

Міністерство освіти і науки України  
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»  
Кафедра мережних технологій

## **РОЗРОБКА АВТОМАТИЗОВАНОЇ СИСТЕМИ СКЛАДАННЯ РОЗКЛАДУ**

**Текстова частина до курсової роботи  
за спеціальністю «Інженерія програмного забезпечення» 121**

Керівник курсової роботи  
док. т.н., доц. Глибовець А.М.  
*(прізвище та ініціали)*

---

*(підпис)*

«1» травня 2020 р.

Виконав студент Усачов К.Ю.  
*(прізвище та ініціали)*

Київ 2020

Міністерство освіти і науки України  
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»  
Кафедра мережних технологій факультету інформатики

ЗАТВЕРДЖУЮ  
Зав. кафедри мережних технологій,  
доктор фіз.-м.н., професор.  
\_\_\_\_\_ Г. І. Малашонок  
(підпис)  
«5» березня 2020 р.

**ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**  
на курсову роботу

студенту 1-го курсу МП, факультету інформатики  
Усачову Кирилу Юрійовичу

Дослідити алгоритми автоматичної генерації розкладу та створити власну систему, що реалізує даний алгоритм.

Вихідні дані:

- Застосунок –прототип власної системи;
- Текстова частина курсової роботи.

Зміст ТЧ до магістерської роботи:

Індивідуальне завдання

Вступ

1 Огляд наявних технологій

2 Розробка власної системи

3 Практичне застосування

Висновки

Список літератури

Дата видачі «5» березня 2020 р. Керівник \_\_\_\_\_  
(підпис)

Завдання отримав \_\_\_\_\_  
(підпис)

<b>Вступ .....</b>	<b>4</b>
<b>1 Дослідження .....</b>	<b>5</b>
1.1 Концепція .....	5
1.2 Дослідження наявних засобів .....	6
1.3 Задача .....	8
1.4 Алгоритм .....	9
1.5 Реалізація .....	11
1.5.1 Технічні засоби.....	11
1.5.1.1 JavaScript Angular .....	11
2 Технічна частина реалізації.....	19
2.1 Фронтенд.....	19
2.1.1 Структура клієнтського застосунку.....	20
2.1.2 Оформлення.....	24
2.1.3 Аутентифікація .....	25
2.2 Захист даних.....	26
2.3 Інтерфейс прикладного програмування сервісу.....	26
2.4 Бекенд .....	28
2.5 Висновки.....	33
3 Практичне використання .....	34
3.1 Запуск.....	34
3.2 Висновки до частини 3 .....	36
4 Висновки.....	37

## **Вступ**

Розвиток ІТ-індустрії сьогодні відкриває двері до зручностей, які раніше були можливі лише в теорії. Одним із таких благ є автоматизація роботи.

Можливість перекласти частину своєї роботи на алгоритм або програму дає користувачам можливість приділити більше уваги іншим аспектам своєї роботи.

Дана робота поділена на три розділи. У першій частині описується алгоритм та використані технології, обґрунтовуються їх переваги та доцільність. Другий розділ містить опис реалізації компонентів системи, як сервісної її частини, так і частини представлення. Фінальний розділ описує практичне використання системи.

Практичне значення проведеної роботи полягає у дослідженні описаного класу задач, виявленні особливостей реалізації алгоритмів їх розв'язку, порівняння наявних рішень-аналогів та досвіді, отриманому під час створення свого власного рішення.

## **1 Дослідження**

### **1.1 Концепція**

Мету роботи автора є створення системи, яка б дозволяла автоматично утворювати розклад занять для університету. Дана задача вимагає реалізації наступних пунктів:

Дослідження наявних систем-аналогів. Для розробки будь-якої системи необхідно мати уявлення про вже наявні рішення. Більшість актуальних проблем уже мають рішення або, принаймні, для них описано шляхи їх реалізації. Задача програміста – розглянути переваги та недоліки наявних рішень та розглянути принципи їх роботи.

Створення сервісу, що реалізує серверну частину системи. Необхідно мати сервіс, до якого у користувачів системи буде можливість звертатися із запитом та отримувати у відповідь результати.

Створення клієнту для зручного користування сервісом. Більшість людей, на яких орієнтований застосунок не зобов'язані мати уявлення про прикладний інтерфейс застосунку, для більш зручного користування необхідний простий і доступний графічний інтерфейс, який дозволить абстрагувати всю взаємодію з сервісом від користувачів.

Забезпечення прикладного інтерфейсу взаємодії з сервісом. Для того, щоб мати можливість працювати з сервісом віддалено і незалежно від клієнтської частини, необхідно мати зовнішній інтерфейс для спілкування з застосунком.

Упровадження системи зберігання готових розкладів. Для уникнення необхідності копіювати результат роботи сервісу, необхідно середовище, де можна буде зберігати дані та діставати їх за необхідності у будь-який момент.

## 1.2 Дослідження наявних засобів

Автором було проведено аналіз готових рішень, які можна знайти в Інтернеті та які є доступними для скачування в публічному доступі.

Програма AVTOR.

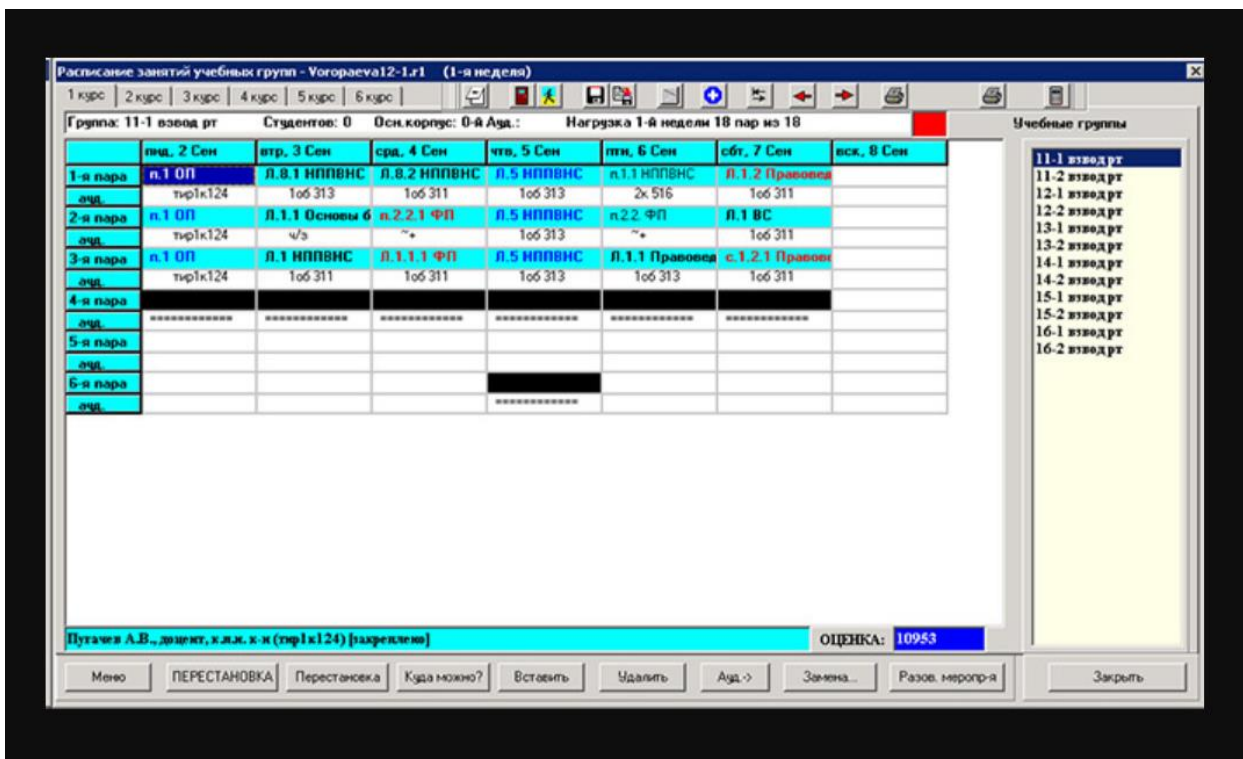


Рисунок 2.1.1 – інтерфейс AVTOR

Можливості:

- створення графіку для студентів/викладачів;
- отримання інформації про учбовий план;
- коригувати параметри оптимізації алгоритму;
- будувати розклад без вікон для учбових груп;
- задавати додаткові параметри, такі як час переходу між корпусами, комбіновані заняття тощо.

