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

# Розробка системи складання розкладу університету

**Текстова частина до курсової роботи**

Галузь знань: 12 «Інформаційні технології»

Спеціальність: 122 «Комп’ютерні науки та інформаційні технології »

Керівник курсової роботи
Старший викладач, кандидат наук
Шабінська М.О
*(підпис)*
“____” _____ 2020 р.

Виконав студент КНІТ-4
Лайко А.В
“____” _____ 2020 р.

Київ 2020

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

ЗАТВЕРДЖУЮ
Завідуючий кафедри інформатики, канд. фіз-мат. наук, доц.
_____ Гороховський С. С.
(підпис)
„____”_____2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Лайко Артему Володимировичу факультету інформатики 4 курсу
ТЕМА Розробка системи складання розкладу університету

Вихідні дані:
Зміст ТЧ до курсової роботи:
Індивідуальне завдання
Календарний план
Анотація
Вступ
РОЗДІЛ 1: Вступ до проблеми створення розкладу
РОЗДІЛ 2: Алгоритми створення розкладу
РОЗДІЛ 3: Опис технологій використаних для створення застосунку
РОЗДІЛ 4: Практична частина
Висновки
Список використаної літератури

Дата видачі „___” _____ 2020 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

**Тема:** Розробка системи складання розкладу університету

**Календарний план виконання роботи:**

| № п/п | Назва етапу дипломного проекту (роботи) | Термін виконання етапу | Примітка |
|---|---|---|---|
| 1. | Отримання теми курсової роботи. | 16.10.2019 | |
| 2. | Пошук тематичної літератури | 23.10.2019 | |
| 3. | Ознайомлення з тематичними матеріалами та побудова структури практичної та теоретичної частин курсової роботи | 27.10.2019 | |
| 4. | Написання вступу та змісту роботи | 11.12.2019 | |
| 5. | Ознайомлення з концепцією проблеми створення розкладу університету | 17.01.2020 | |
| 6. | Ознайомлення з існуючими принципами та підходами вирішення проблеми, пошук алгоритмів, що задовольняють вимоги | 8.02.2020 | |
| 7. | Створення застосунку для створення розкладу університету | 25.02.2020 | |
| 7. | Тестування роботоспроможності створеного застосунку | 29.02.2020 | |
| 8. | Написання другого та третього розділів базуючись на отриманих знаннях | 17.03.2020 | |
| 9. | Форматування курсової роботи відповідно до визначених вимог | 4.04.2020 | |
| 10. | Створення презентації та написання доповіді для захисту курсової роботи. | 7.04.2020 | |
| 11. | Узгодження попередньої версії роботи з керівником | 8.04.2020 | |
| 12. | Внесення змін до курсової роботи відповідно до зауважень наукового керівника | 14.04.2020 | |
| 13. | Захист курсової роботи | Квітень 2020 | |

Студент Лайко А.В.

Керівник Шабінська М.О.

“_____”_____

# Анотація

У даній курсовій роботі було визначено проблему створення розкладу університету, розглянуто принципи та алгоритми для її вирішення. Проведено їх порівняння задля визначення переваг та недоліків використання цих алгоритмів. Розроблено застосунок для створення розкладу університету. Докладено опис технологій що були використання у практичній частині. Дана робота використовує такі рішення як Spring Boot та Vue.js.

# Зміст

# Вступ

Кожен навчальний заклад прагне оптимізувати використання наявних ресурсів, що використовуються в освітньому процесі. Одним з основних способів є раціональне використання часу викладацького складу та студентів.

Таким чином вирішення проблеми щодо створення оптимального розкладу занять є найбільш пріоритетним. Оскільки правильно розроблений розклад допомагає у розкритті потенціалу викладачів та студентів, що у свою чергу призводить до покращення рівня освіти загалом. На додачу, правильний розклад рівномірно розподіляє навантаження на студентів та викладачів. Через це від розкладу залежить якість засвоєння навчальних матеріалів, використання часових ресурсів викладачів та матеріальної бази університету.

Проблема створення навчального розкладу відноситься до класу задач NP. Для розв’язку даної проблеми необхідно враховувати велику кількість параметрів та обмежень. Незважаючи на велику кількість досліджень спрямованих на вирішення даної проблеми, все ще не існує досконального рішення.

На даний момент, запропоновані рішення цієї проблеми задовольняють вимоги до них лише частково, через що, після автоматичної обробки даних розкладу ще необхідна ручна перевірка та «підгонка» результату.

Дана курсова робота має на меті дослідження алгоритмів, їх порівняння та розробку прототипу для створення розкладу занять університету.

