баз даних. Створення підключення за допомогою цієї бібліотеки не потребує написання класів для конфігурації підключення, потрібно лише у файлі налаштування вказати дані для доступу.

Spring Boot DevTools – інструменти розробника які полегшують запуск додатку, а також забезпечують так званий LiveReload, тобто внесення змін викликає перезавантаження сторінки і вона відображається уже відредагована.

PostgreSQL Driver додає можливість підключатись до баз даних PostgreSQL використовуючи Java Data Base Driver.

Після того, як проект буде створений і всі залежності проініціалізовано, проводиться організація файлової структури додатку. Пакет серверної частини повинен включати директорію з контролерами, моделями та репозиторіями. Також було додано пакет для власноруч створених виключень, потрібно це для опрацювання різного роду ситуацій які можуть виникати при роботі з даними, але не покриватись стандартним набором виключень, що пропонує Java.

Робота з базою даних буде включати роботу з трьома таблицями: з аудиторіями, з корпусами та з забронюваними аудиторіями. Для цього у кожному із попередньо створених пакетів (model, controller, repository) створюється три файли, по одному на кожну таблицю. У цих файлах буде описана логіка взаємодії з даними.

Точкою входу в додаток є автоматично створений java-файл SpringBackendApplication. Кінцева структура продемонстрована на рисунку 3.24.

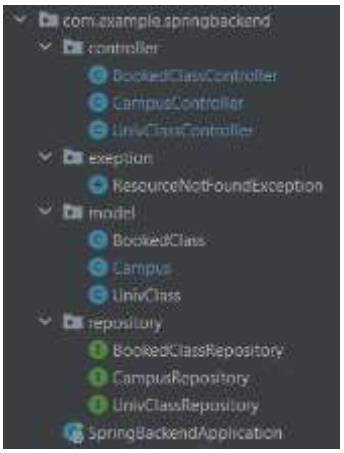

Рисунок 3.24 – Файлова структура додатку

У першу чергу описується логіка роботи з таблицею корпусів. Розпочинається процес зі створення класу Campus, що буде прототипом рядка даних в campuses. За допомогою Spring анотацій прописується, що даний клас є сутністю і вказується з якої таблиці брати інформацію. Оскільки в базі використовується це стандартна схема, додатково окрім назви таблиці, вказується і назва схеми. Якщо цього не зробити додаток не зможе отримати доступ до даних.

```
@Entity
@Table(name = "campuses", schema = "booking")
public class Campus {
```

Рисунок 3.25 – Анотації класу Campus

Описання полів класу і за сумісництвом колонок в таблиці відбувається також через анотацію. @Column вказує, що це поле відноситься до певної колонки, зв’язок задається за допомогою параметру name з іменем колонки, дані з якої поміщатимуться в дане поле. Також поле id додатково маркується анотаціями @Id та @GeneratedValue, що описує спосіб генерації ідентифікатору у разі додавання нового рядку в базу (рис.3.26).

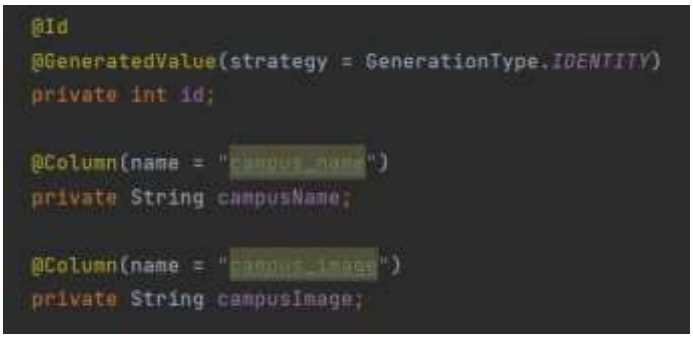

Рисунок 3.26 – Описання полів

Останнім кроком є генерація конструкторів, гетерів та сетерів, щоб забезпечити доступ до об’єктів класу та їх полів з інших частин додатку (рис.3.27).