## Програма aSc Timetables

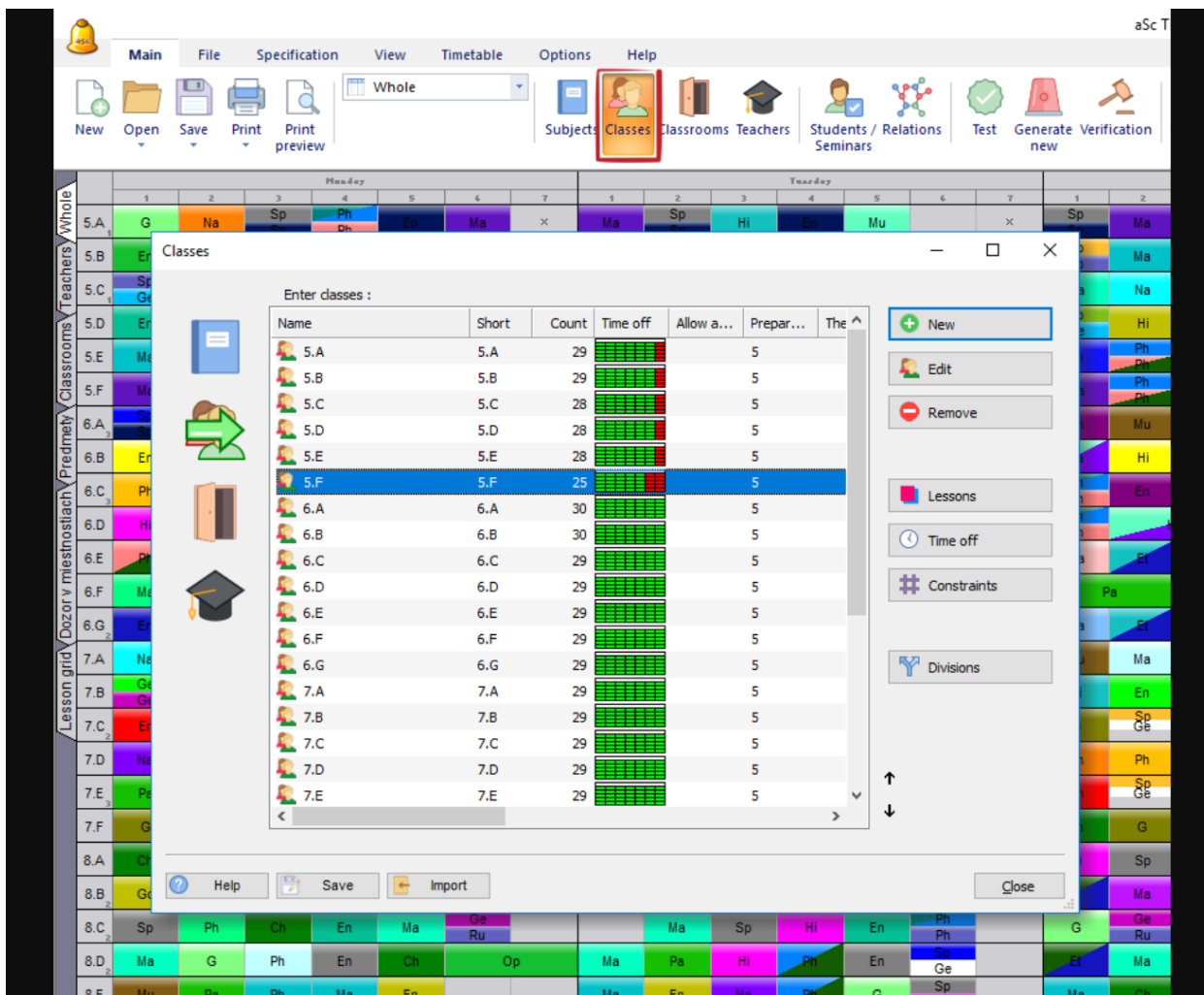


Рисунок 2.1.2 – інтерфейс aSc Timetables

Можливості:

- Завдання детальної інформації про викладачів/учителів;
- Уточнення інформації про проведення учбового процесу:
  - поділ на групи
  - поділ на курси

- уточнення інформації про аудиторії будівлі, час проведення, побажання щодо щільності проведення занять тощо.

### 1.3 Задача

Проблема генерування розкладу відноситься до області теорії розкладів – розділу науки, присвяченому календарному плануванню. Задачі подібного типу часто виникають при плануванні промислових процесів, але типовою є задача складання розкладу проведення занять з урахуванням моменту проведення заняття, учасників (групи та викладача) і місця проведення (аудиторії).

На жаль більшість задач в рамках теорії потребують перебору великого набору варіантів, який може зростати залежно від кількості задач. Крім того, реалізація розв’язків даних задач на практиці, як правило, потребує введення додаткових обмежень.

Складання розкладу відноситься до категорії NP-повних задач. Це означає, що для даної задачі існує відповідь «чи можна її розв’язати», при цьому складність задачі відноситься до рівня NP, тобто для якої можна створити недетерміновану послідовність дій, яка може вирішити дану задачу за поліноміальний час, тобто  $\mathcal{P} \subseteq \mathcal{NP}$ .

Формулюючи задачу неформально, потрібно побудувати розклад занять для вищого навчального закладу, тобто побудувати множину кортежів із значень для

- часу проведення;
- предмету;
- викладача даного предмету;



- групи даного предмету;
- аудиторії.

Тобто розв'язком даної задачі є множина  $\{T, C, I, G, A\}$ , де значеннями є відповідні значення з доменів часу, предмету, викладача, групи та аудиторії відповідно, для якої для різних значень  $T$  не існує перетинів серед множин значень  $C, I, G$  та  $A$ .

При цьому очевидно, що залежно від наданих вхідних даних задача сформулювати розклад може не мати повного рішення. Якщо рішення існує, знайти його оптимальним шляхом не є тривіальною задачею, і не існує поки алгоритму, здатного сформулювати розклад краще за «природні» евристичні людські алгоритми. Через це використання автоматизації для подібної задачі не можна повноцінно вважати заміною ручної роботи, а скоріше, доповненням для механічного спрощення.

Для реалізації сервісної частини застосунку було використано алгоритм, який намагатиметься сформулювати частковий розклад, у випадку неспроможності сформулювати повноцінний неперетин для заданих значень.

## 1.4 Алгоритм

Внутрішня взаємодія програми відбувається завдяки спілкуванню «наглядачів» (observers). Кожен наглядач є сутністю, яка представляє категорію, значення об'єктів якої не має перетинатися для заданого часу. Відповідно, сутності, представлені в програмі:

InstructorObserver – викладач;

GroupObserver – група;

ClassObserver – заняття (для уникнення потенційної плутанини зі словом “course”);

ClassroomObserver – аудиторія.

Кожен «наглядач» володіє інформацією про свій об’єкт, та може надавати усім інформацію про зайнятість в даний час. Викладач дізнається у групи інформацію, чи вільна вона в конкретний час, якщо ні, обирається інший час, коли група вільна. Аналогічно відбувається перевірка інформації для аудиторії (програма вважає, що в одній аудиторії може проводитися одночасно лише одна пара).

Додаткове обмеження на аудиторію: чи підходить аудиторія для проведення лекцій. Лекції потребують достатнього простору, але який саме простір підходить для проведення лекцій – залежить від вузу, тому замість встановлення точних розмірів, аудиторія має параметр, що визначається самим користувачем (isLectureSuitable) на основі його суб’єктивної оцінки. При цьому алгоритм врахує, що практичні заняття, які потребують менше простору, можуть бути проведені будь-де, натомість лекції можуть бути проведені лише у відповідних лекційних аудиторіях (рисунок 1.4.1).

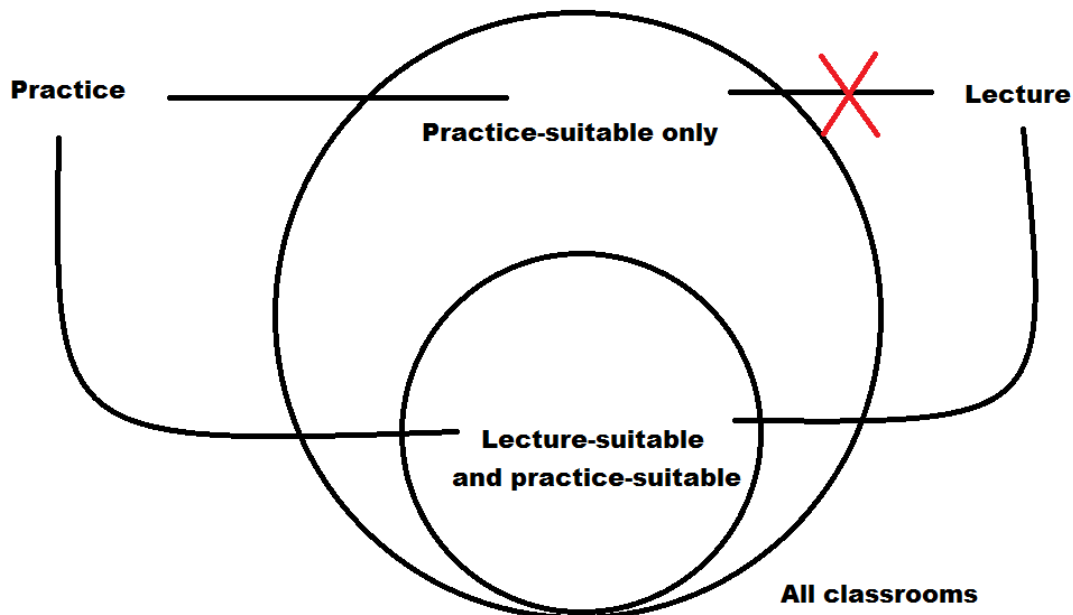


Рисунок 1.4.1 – відповідність типів аудиторій

Фінальний етап алгоритму – розміщення всіх нерозміщених занять. У разі якщо якісь заняття було відкинуто алгоритмом (або за будь-яких інших причин), утворений розклад можна буде відредагувати вручну. Для цього сервіс передбачає процедуру оновлення збереженого розкладу, а фронтенд-клієнт дозволяє безпосередньо вносити зміни до таблиці і надсилати їх до сервісу.