# 1. Введення до проблеми створення розкладу

## 1.1 Розуміння проблеми створення розкладу

Розклад занять університету є одним з найважливіших компонентів організації освітнього процесу, оскільки він є запорукою передачі знань. Будь-який розклад занять у свою чергу залежить від таких чинників як: аудиторія, викладач, день тижня, час та певна група студентів. Також розклад не повинен мати колізій у часі, аудиторіях, студентських групах та викладачах, задовольняючи визначені обмеження. Обмеження щодо розкладу вносяться спираючись на побажання викладачів та студентів.

Таким чином створення якісного розкладу утворює доволі важку задачу, в котрій необхідно максимально задовільнити потреби усіх сторін в такий спосіб, щоб не порушити заздалегідь визначені обмеження щодо занять. Через це в багатьох навчальних закладах роботу із складання розкладу покладено на плечі співробітників та методистів, котрі вручну комбінують усі можливі варіанти, що в свою чергу призводить до неоптимального використання людських ресурсів.

## 1.2 Класифікація проблеми складання розкладу

Проблема створення розкладу університету відноситься до класу NP-повних задач та є гібридною проблемою оптимізації. Вона повинна задовольняти наданий перелік залежностей[1.3.1] та відповідати максимально можливій якості рішення[1.4].

Оскільки дана проблема відноситься до класу NP, вона не може бути розв’язаною за поліноміальний час через експоненційне зростання необхідного часу по відношенню до кількості заданих параметрів, через що одним з оптимальних підходів є евристичний.

## 1.3 Формалізація проблеми створення розкладу

В загальному вигляді проблема створення розкладу занять викладається як розподіл певної кількості подій скінченної множини у часі з урахуванням обмежень заданих користувачем, місця події та необхідних ресурсів.

Обмеження можуть бути жорсткими та м’якими. Жорсткі обмеження мають бути виконані повністю, та не мати жодних конфліктів, бо інакше можливі різноманітні аномалії розкладу. М’які обмеження в свою чергу є бажаними, але не обов’язковими до виконання.

Прикладом жорстких обмежень можуть бути кількість днів для навчання у тижні та/або кількість занять у день. Таким чином необхідно розробити такий розклад для кожної студентської групи на кожен навчальний день тижня, щоб навантаження на студентів та викладачів було рівномірно розподілено. В свою чергу м’які обмеження можуть включати в себе побажання викладачів та/або студентів щодо аудиторій та/або часових проміжків. Наприклад викладач бажає проводити заняття лише по вівторках у певній аудиторії, або в будь-якій аудиторії окрім заданої, студенти не хочуть мати «вікон» по середах, тощо.

### 1.3.1 Визначення обмежень розкладу

Метою створення оптимального розкладу є повне задовільнення жорстких обмежень та максимально можлива кількість м’яких обмежень.

В рамках даної курсової роботи був визначений наступний перелік вимог для створення розкладу.

**Жорсткі обмеження**

- Викладач не може викладати два заняття водночас

- Предмет не може викладатись в двох різних аудиторіях в один й той самий час(виключення становлять практики та/або семінари котрі викладаються різними викладачами в різних аудиторіях)
- Викладач проводить одне заняття з одного предмету в одній аудиторії в певний момент часу.
- В одній аудиторії не може відбуватись більше ніж одне заняття в один момент часу.
- Аудиторія повинна вміщати в себе повний обсяг студентів що відвідують заняття.

**М’які обмеження**

- Викладачі можуть висувати пропозиції щодо улюблених днів, часових проміжків та аудиторій для проведення занять.
- Заняття повинні бути розподілені таким чином, щоб мінімізувати кількість «вікон» для викладачів та студентів.
- Заняття повинні починатись якомога раніше, аби мінімізувати кількість пізніх пар.

Правильна комбінація цих вимог та залежностей утворює оптимальний графік занять, що буде задовольняти освітні вимоги викладачів, студентів та університету загалом.

### 1.4 Визначення критерію якісного розкладу

Для того, аби мати уявлення щодо якості створеного розкладу необхідно ввести формальний критерій за котрим можна оцінити його. Таким критерієм може виступати кількість виконаних м’яких вимог щодо створеного розкладу, чим більше це число, тим більш якісним є розклад.

$$Q = z/a$$

Де Q – коефіцієнт якості, z – кількість задоволених м'яких вимог, $a$ – загальна кількість заздалегідь визначених м'яких вимог до розкладу. Отже, маючи Q = 1, ми можемо стверджувати, що отримали ідеальний розклад для заданого набору параметрів.