Рисунок 3.27 – Конструктори, гетери, сетери

CampusRepository це інтерфейс, що наслідує JpaRepository і забезпечує основні операції по знаходженню, збереженню і видаленню даних, так звані CRUD операції [5]. При потребі список стандартних операцій можна розширити власнимим. Як наприклад це зроблено в UnivClassRepository, де було додано операцію пошуку за корпусом і типом (рис.3.28).

```
@Repository
public interface UnivClassRepository extends JpaRepository<UnivClass, Integer> {
    List<UnivClass> findByCampusAndClassType(String campus, String classType);
}
```

Рисунок 3.28 – Додаткова CRUD операція

Основним файлом для забезпечення роботи додатку з таблицею campuses є CampusController. Він пов’язує в собі логіку попередніх двох файлів та містить методи отримання даних з бази. При наявності клієнтської частини за допомогою анотації вказується з якої адреси можуть надходити запити. Відсутність цієї мітки спричинить помилку доступу. Також прописується за якою адресою можна отримати доступ до даного ресурсу (рис.3.29).

```
@CrossOrigin(origins = "http://localhost:3000")
@RestController
@RequestMapping("/api/v1/")
```

Рисунок 3.29 – Анотації налаштувань контролеру

В тілі класу-контролеру описуються методи доступу до інформації. Кожен метод маркується Анотацією, що вказує яким запитом здійснюватиметься доступ – GET чи POST та частина URL-адреси за якою можна отримати інформацію з таблиці. Якщо виклик методу потребує додаткових параметрів вони прописуються в адресній стрічці, а в анотації записуються у фігурних дужках (рис.3.30).

```
public class UnivClassController {

    @Autowired
    private UnivClassRepository univClassRepository;

    @GetMapping("/classes")
    public List<UnivClass> getAllClasses() { return univClassRepository.findAll(); }

    @GetMapping("/classes/{campus}/{classType}")
    List<UnivClass> findByCampusAndClassType(@PathVariable String campus, @PathVariable String classType) {
        return univClassRepository.findByCampusAndClassType(campus, classType);
    }

}
```

Рисунок 3.10 – Тіло класу-контролера

Робота над логікою доступу до таблиці campuses завершена. Для інших двох таблиць виконуються ті ж дії.

Після запуску проекту для перевірки його роботи можна здійснити перехід за посиланням http://localhost:8080/api/v1/classes. Перехід запустить процес отримання даних з таблиці classes і в результаті на екрані відобразяться всі аудиторії у вигляді масиву об’єктів з полями, що були описані в класі UnivClass (рис.3.31).

Рисунок 3.11 – Аудиторії з таблиці classes

Останнім кроком є підключення backend-частини додатку до користувацького інтерфейсу. Для цього в директорії з React-проектом створюється папка services. У цій папці знаходитимуться файли, що відповідають за надсилання запитів з клієнтської сторони на сервер.

Оскільки React нативно не підтримує роботу з HTTP запитами, для розширення його функціоналу потрібно встановити бібліотеку axios. Вона надає інструменти для відправки запитів та отримання відповідей із серверу.

Основна логіка описується у файлі BookingService. Перш за все створюються константи, що зберігатимуть адреси, за якими надсилатимуться запити (рис.3.32).

```
const BOOKING_API_BASE_URL = "http://localhost:8080/api/v1";
const BOOKING_API_CAMPUS_URL = BOOKING_API_BASE_URL + "/campuses";
const BOOKING_API_CLASS_URL = BOOKING_API_BASE_URL + "/classes";
const BOOKING_API_BOOKED_CLASS_URL = BOOKING_API_BASE_URL + "/booked_classes";
```

Рисунок 3.12 – Адреси доступу до серверу

Далі створюється клас BookingService. У тілі цього класу записані методи за допомогою яких користувач надсилатиме запити (рис.3.33).

Рисунок 3.13 – Тіло класу BookingService

Результатом роботи цих методів є об’єкт відповіді Response, який у випадку вдалої роботи поверне запитувані дані, а вразі – невдачі код помилки і помилку.

У файлі data.js описуються функції обробки отриманих відповідей і виокремлення з них даних, потрібних для відмальовки інтерфейсу. Також у цьому файлі генерується об’єкт з обраною користувачем інформацією, який потім за допомогою POST запиту буде записано в таблицю booked_classes.

Створені методи викликатимуться в коді компонент, де відбуватиметься рендеринг списків корпусів чи авдиторій, а також при підтвердженні користувачем броні.

Рисунок 3.14 – Код для відмальовки списку аудиторій

На рисунку 3.34 показано приклад відмальовки карток з аудиторіями за отриманими даними з таблиці classes.

3.4. Використання веб-додатку

Після запуску додатку, відображається стартова сторінка, що містить головну частину та блок з інформацією (рис.3.35).