Далі наведено опис технологій фронтенду та бекенду, та їх реалізація.

## 1.5 Реалізація

### 1.5.1 Технічні засоби

#### 1.5.1.1 JavaScript Angular



Рисунок 1.5.1.1 – Angular

Для розробки клієнтського застосунку було використано фреймворк Angular (рисунок 1.5.1.1) мови JavaScript.

Angular – відкрита платформа для створення клієнтських веб-застосунків. Розроблена Google.

Являється спадкоємцем AngularJS – схожої платформи від тієї ж самої команди розробників, але написаної на чистому JavaScript. Angular, на відміну від попередньої версії, написаний із використанням TypeScript, має іншу архітектуру, у зв'язку з чим отримав змінену назву для уникнення плутанини. Веде свою нумерацію версій, починаючи з другої.

Актуальна версія на момент написання роботи – 9.1.6.

Відмінності від аналогічних рішень (React, Vue):

- Використання реального DOM;

- велика кількість вбудованих функцій;
- взаємозв'язність усіх функцій фреймворка;
- типізація;
- Angular-language-service (забезпечує автозаповнення шаблонів компонентів);
- одностороння прив'язка даних.

Дане рішення є повноцінним фреймворком, адже реалізовано за допомогою інверсії управління: код Angular виконується самостійно, надаючи розробнику можливість вбудувати у завчасно передбачені місця свою власну логіку, яка буде відпрацьовувати відповідним чином.

Даний фреймворк побудований на компонентному підході: веб-сторінка розбивається на окремі елементи, для кожного з яких передбачено:

- шаблон (файл з HTML-розміткою);
- таблиця стилів;
- програмний код компонента (TypeScript-файл, де задається поведінка даного компонента);
- файл з тестами коду компонента.

Чітка структура проекту на цій платформі дозволяє робити проект масштабованим. Наявні вбудовані функції дозволяють уникнути підключення до проекту сторонніх рішень, які можуть бути недостатньо сумісними одне з одним.

Статична типізація дозволяє уникнути багатьох помилок на етапі написання коду, що робить фреймворк складнішим для вивчення, порівняно з аналогами, але більш надійним для розробки великих та середніх проектів.

### 1.5.1.2 Java Spring Boot



Рисунок 1.5.1.2.1 – Spring Boot

Spring Boot (рисунок 1.5.1.2.1) являється середовищем на основі Java с відкритим вихідним кодом. Основне призначення платформи – створення мікросервісів. Розроблений Pivotal Team.

Також являється фреймворком: розробник лише програмує свої класи, анотуючи їх відповідним чином, щоб передати управління ними в Spring під час роботи.

Прив'язаність до XML-конфігурацій, які використовує Spring Framework, є надмірною, що послугувало виникненням Spring Boot, який дозволяє автоматизувати конфігурування і звести роботу з на лаштунками проекту до мінімуму.

Дозволяє розробникам самостійно створювати та розгортати сервіси, кожен з яких може бути власним процесом, що забезпечує полегшену модель для підтримки за стосунків.

Особливості Spring Core можна застосовувати в будь-якому Java-застосунку.

Актуальна версія на момент написання роботи – 5.2.2.

Переваги, що виділяють платформу серед рішень-аналогів:

- простота у розгортці;
- гарна базова підтримка масштабованості;
- мінімальне конфігурування і, як наслідок, менший час на створення;
- можливість створювати застосунок як набір слабкозв'язаних компонент, що полегшує введення нових функцій, незалежно від основної логіки взаємодії програми з іншими її компонентами.

Фреймворк надає стандартні налаштування «з-під коробки», але й при цьому може бути швидко зміненим у випадку, якщо зміни вимог до проекту починають відрізнятися від значень за промовченням.

Для зручності розробки платформа надає ряд допоміжних засобів, таких як:

- метрики;
- безпека;
- зовнішня конфігурація;
- вбудований сервер;

Мова розробки застосунків на Spring Boot – Java, що забезпечує рішення, засновані на цій платформі, кросплатформенністю, швидкодією, статичними перевітками типів та відносною простотою.

Використана версія Java під час роботи – 11.0.7.

### **1.5.2 Збереження даних**

Як було зазначено у меті проекту, система має забезпечити зберігання даних, що дозволить переглядати та модифікувати раніше створені розклади.

Для того, щоб обрати сховище, яке б підходило до створюваної системи найкраще, потрібно визначити структуру, відповідно до якої зберігати розклад буде найбільш доцільно.

#### **1.5.2.1 Рішення на основі реляційних баз даних**

Розклад має містити інформацію про дні, кожен день містить від нуля до  $n$  занять. Заняття пов'язано з викладачем, групою, аудиторією та часом проведення. Реляційний підхід вимагає виділення викладачів, груп та аудиторій в окремі сутності. Як наслідок, для формування відповіді на запит до наявного розкладу, потрібно буде провести з'єднання декількох таблиць, і проводити такі з'єднання для кожного заняття, присутнього в розкладі. Щоб забезпечити першу нормальну форму бази, необхідно виокремити інформацію по кожному дню/заняттю, створивши окремі таблиці з зовнішніми ключами до кожного такого запису. Отже сумарно в базі даних буде присутня значна кількість дрібних даних, що сповільнить роботу з БД під час пошуку відповідних частинок одного заняття одного розкладу. Автономність великої кількості дрібних даних для даної задачі не є виправданою, адже для створення нового розкладу буде використовуватися новий набір викладачів/аудиторій, що нівелює переваги повторного використання посилань на вже присутні записи.



Ще одна проблема, що витікає з наявності великої кількості дрібних даних: складність підтримки консистентності даних. Приклад:

*Користувач створює розклад, у якому фігурує певна аудиторія*

*Інший користувач створює розклад для іншого вузу, у якому фігурує аудиторія з таким же номером (рисунок 1.5.2.1.1).*

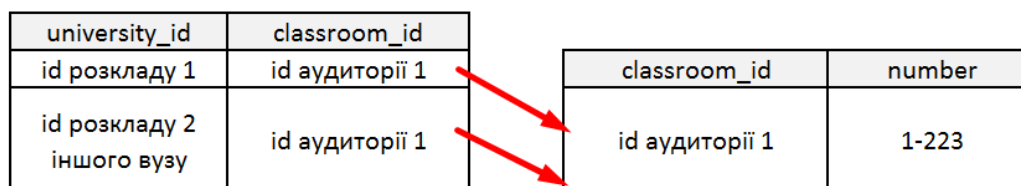


Рисунок 1.5.2.1.1 – помилкова зв'язність

Якщо вважати однією сутністю аудиторію з одним і тим же номером, проблем не буде до тих пір, доки один з користувачів не захоче відредагувати розклад, змінивши номер цієї аудиторії, і в такому випадку номер зміниться в усіх розкладах, що є некоректною поведінкою (рисунок 1.5.2.1.2):

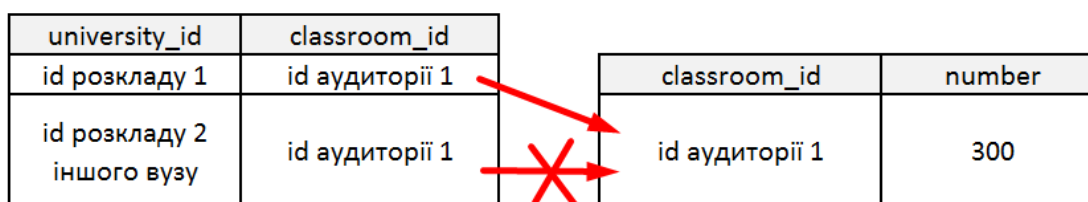


Рисунок 1.5.2.1.2 – некоректне оновлення

Щоб цього уникнути, потрібно буде зберігати інформацію кожного нового розкладу автономно, тобто база даних буде містити великий об'єм інформації, який натомість буде відноситися лише до одного розкладу, що нівелює цінність зв'язків і реляційного підходу організації даних.