Для отримання загальної формули якості розкладу ми можемо припустити, що часткове задоволення поставлених вимог є більш ймовірним варіантом.

$$Q(z) = \sum_{i=1}^{a} \frac{Q_i(z)}{Q_i}$$

В цьому випадку ми маємо більш точну оцінку ефективності запропонованого алгоритму, через врахування кожного випадку часткового виконання поставлених вимог.

### 1.5 Математичне формулювання проблеми створення розкладу

Математичне визначення проблеми створення розкладу включає в себе:

$n$ – кількість занять $E = \{e_1, e_2, \dots, e_n\}$

k – число доступних часових інтервалів $T = \{t_1, t_2, \dots, t_k\}$

l – кількість наявних аудиторій $R = \{r_1, r_2, \dots, r_l\}$

m – кількість груп студентів $S = \{s_1, s_2, \dots, s_m\}$

SC – множина м'яких обмежень

HC – множина жорстких обмежень

Маючи задані змінні, можна отримати матриці якості для відношень між заняттями(студенти-заняття, аудиторія-заняття, заняття-часовий інтервал). Таким чином матриця відношення студенти-заняття $A$ має розмір $k \times n$ та кожне значення $A_{ij}$ має значення $Q(A_{ij})$.

### 1.6 Визначення існуючих підходів у вирішені проблеми створення розкладу

Перше запропоноване визначення проблеми створення розкладу університету включало в себе три множини: викладачі, аудиторії та часові інтервали. Для вирішення даної проблеми були запропоновані наступні підходи: Graph coloring, Constraint Satisfaction, Genetic algorithms, Tabu search algorithm, Simulated Annealing та підходи вирішення проблеми що базуються на дослідженнях штучного інтелекту. Дана курсова робота має на меті розглянути наведені алгоритми та порівняти їх ефективність.

## 2. АЛГОРИТМИ СТВОРЕННЯ РОЗКЛАДУ

На сьогоднішній день існує багато способів вирішення проблеми створення розкладу університету, за допомогою різних підходів.

### 2.1 Алгоритм розмалювання графу для вирішення проблеми створення розкладу

Перше визначення проблеми створення розкладу з’явилось у 1963 році та складалось з трьох множин(викладачі, аудиторії та часові проміжки). Одним з перших способів вирішення даної проблеми став алгоритм побудований на принципі розмалювання графу.

Даний алгоритм має на меті розмалювання графу таким чином, щоб сусідні вершини не мали однакового кольору. Результуючий граф не повинен мати конфлікти у розкладі та використовувати мінімальну кількість кольорів. В даному алгоритмі вершини являють собою заняття, сторони є обмеженнями, а кольори – часовими інтервалами, відповідно.

Вдосконалені варіації даного рішення включають в себе відокремлення наявних вершин графу для зменшення кількості можливих кольорів, таким чином комбінуючи незалежні вершини з меншою кількістю кольорів, можна спромогтися покращити результати запропонованого алгоритму.

Але даний алгоритм, незважаючи на те, що надає просте рішення для проблеми побудови розкладу, створюючи ефективний розклад, має низку недоліків, одним з котрих є його належність до NP-класу, його час виконання з великою кількістю залежностей може не давати належної ефективності, також він не має змоги розв'язати дану задачу за наявності попередньо визначених часових інтервалів(кольорів) для занять, через що його використання для створення повноцінного, складного розкладу є проблематичним.

### 2.2 Алгоритм забезпечення виконання визначених обмежень

Метод забезпечення виконання визначених обмежень може бути визначений як спосіб обмеження простору об'єктів. Він має на меті можливість знаходження наборів таких значень, які могли б бути визначені як значення змінних та задовільнити заздалегідь визначені обмеження.

Це є рішення проблеми з трьома змінними, що виглядає як

$$CS = (X, D, C)$$

Де X – являє собою кінцевий набір змінних $x_1, x_2, \dots, x_n$,

D – скінчений набір значень домену $d_1, d_2, \dots, d_n$ ,

Таким чином, щоб кожна змінна з цього домену може бути обрана та C – як скінченний набір обмежень $c_1, c_2, \dots, c_n$. В даному розв'язку обмеження на додачу залежать від підмножини змінних X. Кінцевий результат даного алгоритму являє собою отримання значення D змінних X в тих випадках, коли ці значення можуть задовільнити надані обмеження C.