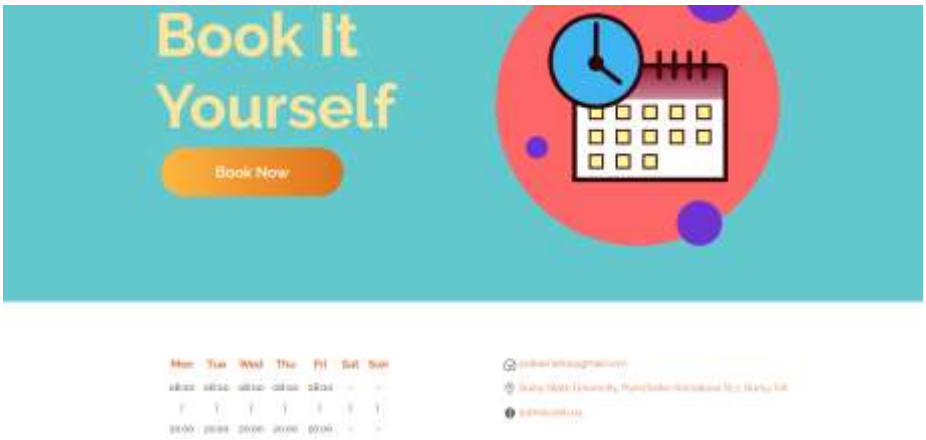

Рисунок 3.15 – Стартова сторінка

Після початку процесу бронювання відбувається перехід до екрану вибору корпусу. У цей час корпуси завантажуються з бази і показуються користувачу.

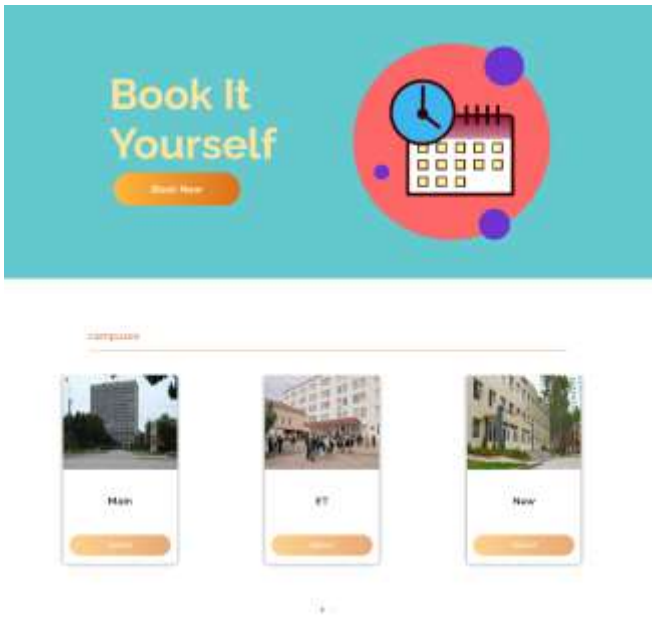

Рисунок 16 – Екран вибору корпусу

Після вибору корпусу, користувача переадресує на сторінку вибору типу авдиторі. Де буде запропоновано обрати або лекційну, або аудиторію для практичних занять (рис.3.37).

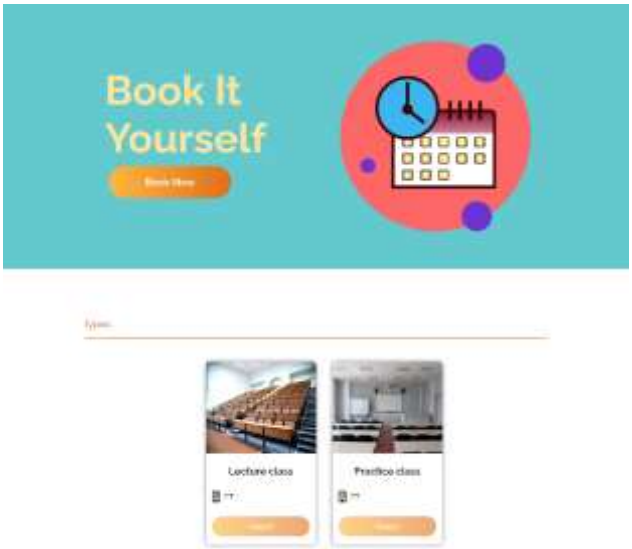

Рисунок 3.17 – Екран вибору аудиторій

Наступним етапом є вибір аудиторії. Як і у випадку з корпусами інформація про аудиторії підвантажується разом з переходом до слайду (рис.3.38).