#### **1.5.2.2 Рішення на основі нереляційних баз даних**

Розклад можна зберігати як єдиний документ, який не містить зовнішніх зв'язків. Переваги такого підходу:

- можливість зберігання розкладу відносно нефіксованої структури (якщо в якийсь день заняття в певний час відсутнє, не потрібно зберігати порожнє фіктивне значення для нього, можна зберігати лише наявні дані);
- простота оновлення (достатньо знайти документ; усі дані, з якими він пов'язаний, містяться при ньому);
- швидкість операцій (відсутність з'єднань та перевірок).

У загальному випадку подібний підхід має і недолік: можливість покласти в сховище невалідні дані. Однак, цей недолік нівелюється у випадку, коли можливість створення нових даних обмежена з боку користувача і виконується лише через посередника (у нашому випадку це сервіс), який за визначеним алгоритмом зберігає дані лише визначеної структури.

Для реалізації системи було обрано нереляційну базу даних MongoDB.

MongoDB – NoSQL база даних, працює на основі документів. Дані зберігаються у форматі BSON (Binary JSON). Даний формат дозволяє здійснювати більш швидкий пошук та обробку даних.

Схема даних реалізується на рівні застосунку, а не бази даних, тобто БД може зберігати дані різної форми. Про забезпечення цілісності дбає розробник, а не сама база, перевірки, притаманні реляційним БД, відсутні, через що операції виконуються швидше.

Актуальна версія на момент розробки – 4.2.0.



Рисунок 1.5.2.2.1 – MongoDB

## **2 Технічна частина реалізації**

### **2.1 Фронтенд**

Клієнтський застосунок являє собою веб-сторінку, побудовану за принципом SPA (single-page application). Така модель надає наступні переваги:

- чітке розділення фронтенду та бекенду, на відміну від багатосторінкового застосунку, де кожна сторінка генерується окремо за запитом до серверу;
- зменшене навантаження на сервер (виходячи з попереднього пункту);
- можливість протестувати сервіс та клієнт окремо один від одного;
- низька зв'язність з бекендом, що надає можливість для створення багатьох клієнтських застосунків.

Недоліком даного підходу є гірша пошукова здатність (порівняно з МРА), адже веб-сторінка надходить користувачу порожньою, і лише після завантаження основних модулів JavaScript, починається динамічна генерація контенту (рисунок 2.1.1), що ускладнює аналіз таких сторінок. Тим не менша для даного класу задач це не є критичним.

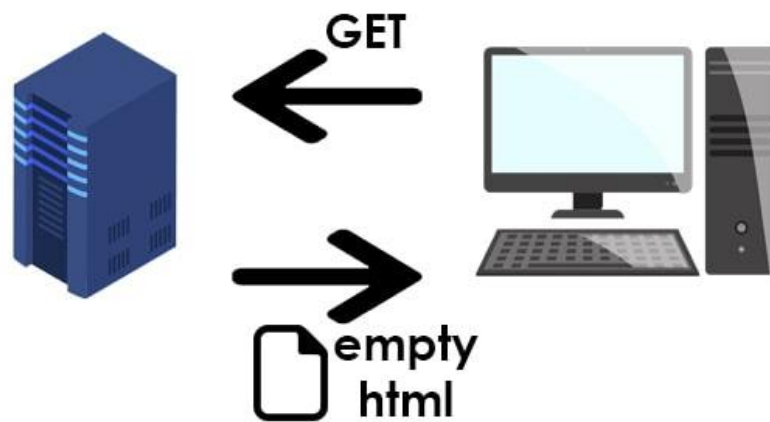


Рисунок 2.1.1 – Взаємодія SPA

### 2.1.1 Структура клієнтського застосунку

Застосунок на Angular реалізує компонентний підхід побудови сторінки: окремі елементи, які являються логічними частинами інтерфейсу виділяються в окремі модулі. Кожному модулю відповідає шаблон сторінки та код, який вбудовується в основний код фреймворку, після чого Angular виконує динамічний рендерінг компоненту в тих випадках, коли його поява передбачена логікою програми. Оскільки Angular бере на себе всю роботу по відтворенню сторінки на основі компонентів, відключення JavaScript робить користування сторінкою неможливим.

Нижче наведено опис основних компонентів програми

AppComponent – обов'язковий батьківський компонент програми, усі інші компоненти є його елементами. У випадку багатокомпонентної архітектури, як правило, є порожнім, або містить у своїй розмітці елементи, що мають обов'язково бути присутніми на всіх сторінках, що є нечастим випадком. У розробленому застосунку є порожнім, і лише містить спеціальний компонент RouterApplet.

RouterApplet – вбудований спеціальний елемент, який дозволяє виконувати перехід між усіма іншими компонентами програми. Для цього сторінка має містити набір шляхів, кожному з яких відповідає певний компонент, який буде генерувати сторінку при переході за даним шляхом. Надалі до кожного компоненту буде прописаний його шлях.

Шлях / – HomeComponent – компонент головної сторінки застосунку. Містить панель навігації, футер та текст вітання (рисунок 2.1.1.1):

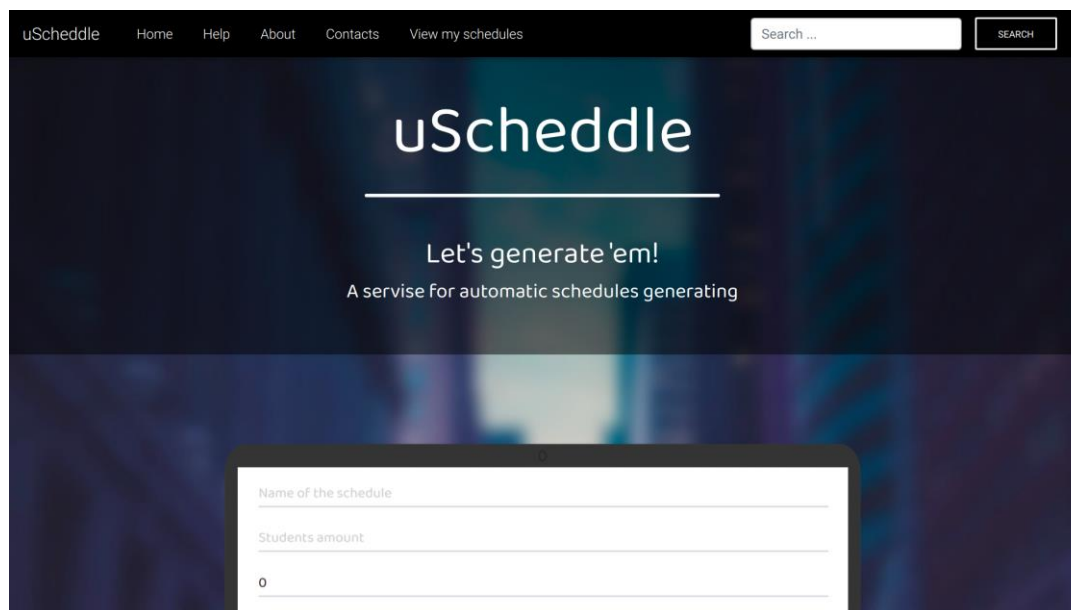


Рисунок 2.1.1.1 – Інтерфейс клієнту

Панель навігації та футер дублюються на всіх стрінках, тому далі вони не їх присутність не зазначено.