### 2.3 Генетичні алгоритми для вирішення проблеми створення розкладу

Генетичні алгоритми є підкласом евристичних методів, та базуються на моделі схожою на природній еволюційний механізм. Даний метод використовує наступні кроки для оптимізації отриманого результату:

1. Відбір
2. Схрещення
3. Заміщення

За допомогою генетичного алгоритму відбувається сортування розкладів за заздалегідь визначеним параметром якості, та їх схрещення для отримання оптимального результату, порушуючи мінімальну кількість м'яких обмежень. На кожній ітерації відбувається відбір найкращих за якістю розв'язку, після чого вони комбінуються між собою утворюючи вдосконалені версії, і за скінчену кількість ітерацій можна отримати задану якість розкладу. Таким чином генетичні алгоритми надають більш якісні результати порівняно з іншими запропонованими розв'язками.

### 2.4 Алгоритм табу-пошуку

Табу-пошук так само входить до множини евристичних алгоритмів оптимізації. Він базується на пошуку найкращого сусіда відносно поточного результату, з урахуванням списку заборон. Якщо результат сусіда не знаходиться в списку заборон, тоді алгоритм буде рухатись до нього, в іншому випадку буде перевірений критерій прагнення, аби не потрапити у пастку локальних оптимумів, щоб запобігти поверненню до попередніх рішень. Якщо сусідський результат є кращим, аніж наявний, базуючись на критерії прагнення, буде виконано рух до даного сусіда, а попередній результат занесено до списку заборон, аби уникнути повернення назад.

Пошук оптимальних результатів триває до моменту досягнення умови припинення.

Скорочений принцип роботи виглядає як:

1. Обираємо рішення $x$ котре буде слугувати початковим, та припускаємо що $F_{PZ} \coloneqq F(x)$, формуємо пустий список заборон.

2. Починаємо пошук нового розв’язку $z$ такого, що будуть задовольнятись наступні умови

$$z \in N'(x)$$

$$F(z) = \min\{F(y) | y \in N'(x)\}$$

перехід $x \to z$ не є забороненим або $F(z) < F_{PZ}$

3. Рухаємося до нового вузла $x := z$ та розширюємо список заборон, якщо $F(x) < F_{PZ}$, тоді замінюємо запис $F_{PZ} := F(x)$

4. Якщо після останнього переходу був задовільнена умова зупинки, то повертаємо результат, в іншому випадку повертаємося до другого кроку та продовжуємо пошук локальних оптимумів.

Наведений алгоритм використовується для побудови розкладу у декілька етапів. Під час першого етапу створюється список набір розв’язків для певної групи студентів після чого відбувається друга фаза із залученням табу-пошуку, щоб отримати можливі найкращі комбінації часових інтервалів для кожної групи не беручи до уваги найгірші комбінації. В третьому, завершальному, етапі відбувається заключний пошук найбільш якісних розв’язків серед отриманих результатів з другого етапу, та застосовується на множину усіх студентських груп.

### 2.5 Алгоритм імітованого відпалювання

Алгоритм імітованого відпалу є однією з варіацій алгоритмів локального пошуку, котрий базується на спостереженнях за фізикою кристалізації речовини під час відпалювання металів. Коли наявний метал нагрівається до температури, що перевищує його температуру плавлення, атоми починають хаотичний рух. А коли метал починають поступово охолоджувати, то в деяких місцях починають утворюватись стани з

низькою кількістю енергії, те місце в котрому буде досягнуте найменш енергійний стан, буде вважатись точкою глобального мінімуму(оптиму).

Використання даного підходу дозволяє позбутись проблем потрапляння у пастку локального оптимуму, що є притаманним для алгоритму табу-пошуку, та робить реалізацію пошуку найбільш якісного розв'язку значно легшою.

Даний підхід спирається на наступні кроки. На початку роботи створюється випадковий розв'язок котрий отримує певну оцінку якості, та стає точкою відліку якості подальших розв'язків. Таким чином алгоритм імітованого відпалу на другому кроці пропонує інший випадковий варіант рішення припускаючи його перевагу у якості в порівняні з попереднім варіантом, якщо це твердження справджується, то запропонований варіант стає поточним і алгоритм повторює свою працю допоки не натрапить на умову зупинки.

Пошук оптимального рішення починається з високого значення «температури», котра з часом падає за формулою регресії $T_{i+1} = T_i \times \gamma$, де $\gamma$ виступає у якості «швидкості охолодження». Проте початкові параметри температури $T$ та «швидкості охолодження» $\gamma$ можуть задаватись як випадково, так і відносно кількості наданих параметрів, таких як:

- кількість м'яких обмежень
- кількість жорстких обмежень
- кількість предметів що будуть викладатись
- кількість груп студентів

Таким чином ми можемо сформулювати математичне відображення розв'язку задачі створення розкладу за допомогою алгоритму імітованого відпалювання.

1. Створюємо випадкове рішення розкладу $S_0$, $S = S_0$ .