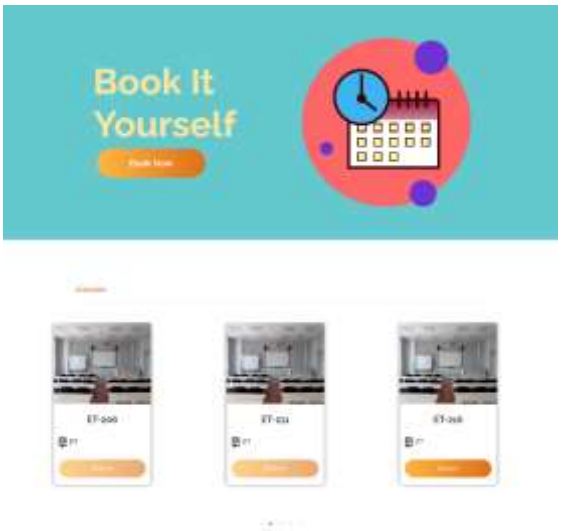

Рисунок 3.18 – Вибір аудиторії

Останнім екраном вибору є вибір дати та часу коли буде здійснена бронь (рис.3.39).

Рисунок 3.19 – Сторінка вибору дати й часу

Після вибору часу користувача переадресує на сторінку підтвердження. Де потрібно ввести своє ім’я та пошту і натисну кнопку підтвердження. Після на тискання відбудеться переадресація на стартову сторінку, а інформація записана в базу (рис.3.40).

Рисунок 3.20 – Запис в таблиці booked_classes

Перевірка запису в базі, показує, що додаток свої функції виконав. Вся потрібна інформація записана.

ВИСНОВКИ

Під час виконання бакалаврської роботи була поставлена та досягнута ціль створити веб-додаток для бронювання навчальних аудиторій в Сумському державному університеті.

Провівши аналіз предметної області для реалізації було обрано JavaScript бібліотеку React для клієнтської частини додатку, оскільки на відміну від конкурентів вона легка, бо не перевантажена великою кількістю інструментів, а потрібні пакети легко можна встановити у процесі розробки. І Java Spring Boot та PostgreSQL для організації взаємодії серверу і бази даних через сувору типізацію даних цієї мови програмування, велику кількість зручних інструментів для веб-розробки Spring та зручну систему керування базами даних PostgreSQL.

У ході роботи над додатком було створено макет за допомогою графічного редактору Figma, макет перенесено в проект у вигляді React-проекту, написана логіка взаємодії серверу і бази даних та налаштовано можливість відправляти HTTP-запити з клієнту.

У результаті було створено додаток у якому користувачі можуть централізовано проводити бронювання аудиторій університету для проведення занять або позанавчальних заходів.

СПИСОК ЛІТЕРАТУРИ

1. Single page application. A complete guide [Електронний ресурс] // AppCheck – Режим доступу до ресурсу: https://appcheck-ng.com/single-page-applications#
2. SQL чи NoSQL – ось в чому питання [Електронний ресурс] // Альтернативна наука – Режим доступу до ресурсу: https://alternativescience.net/programming/242-sql-chy-nosql-os-v-chomu-pytannya/
3. ReactJS State: SetState, Props and State Explained [Електронний ресурс] // SimpliLearn – Режим доступу до ресурсу: https://www.simplilearn.com/tutorials/reactjs-tutorial/reactjs-state#what_is_state_in_reactjs
4. WHAT IS CSS? [Електронний ресурс] // w3.org – Режим доступу до ресурсу: https://www.w3.org/Style/CSS/Overview.en.html
5. Spring Data JPA [Електронний ресурс] // Хабр – Режим доступу до ресурсу: https://habr.com/ru/post/435114/
6. SASS Documentation [Електронний ресурс] // sass-lang.com – Режим доступу до ресурсу: https://sass-lang.com/documentation/
7. Redux Documentation [Електронний ресурс] // react-redux.js.org – Режим доступу до ресурсу: https://react-redux.js.org/introduction/getting-started
8. React Documentation [Електронний ресурс] // uk.reactjs.org – Режим доступу до ресурсу: https://uk.reactjs.org/docs/getting-started.html
9. PostgreSQL 14.3 Documentation [Електронний ресурс] // postgresql.org – Режим доступу: https://www.postgresql.org/docs/current/
10. Java Documentation [Електронний ресурс] // docs.oracle.com – Режим доступу до ресурсу: https://docs.oracle.com/en/java/
11. Шилдт, Герберт. Java. Полное руководство, 10-е изд. : Пер. с англ. –СПб. : ООО «Диалектика», 2020. – 1488 с. : ил. – Парал. тит. англ.