Шлях /schedlist – SchedlistComponent – компонент, що відповідає за відображення розкладів, які даний користувач згенерував за допомогою цього сервісу (рисунок 2.1.1.2):

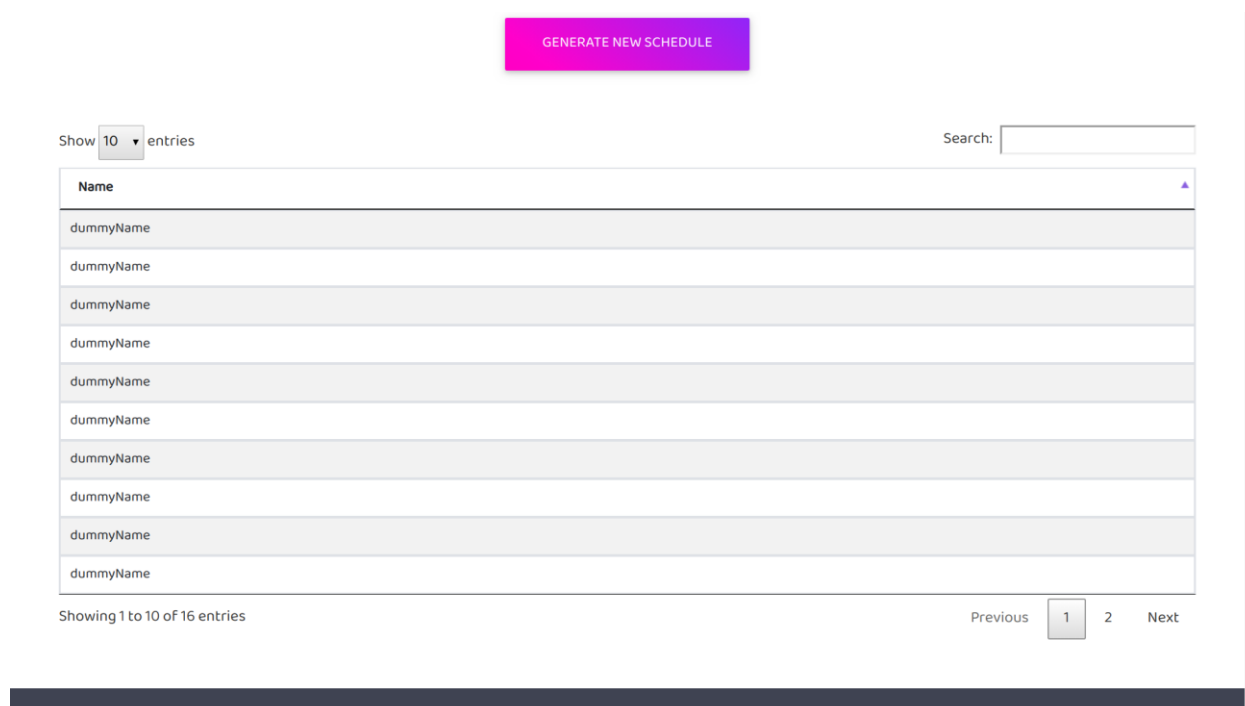
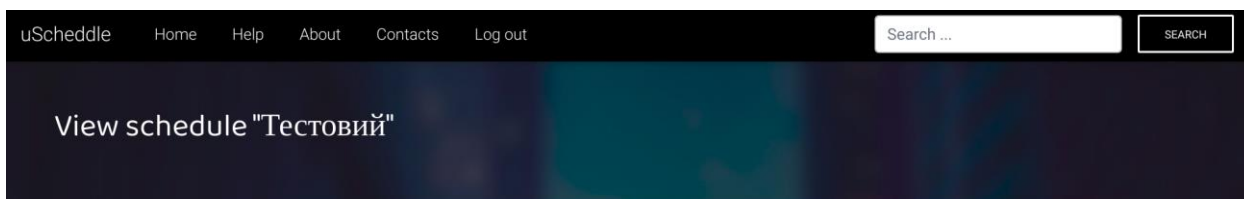


Рисунок 2.1.1.2 – Список розкладів

Кожен розклад можна редагувати та видаляти.

Шлях /schedview/:id – SchedviewComponent – сторінка перегляду самого розкладу. Параметр URL id вказує ідентифікатор розкладу.

Розклад відображається у вигляді таблиці. Демонстрацій вигляд урізаної таблиці (рисунок 2.1.1.3):



Mon					
Time	Course	Instructor	Group	Weeks	Classroom
13:30-15:00	Базн даних	Гулаєва Н.М.	1	1-14	1-223
14:00-15:20	Java	Глибовець А.М.	2	1-14	1-225
Tue					
Time	Course	Instructor	Group	Weeks	Classroom
13:30-15:00	Базн даних	Гулаєва Н.М.	1	1-14	1-223
14:00-15:20	Java	Глибовець А.М.	2	1-14	1-225
Wed					

Рисунок 2.1.1.3 – Таблиця розкладу

Шлях /main-request – MainRequestComponent – сторінка, де можна задати параметри для генерації розкладу (рисунок 2.1.1.4).

 The image shows a web form titled 'Test schedule'. It contains several input fields: a text field for 'Test course' with the value 'Lecture', a text field for 'Instructor' with the value 'John Doe', and a text field for 'Classroom' with the value '100'. Below these fields are two large, rounded rectangular buttons: a blue one labeled 'ADD A COURSE' and a green one labeled 'ADD A CLASSROOM'. At the bottom of the form is a purple button labeled 'GENERATE'.

Рисунок 2.1.1.4 – Заповнення даних

Шлях /searchlist – SearchlistComponent – Сторінка відображення результатів пошуку розкладу у панелі пошуку. Інтерфейс ідентичний SchedlistComponent.

Також присутні два ненавігаційних елементи, які є частинами всіх сторінок:

NavbarComponent – панель навігації. Містить основні посилання та пошукову панель (Рисунок 2.1.1.5).

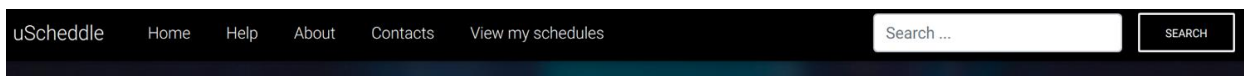


Рисунок 2.1.1.5 – Панель навігації

FooterComponent – футер, нижній елемент сторінки (рисунок 2.1.1.6).

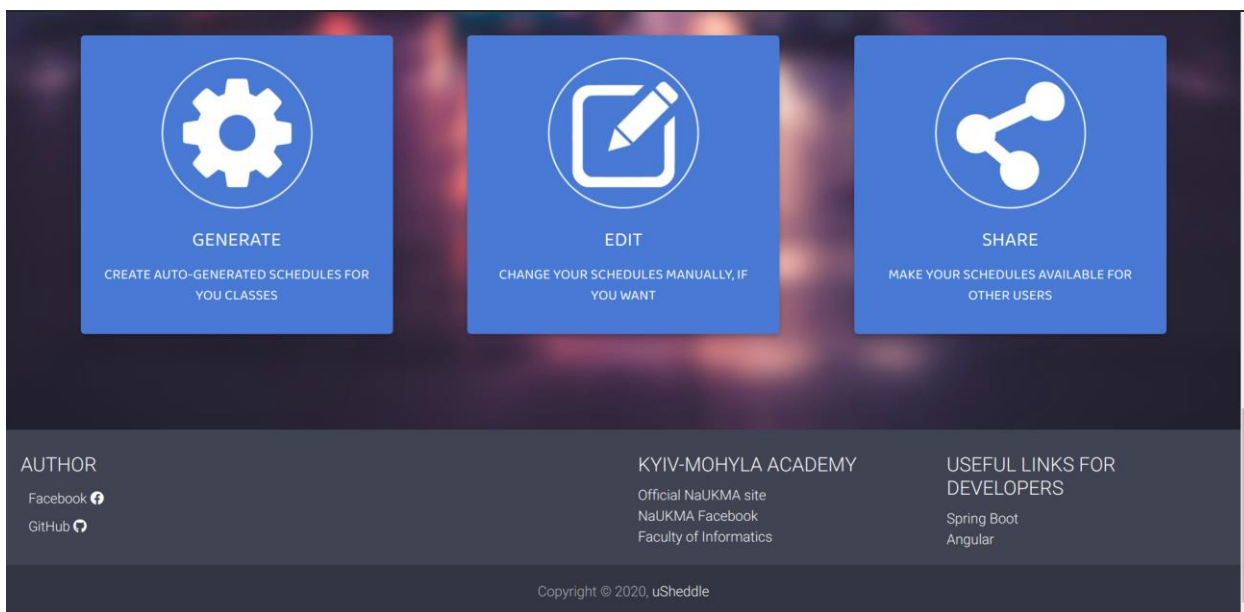


Рисунок 2.1.1.6 – Інші елементи сторінки

## 2.1.2 Оформлення

Сайт оформлено у стилістиці Material Design. Для стилізації використано фреймворк Bootstrap та його розширена версія: MaterialDesign Bootstrap.



### 2.1.3 Аутентифікація

Для того, щоб мати можливість зберігати розклад за певним користувачем, необхідно мати можливість ідентифікувати даного користувача. Задля цієї мети в системі запроваджено аутентифікацію.

Для того, щоб забезпечити передачу користувачем своїх паролей до захищених ресурсів безпосередньо у застосунок, аутентифікація була зроблена через OAuth – схему авторизації, завдяки якому є можливість передати третій стороні (у даному випадку нею є наш застосунок) без необхідності передавати паролі і інші чутливі дані безпосередньо третій стороні.

У якості авторизатора обрано Microsoft, аутентифікація на сайті вважається успішною, коли пройшла успішна аутентифікація за допомогою аккаунту Azure AD (Office 365, Microsoft 365, Azure portal).

Сценарій аутентифікації:

1. Користувач на сайті своєю дією звертається до ресурсів, які потребують авторизованого доступу.
2. Angular перехоплює цю дію і перенаправляє користувача на сторінку microsoft.online.
3. Після успішної аутентифікації, microsoft.online перенаправляє користувача до ресурсу, який він потребував.

Будь-який момент користувач може вийти зі свого акаунту, натиснувши кнопку Log out на панелі навігації. Кнопка доступна лише якщо користувач уже

попередньо пройшов аутентифікацію (рисунок 2.1.3.1).



Рисунок 2.1.3.1 – Вихід з системи

## 2.2 Захист даних

Після генерації розкладу, система зберігає інформацію про автора в базі даних.

Для того, щоб мати можливість змінити або видалити розклад, клієнт має надіслати із запитом ідентифікаційну інформацію, у разі неспівпадіння якої операцію буде відхилено.

Під час перегляду розкладів, клієнт буде автоматично визначати, чи може даний користувач редагувати/видаляти цей розклад. У разі наявності відповідних прав, сторінка буде містити кнопки для оновлення та видалення розкладу.

## 2.3 Інтерфейс прикладного програмування сервісу

Взаємодія з сервісом відбувається посередком зовнішнього API.

Взаємодія побудована за принципом RESTful. Сервіс не зберігає стан взаємодії з користувачем між запитами, натомість якщо є потреба в інформації, що ідентифікує користувача, ця інформація передається разом із кожним запитом.

Уся взаємодія RESTful API зводиться до чотирьох операцій:

- Додавання нових даних на сервер (Create);

- Отримання даних (**R**ead);
- Модифікація наявних даних (**U**ppdate);
- Видалення даних (**D**eleate).

Усі запити відносяться до однієї з цих чотирьох категорій.

Передбачено наступні запити:

URL	Опис
/generate	Запит на генерацію нового розкладу
/read/{id}	Запит на отримання розкладу за його ідентифікатором
/search?name={name}	Запит на отримання розкладу за наданим ім'ям
/schedules	Запит на отримання всіх розкладів, згенерованих даним користувачем
/edit/{id}	Запит на редагування розкладу за його ідентифікатором
/delete/{id}	Запит на видалення розкладу за його ідентифікатором

Як видно з таблиці, операція отримання розкладу може бути викликана безпосередньо.

Усі інші операції потребують авторизації. Для того щоб співставити себе з користувачем в системі, необхідно надіслати токен з ідентифікатором (рисунок 2.3.1), отриманим ззовні, а саме Microsoft Account ID.

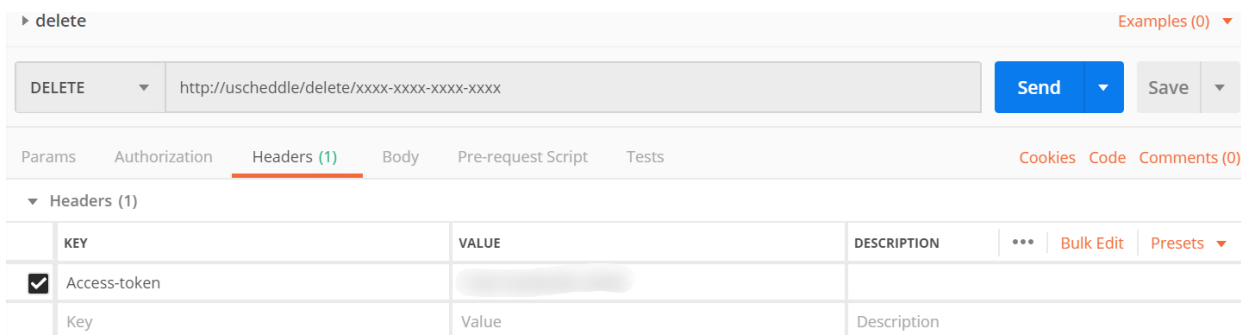


Рисунок 2.3.1 – Access token

Таким чином забезпечується обмеження прав на розклади: змінити або видалити чужий розклад не вдасться, є можливість лише переглянути його.

## 2.4 Бекенд

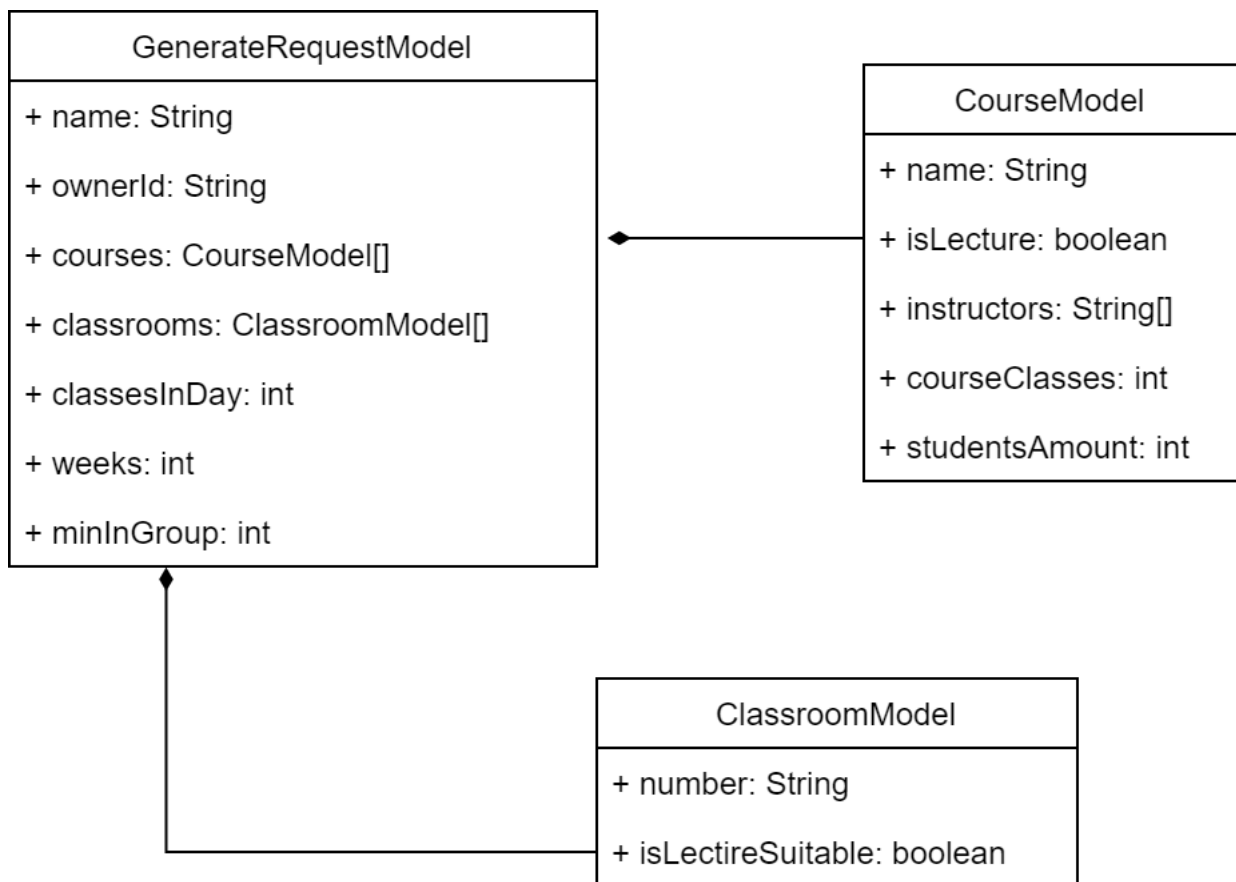
Структуру сервісу можна представити чотирма модулями:

- model
- repository
- resources
- solving

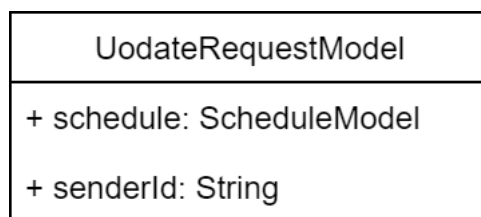
Модуль model відповідає за представлення даних. Дані поділяються на вхідні, вихідні, та дані рівня сховища даних.