12. Spring Boot [Електронний ресурс] // spring.io – Режим доступу до ресурсу: https://spring.io/projects/spring-boot
13. HTTP [Електронний ресурс] // mdn web docs – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/HTTP
14. Введение в REST API – RESTful веб-сервисы [Електронний ресурс] // Хабр – Режим доступу до ресурсу: https://habr.com/ru/post/483202/
15. Spring REST Docs [Електронний ресурс] // spring.io – Режим доступу до ресурсу: https://spring.io/projects/spring-restdocs#learn

ДОДАТОК А

А.1 Програмний код клієнтської частини

App.js

```
function App() {
    const dispatch = useDispatch();

    const displayInfoHandler = () => {
        dispatch(displayInfo());
    }

    const displayCampusHandler = () => {
        dispatch(displayCampus());
    }

    const displayTypeHandler = () => {
        dispatch(displayType());
    }

    const displayClassHandler = () => {
        dispatch(displayClasses());
    }

    const selectLectureHandler = () => {
        dispatch(selectLecture());
    }

    const selectPracticeHandler = () => {
        dispatch(selectPractice());
    }

    const displayCalendarHandler = () => {
        dispatch(displayCalendar());
    }

    const displayConfirmHandler = () => {
        dispatch(displayConfirm());
    }

    const mainPageHandlers = {
        displayInfoHandler,
        displayTypeHandler,
        displayClassHandler,
        selectLectureHandler,
        selectPracticeHandler,
        displayCalendarHandler,
        displayConfirmHandler
    }

    return (
        <div className="app">
            <Header displayType={displayCampusHandler}/>
            <MainPage handlers={mainPageHandlers}/>
        </div>
```

```
    );
}

export default App;
```

Header.jsx

```
const Header = (props) => {

    return (
        <div>
            <header className={style.header}>
                <div className={style.header__container}>
                    <h1 className={style.header__text}>Book It Yourself</h1>
                    <button className={style.header__button}
onClick={props.displayType}>Book Now</button>
                </div>
                <img className={style.header__logo} src={logoImage} alt="" />
            </header>
        </div>

    );
};

export default Header;
```

MainPage.jsx

```
const MainPage = (props) => {

    return (
        <div className={style.wrapper} >
            {props.display.display === "info" && <InfoPage />}
            {props.display.display === "campus" && <Campus
handlers={props.handlers}/>}
            {props.display.display === "type" && <TypePage
handlers={props.handlers}/>}
            {props.display.display === "class" && <Classes
type={props.classType.classType} handlers={props.handlers}/>}
            {props.display.display === "calendar" && <Calendar
handlers={props.handlers}/>}
            {props.display.display === "confirm" && <ConfirmPage
handlers={props.handlers}/>}
        </div>
    );

 }

    function mapStateToProps (state) {
        return {
            display: state.display,
            classType: state.classType
        }
    }

export default connect(mapStateToProps)(MainPage)
```

InfoPage.jsx

```
const InfoPage = () => {
    return (
        <div className={style.main}>
            <Schedule />
            <Contacts />
        </div>
    );
};

export default InfoPage;
```

Campus.jsx

```
const responsive = {
    0: { items: 1 },
    568: { items: 2 },
    1024: { items: 3 },
};

const Campus = (props) => {

    let campuses = getCampuses();

    const campusList = campuses.map((camp) => <Card title={camp.campusName}
                                                    cardType="campus"

displayPage={props.handlers.displayTypeHandler}
                                                setCamp={setBookingInfo}/>)

    return (
        <div className={style.container}>
            <UnderlinedText name="campuses"/>
            <AliceCarousel
                autoHeight
                mouseTracking
                items={campusList}
                paddingLeft={100}
                // paddingRight={200}
                responsive={responsive}
                controlsStrategy="responsive"
                disableButtonsControls={true}
            />
        </div>
    );
}
```

TypePage.jsx

```
const TypePage = (props) => {

    let bookingInfo = getBookingInfo();
    let campus = bookingInfo.camp;

    return (
        <div>
            <UnderlinedText name="types" />
            <div className={style.types}>
                <Card title="Lecture class" camp={campus} img={lecImg}
```

```
                    type="lecture"
displayPage={props.handlers.displayClassHandler}
                      selectType={props.handlers.selectLectureHandler}
setType={setBookingInfo}/>
                <Card title="Practice class" camp={campus} img={pracImg}
                    type="practice"
displayPage={props.handlers.displayClassHandler}
                      selectType={props.handlers.selectPracticeHandler}
setType={setBookingInfo}/>
            </div>
        </div>
    );
};

export default TypePage;
```