Підпакет request містить класи, якими може бути представлено запити, що надходять від користувача:

Структура моделі відправки запиту на генерацію розкладу:

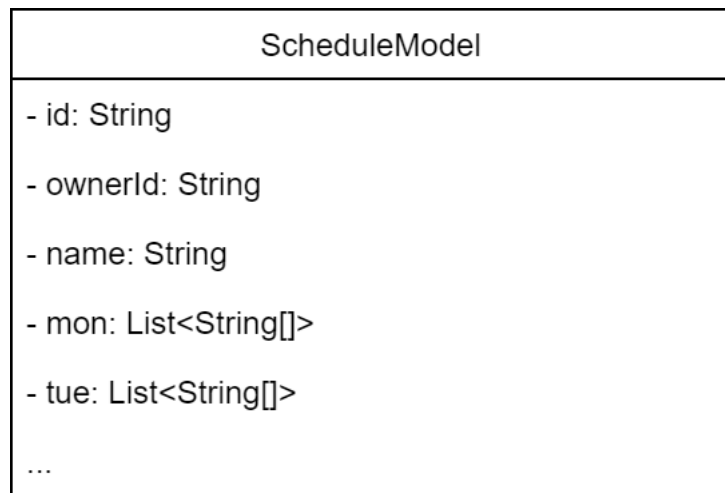


Структура моделі відправки запита на оновлення:



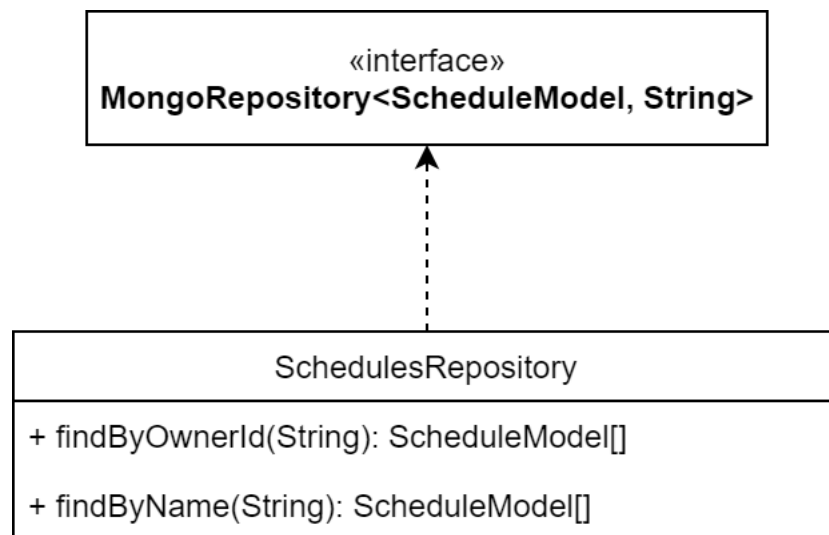
Модель ScheduleModel являється репрезентацією моделі документу для розкладу. Слід зауважити, що база даних завжди має схему, але реалізовуватися ця схема може як в самій базі даних, так і на рівні застосунку. У даному випадку модель розкладу є ні чим іншим, як моделлю для структури, яка буде зберігатися у документах БД.

Структура ScheduleModel:



Розклад має назву та власника, а також для днів тижня містить репрезентацію занять для кожного дня. Обмеження на структуру задаються у запиті (кількість пар для кожного дня може бути розподілена лише в межах тієї кількості пар, яка може бути максимально в одному дні).

Пакет repository представлений інтерфейсом SchedulesRepository:



Інтерфейс розширює MongoRepository, що дозволяє використовувати даний інтерфейс для організації доступу до даних. Визначено операції пошуку за власником та ім'ям, що на сторінці дозволяє здійснювати відображення

списку розкладів конкретного користувача, а також здійснювати пошук розкладів за іменем.

Пакет resource представлений класом `DataAccessRestController`. Даний клас містить набір методів, що відповідають методам API. У процесі роботи основного потоку програми контролер «очікує» надходження даних до відповідних шляхів, прив'язаних до методів:

```
@RequestMapping(value = "/generate", method = RequestMethod.POST,
    consumes = {MediaType.APPLICATION_JSON_VALUE},
    produces = {MediaType.APPLICATION_JSON_VALUE})
public @ResponseBody ResponseEntity<ResponseModel> create(
    @RequestBody final GenerateRequestModel request);
```

Поле анотації `consumes` вказує на структуру, яку повинно мати тіло запита, у разі невідповідності користувач отримує помилку 400 (Bad request).

У разі співпадіння структури тіла, фреймворк намагатиметься співставити поля наданого JSON атрибутам класу запиту, у разі відсутності того чи іншого поля, його значення буде встановлено в значення за промовчанням для даного типу даних.

У разі помилкового запиту буде згенеровано відповідь, що включає в себе інформацію про помилку. У разі успішного створення розкладу можлива лише одна відповідь: OK, status 200.

Можливі відповіді, які може продукувати контролер:

Відповідь	Коментар	Реакція клієнту
OK	Тіло відповіді містить інформацію про	Клієнт переходить на сторінку відображення

	згенерований розклад, у випадку оновлення тіло містить посилання на цей же розклад	розкладу за отриманим ідентифікатором, у випадку оновлення клієнт оновлює поточну сторінку перегляду розкладу
BAD REQUEST	Тіло містить інформацію про некоректність запиту; може бути відтворено при використанні API у чистому вигляді, веб-клієнт не дасть відтворити дану помилку завдяки валідації	Передбачено виведення помилки про необхідність перевірити правильність дій або зв'язатись з технічною підтримкою
FORBIDDEN	Може бути отримано під час несанкціонованої спроби змінити або видалити чужий розклад; помилка не відтворювана через створений клієнт (не передбачено кнопок та елементів інтерфейсу, які можуть спровокувати таку помилку)	Передбачено виведення помилки про необхідність перевірити правильність дій або зв'язатись з технічною підтримкою



## **2.5 Висновки**

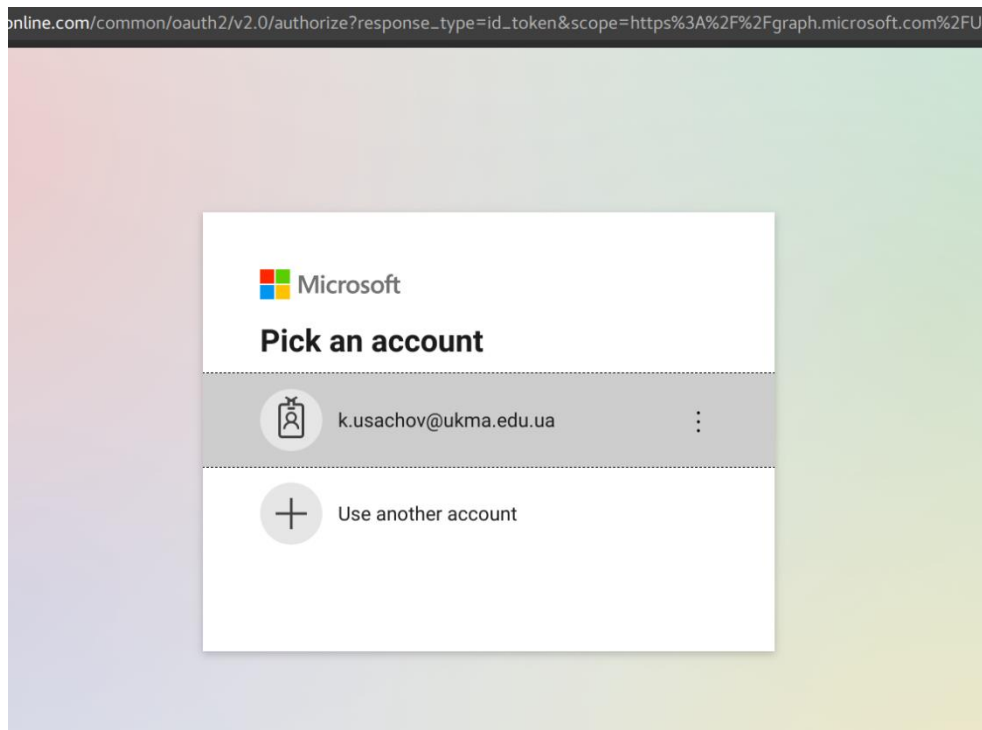
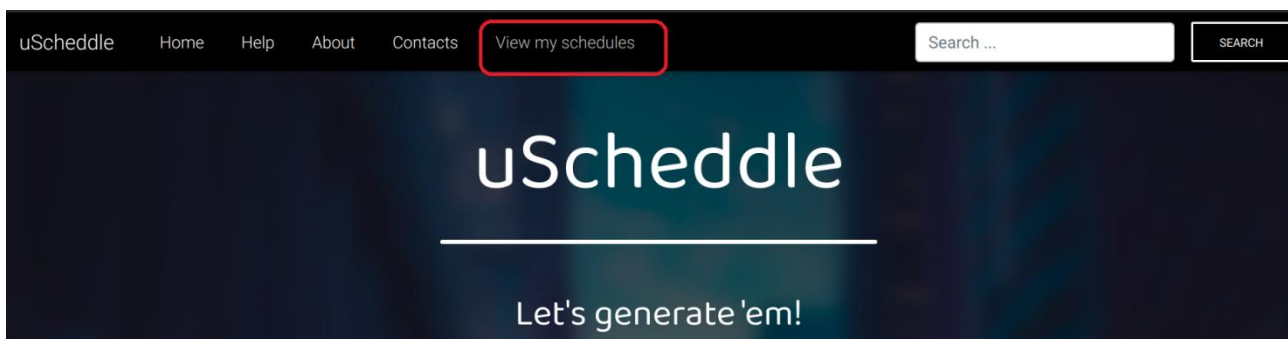
Було досліджено етапи розробки застосунку, досліджено переваги використаних технологій, зокрема масштабованість та самоконфігурованість застосунків на Spring Boot, надійність розробки клієнтів на Angular, переваги аутентифікації та авторизації через OAuth, ключові особливості RESTful API. Приділено увагу зручності інтерфейсу та безпеці.