Classes.jsx

```
const responsive = {
    0: { items: 1 },
    568: { items: 2 },
    1024: { items: 3 },
};

const Classes = (props) => {

    let image;

    if(props.type === "lecture") {
        image = lecImg;
    } else {
        image = pracImg;
    }

    let bookingInfo = getBookingInfo();
    let campus = bookingInfo.camp;
    let classType = bookingInfo.type;

    let classData = getClasses(campus, props.type);
    const classList = classData.map((item) => <Card title={item.classNumber}
camp={campus} img={image}
                                                    type={item.type}
cardType="classes"

displayPage={props.handlers.displayCalendarHandler}
                                                    setClass={setBookingInfo}
/>)


    return (
        <div className={style.wrapper}>
            <UnderlinedText name="classes"/>
            <AliceCarousel
                autoHeight
                mouseTracking
                items={classList}
                paddingLeft={100}
                responsive={responsive}
                controlsStrategy="responsive"
                disableButtonsControls={true}
            />
```

```
        </div>

    );
}

export default Classes;
```

Calendar.jsx

```
export default class Calendar extends React.Component {
    static defaultProps = {
        date: new Date(),
        years: [2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031,
2032],
        monthNames: ['January', 'February', 'March', 'April', 'May', 'June',
'July',
            'August', 'September', 'October', 'November', 'December'],
        weekDayNames: ['Mon', 'Tue', 'Wed', 'Thu' , 'Fri', 'Sat', 'Sun'],
        onChange: Function.prototype
    };

    state = {
        date: this.props.date,
        currentDate: new Date(),
        selectedDate: null,
        bookingTime: []
    };

    get year() {
        return this.state.date.getFullYear();
    }

    get month() {
        return this.state.date.getMonth();
    }

    get day() {
        return this.state.date.getDate();
    }

    loadBookingTime(selectedDate) {
        let bookingInfo = getBookingInfo();
        let baseInfo = getBaseInfo();
        let time = calendar.getTime();
        let bookingTime = [];
        let registeredTime = [];

        // console.log(baseInfo[0].date);
        // console.log(selectedDate.toLocaleDateString('en-GB'));
        console.log(bookingInfo);

        baseInfo.forEach((elem) => {
            console.log(elem.date);
            if(elem.class === bookingInfo.class && elem.date ===
selectedDate.toLocaleDateString('en-GB')) {
                registeredTime.push({startTime: elem.startTime,
                                endTime: elem.endTime});
            }

        });
```

```
        time.forEach((timeElem => {
            if(registeredTime.length > 0 && timeElem.startTime !=
registeredTime[0].startTime) {
                bookingTime.push(timeElem);
            }
        }));

        if (bookingTime.length === 0) {
            return time;
        }
        return bookingTime;
    }

    handlePrevMonthButtonClick = () => {
        const date = new Date(this.year, this.month - 1);

        this.setState({ date });
    };

    handleNextMonthButtonClick = () => {
        const date = new Date(this.year, this.month + 1);

        this.setState({ date });
    };

    handleSelectChange = () => {
        const year = this.yearSelect.value;
        const month = this.monthSelect.value;

        const date = new Date(year, month);

        this.setState({ date });
    };

    handleDayClick = date => {
        this.setState({ selectedDate: date, bookingTime:
this.loadBookingTime(date) });

        //console.log(this.state);

        this.props.onChange(date);
        setBookingInfo('date', date);
    };

    render() {
        const { years, monthNames, weekDayNames } = this.props;
        const { currentDate, selectedDate } = this.state;

        const monthData = calendar.getMonthData(this.year, this.month);

        return (
            <div >
                <UnderlinedText name="calendar"/>
                <div className={style.container}>
                    <div className={style.calendar}>
                        <div className={style.header}>
                            <button onClick={this.handlePrevMonthButtonClick}>
                                <FontAwesomeIcon  className={style.angle_left}
icon={solid('angle-left')} />
                                Prev Month
                            </button>
                            <div className={style.title}>
```

```
                                <span>{monthNames[this.month]} </span>
                                <span>{this.year}</span>
                            </div>
                            <button
onClick={this.handleNextMonthButtonClick}>Next Month
                                <FontAwesomeIcon  className={style.angle_right}
icon={solid('angle-right')} />
                            </button>
                        </div>

                        <div className={style.table}>
                            <thead>
                            <tr>
                                {weekDayNames.map(name =>
                                    <th key={name} className={classnames(name
=== 'Sat' && style.weekend,
                                        name === 'Sun' && style.weekend
                                    )}>{name}</th>
                                )}
                            </tr>
                            </thead>

                            <tbody>
                            {monthData.map((week, index) =>
                                <tr key={index} className="week">
                                    {week.map((date, index) => date ?
                                        <td
                                            key={index}
                                            className={classnames(style.day,
                                                calendar.areEqual(date,
currentDate) && style.today,
                                                calendar.areEqual(date,
selectedDate) && style.selected,
                                                calendar.getDayOfWeek(date) ===
5 && style.inactive,
                                                calendar.getDayOfWeek(date) ===
6 && style.inactive,
                                                date <
currentDate.setHours(0,0,0,0) && style.inactive)}
                                            onClick={() =>
this.handleDayClick(date)}
                                        >{date.getDate()}</td>
                                        :
                                        <td key={index} />
                                    )}
                                </tr>
                            )}
                            </tbody>
                        </div>
                    </div>
                    <TimePicker handlers={this.props.handlers}
bookingTime={this.state.bookingTime}/>
                </div>

            </div>
        );
    }
}
```

ConfirmPage.jsx

```
const ConfirmPage = (props) => {
    let bookingData = getBookingInfo();

    return (
        <div className={style.confirm}>
            <UnderlinedText name="confirmation"/>
            <div className={style.container}>
                <InputFields />
                <BookingData type={bookingData.type} date={bookingData.date}
                             startTime={bookingData.startTime}
class={bookingData.class} displayPage={props.handlers.displayInfoHandler}/>
            </div>

        </div>
    );
}

export default ConfirmPage;
```

action.js

```
export function displayInfo() {
    return {
        type: DISPLAY_INFO,
        payload: "info"
    }
}

export function displayCampus() {
    return {
        type: DISPLAY_CAMPUS,
        payload: "campus"
    }
}

export function displayType() {
    return {
        type: DISPLAY_TYPE,
        payload: "type"
    }
}

export function displayClasses() {
    return {
        type: DISPLAY_CLASS,
        payload: "class"
    }
}



export function selectLecture() {
    return {
        type: SELECT_LECTURE,
        payload: "lecture"
    }
}

export function selectPractice() {
```

```
    return {
        type: SELECT_PRACTICE,
        payload: "practice"
    }
}

export function displayCalendar() {
    return {
        type: DISPLAY_CALENDAR,
        payload: "calendar"
    }
}

export function displayConfirm() {
    return {
        type: DISPLAY_CONFIRM,
        payload: "confirm"
    }
}
```

А.2 Програмний код серверної частини

BookedClassController.java

```
package com.example.springbackend.controller;

import com.example.springbackend.model.BookedClass;
import com.example.springbackend.repository.BookedClassRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@CrossOrigin(origins = "http://localhost:3000")
@RestController
@RequestMapping("/api/v1/")
public class BookedClassController {

    @Autowired
    private BookedClassRepository bookedClassRepository;

    @GetMapping("/booked_classes")
    public List<BookedClass> getAllBookedClasses() {
        return bookedClassRepository.findAll();
    }

    @PostMapping("/booked_classes")
    public BookedClass createBookedClass(@RequestBody BookedClass bookedClass) {
        return bookedClassRepository.save(bookedClass);
    }

    @DeleteMapping("/booked_classes/{date}/{startTime}")
    public ResponseEntity<Map<String, Boolean>> deleteBookedClass(@PathVariable
Date date, @PathVariable Date startTime) {
        BookedClass bookedClass =
bookedClassRepository.findByBookingDateAndStartTime(date, startTime);
```

```
        bookedClassRepository.delete(bookedClass);
        Map<String, Boolean> response = new HashMap<>();
        response.put("deleted", Boolean.TRUE);

        return ResponseEntity.ok(response);
    }

}
```

BookedClassRepository.java

```
package com.example.springbackend.repository;

import com.example.springbackend.model.BookedClass;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Date;

@Repository
public interface BookedClassRepository  extends JpaRepository<BookedClass,
Integer> {
    BookedClass findByBookingDateAndStartTime(Date date, Date startTime);
}
```