## 3 Практичне використання

### 3.1 Запуск

Спочатку ми маємо пройти авторизацію.

Для цього потрібно натиснути View my schedules, після чого буде зроблено пере направилення на сторінку Microsoft (рисунки 3.1.1-2).



Рисунки 3.1.1-2 – Автоизація

Після успішної аутентифікації ми потрапляємо на сторінку з нашими розкладами (рисунок 3.1.1.3). Ми можемо відкрити будь-який наш розклад, просто натиснувши його:

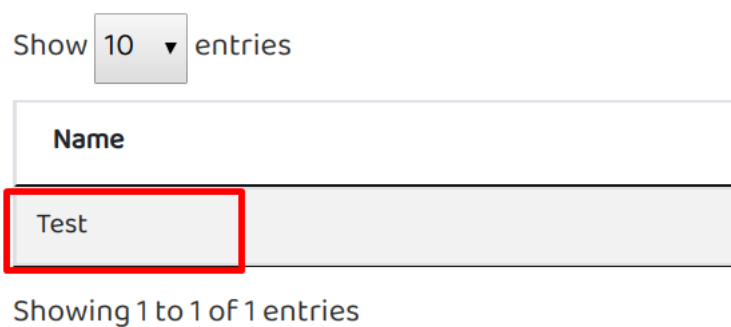


Рисунок 3.1.1.3 – Список розкладів

Для того, щоб вручну змінити розклад (рисунок 3.1.1.4), достатньо внести зміни в таблицю і натиснути кнопку збереження змін:

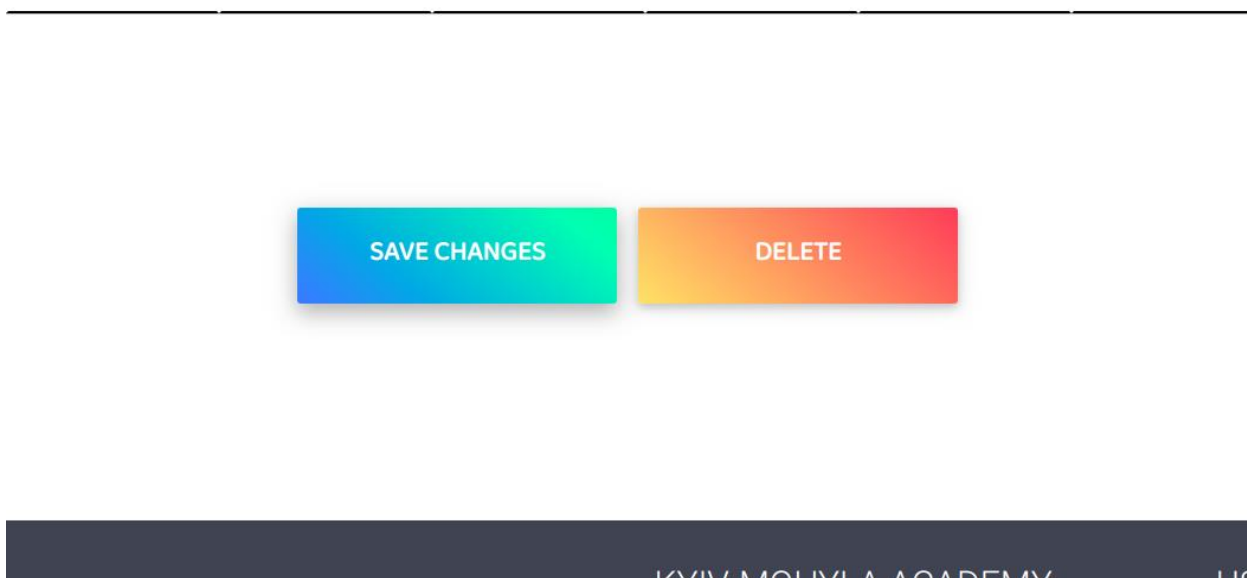


Рисунок 3.1.1.4 – Управління розкладом

Для видалення розкладу, відповідно, кнопку видалення.

Для того, щоб запросити у сервера генерацію нового розкладу, необхідно на сторінці зі своїми розкладами натиснути кнопку (рисунок 3.1.1.5) генерації нового розкладу.

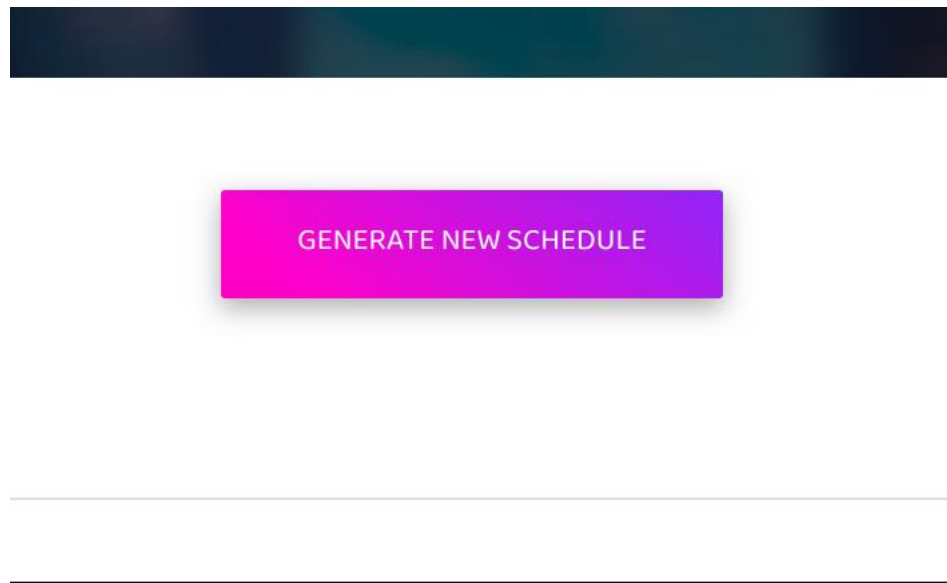


Рисунок 3.1.1.5 – Кнопка генерації

Система перенаправить користувача на сторінку з полями для введення інформації. Необхідно заповнити поля, не залишаючи порожньої інформації.

### **3.2 Висновки до частини 3**

У цій частині автором було зроблено підсумок до всієї роботи та продемонстровано роботу застосунку, зробленого із використанням досліджених технологій. Система працює та дозволяє автоматично створювати розклад на основі наданих вхідних даних.

## **4 Висновки**

Результатом роботи став прототип сервісу автоматичної генерації розкладу, веб-клієнт для даного сервісу, та API. У процесі дослідження алгоритмів було виявлено тонкощі та недоліки наявних рішень. Зокрема, виявлено неможливість створити абсолютно оптимальний алгоритм, а також складність уніфікувати сервіс під потреби будь-якого навчального закладу.

Було розглянуто та досліджено платформи для реалізації поставленої задачі як зі сторони фронтенду, так і зі сторони бекенду. Досліджено переваги даних платформ та тонкощі. Реалізовано авторизацію через третю сторону, упроваджено мінімальне розмежування прав.

Надалі планується вдосконалення алгоритму та розширення можливостей більш точного і більш повного налаштування параметрів для запиту генерації.

## Література та посилання

[1] Використання багатоагентних алгоритмів для вирішення задач складання розкладів, [електронний ресурс] режим доступу:

<https://cyberleninka.ru/article/n/ispolzovanie-mnogoagentnyh-algoritmov-dlya-resheniya-zadach-sostavleniya-raspisaniy/viewer>, вільний

[2] Алгоритмический подход к автоматическому и полуавтоматическому составлению расписания, [електронний ресурс], режим доступу:

<http://elibrary.asu.ru/xmlui/bitstream/handle/asu/3678/vkr.pdf?sequence=1&isAllowed=y>, вільний

[3] Задача складання шкільного розкладу, , [електронний ресурс], режим доступу: <http://www.mnogosmenka.ru/pilikov/school.htm>, вільний