BookedClass.java

```
package com.example.springbackend.model;

import javax.persistence.*;
import java.util.Date;

@Entity
@Table(name = "booked_classes", schema = "booking")
public class BookedClass {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "number")
    private String classNumber;

    @Column(name = "campus")
    private String campus;

    @Column(name = "type")
    private String classType;

    @Column(name = "booking_date")
    private Date bookingDate;

    @Column(name = "start_time")
    private Date startTime;

    @Column(name = "end_time")
    private Date endTime;
```

```
    @Column(name = "user_name")
    private String userName;

    @Column(name = "user_email")
    private String userEmail;

    public BookedClass() {}

    public BookedClass(String classNumber, String campus, String classType, Date
bookingDate, Date startTime, Date endTime, String userName, String userEmail) {
        this.classNumber = classNumber;
        this.campus = campus;
        this.classType = classType;
        this.bookingDate = bookingDate;
        this.startTime = startTime;
        this.endTime = endTime;
        this.userName = userName;
        this.userEmail = userEmail;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getClassNumber() {
        return classNumber;
    }

    public void setClassNumber(String classNumber) {
        this.classNumber = classNumber;
    }

    public String getCampus() {
        return campus;
    }

    public void setCampus(String campus) {
        this.campus = campus;
    }

    public String getClassType() {
        return classType;
    }

    public void setClassType(String classType) {
        this.classType = classType;
    }

    public Date getBookingDate() {
        return bookingDate;
    }

    public void setBookingDate(Date bookingDate) {
        this.bookingDate = bookingDate;
    }

    public Date getStartTime() {
        return startTime;
```

```
    }

    public void setStartTime(Date startTime) {
        this.startTime = startTime;
    }

    public Date getEndTime() {
        return endTime;
    }

    public void setEndTime(Date endTime) {
        this.endTime = endTime;
    }

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getUserEmail() {
        return userEmail;
    }

    public void setUserEmail(String userEmail) {
        this.userEmail = userEmail;
    }
}
```

CampusController.java

```
package com.example.springbackend.controller;

import com.example.springbackend.model.Campus;
import com.example.springbackend.repository.CampusRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@CrossOrigin(origins = "http://localhost:3000")
@RestController
@RequestMapping("/api/v1/")
public class CampusController {

    @Autowired
    private CampusRepository campusRepository;

    @GetMapping("/campuses")
    public List<Campus> getAllCampuses() {
        return campusRepository.findAll();
    }

}
```

CampusRepository.java

```
package com.example.springbackend.repository;

import com.example.springbackend.model.Campus;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface CampusRepository extends JpaRepository<Campus, Integer> {

}
```

Campus.java

```
package com.example.springbackend.model;

import javax.persistence.*;

@Entity
@Table(name = "campuses", schema = "booking")
public class Campus {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "campus_name")
    private String campusName;

    @Column(name = "campus_image")
    private String campusImage;

    public Campus() {}

    public Campus(String campusName, String campusImage) {
        this.campusName = campusName;
        this.campusImage = campusImage;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getCampusName() {
        return campusName;
    }

    public void setCampusName(String campusName) {
        this.campusName = campusName;
    }

    public String getCampusImage() {
        return campusImage;
    }

    public void setCampusImage(String campusImage) {
        this.campusImage = campusImage;
```

```
    }
}
```

UnivClassController.java

```
package com.example.springbackend.controller;

import com.example.springbackend.model.UnivClass;
import com.example.springbackend.repository.UnivClassRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@CrossOrigin(origins = "http://localhost:3000")
@RestController
@RequestMapping("/api/v1/")
public class UnivClassController {

    @Autowired
    private UnivClassRepository univClassRepository;

    @GetMapping("/classes")
    public List<UnivClass> getAllClasses() {
        return univClassRepository.findAll();
    }

    @GetMapping("/classes/{campus}/{classType}")
    List<UnivClass> findByCampusAndClassType(@PathVariable String campus,
@PathVariable String classType) {
        return univClassRepository.findByCampusAndClassType(campus, classType);
    }

}
```

UnivClassRepository.java

```
package com.example.springbackend.repository;

import com.example.springbackend.model.UnivClass;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface UnivClassRepository extends JpaRepository<UnivClass, Integer> {
    List<UnivClass> findByCampusAndClassType(String campus, String classType);
}
```

UnivClass.java

```
package com.example.springbackend.model;

import javax.persistence.*;

@Entity
@Table(name = "classes", schema = "booking")
public class UnivClass {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "number")
    private String classNumber;

    @Column(name = "campus")
    private String campus;

    @Column(name = "type")
    private String classType;

    public UnivClass() {}

    public UnivClass(String classNumber, String campus, String classType) {
        this.classNumber = classNumber;
        this.campus = campus;
        this.classType = classType;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getClassNumber() {
        return classNumber;
    }

    public void setClassNumber(String classNumber) {
        this.classNumber = classNumber;
    }

    public String getCampus() {
        return campus;
    }

    public void setCampus(String campus) {
        this.campus = campus;
    }

    public String getClassType() {
        return classType;
    }
    public void setClassType(String classType) {
        this.classType = classType;
    }
}
```