2. Встановлюємо високий показник початкової температури $T_0$ та показник ймовірності зміни розкладу, котрий впливає на кількість можливих комбінацій із зміною часових інтервалів, аудиторій та/або порядку занесених до розкладу занять.

3. Маючи поточний розклад $S_0$ та показник ймовірності зміни розкладу генерується новий варіант розкладу $S'$, котрий відрізняється від початкового.

4. Отримавши нову варіацію розкладу вираховується дельта якості $\Delta Q = Q(S') - Q(S)$, і спираючись на зміну якості у кращу сторону(за умови $\Delta Q > 0$) чи у гіршу сторону(за умови $\Delta Q < 0$).

5. Після отримання дельти якості вираховується функція щодо зміни поточної «температури» та вірогідність використання отриманого рішення розкладу в якості поточного $\beta = e^{-\Delta Q/T}$, але при поганому показнику дельти попереднє значення розв'язку може бути використано з вірогідністю $\rho = (1 - \beta)$. Значенням до зміни температури може бути кількість відпрацьованих циклів до цього, так само цей показник може слугувати в якості критерію зупинки алгоритму у випадку якщо цільовий показник якості не було можливим отримати.

6. Якщо не був задовільнений критерій зупинки алгоритму, то процес повторюється з кроку №3, або зупиняється, якщо був перевищений допустимий показник кількості ітерацій, або показник виконання декількох ітерацій поспіль без належного позитивного приросту якості розв'язку.

Також для покращення отриманих результатів роботи даного алгоритму його можна комбінувати з варіаціями генетичних алгоритмів, та додатково зберігати декілька найбільш результативних розв'язків до пам'яті та оптимізувати їх після отримання задовільного критерію зупинки

алгоритму імітації відпалювання. На якість результату ще впливає швидкість зміни «температури», за низького фактору зміни температури даний підхід може надати більш точний та якісний розв’язок, хоча це й може призвести до збільшення часових витрат. Через це фактор зміни температури необхідно визначати для кожної ситуації окремо, та мати деяку інформацію щодо попередніх результатів та спостережень над ними.

# 3. Опис технологій використаних для реалізації застосунку

Практична частина даної курсової роботи має на меті ознайомлення з новими технологіями для розробки ефективних застосунків та імплементацію базового користувацького інтерфейсу для створення розкладу університету, його перегляду, редагування та поширення між викладачами та студентами.

## 3.1 Вимоги до інструментів та технологій

Для розробки застосунку було визначено наступний перелік обов’язкових вимог до технологій, що будуть застосовані:

- Можливість реалізації REST API взаємодії між клієнтом та сервером
- Наявність ORM(об’єктно реляційної моделі даних)
- Співпраця з СУБД MySQL та/або PostgreSQL
- Можливість створення реактивного клієнтського застосунку
    - Можливість імплементації модульного підходу
    - Наявність контролю поточного стану застосунку
- Легкість подальшої підтримки існуючого застосунку, його розширення та/або відлагодження

- Наявність великої спільноти та багатої документації, аби пришвидшити розробку та підвищити її якість

Сформувавши перелік вимог до технологій, був здійснений пошук та обрані кандидати, котрі задовольняють частину потреб у повному або наближеному до повного обсязі.

### 3.1.1 Перелік інструментів-кандидатів

Маючи уявлення про вимоги до інструментів, було сформовано наступний список кандидатів, що задовольняють потреби у повному або найбільшому обсязі.

**Технології для створення REST API**

- Spring Boot, що базується на мові Java
- Express, що базується на node.js
- Laravel, що базується на php

**Технології для створення клієнтської частини**

- React.js
- Vue.js
- Angular.js

## 3.2 Обрані технології

Зважаючи на усі переваги та недоліки наведених технологій, було обрано до використання Spring Boot як фреймворк побудови REST API серверу, та Vue.js як бібліотеку для створення клієнтської частини застосунку. Комбінація наведених інструментів дає змогу в короткий час створити повноцінне застосування, котре може бути за потреби легко розширене та адаптовано під вимоги, що можуть змінюватись з часом.

### 3.3 Опис Spring Boot

Spring Boot є відносно молодим фреймворком з відкритим кодом від компанії Pivotal, розроблений з метою пришвидшення та спрощення створення застосунків. Він базується на Spring Framework та звільнює розробника від необхідності задавати надмірну конфігурацію платформи та дозволяє у короткий час підняти готовий застосунок на Tomcat або Jetty сервері.

Даний фреймворк покликаний вирішити питання створення надійних back-end рішень для багатьох галузей та позбутися «пилу» надмірності свого «великого» попередника у вигляді Spring Framework. На додачу, Spring Boot має ідею того, щоб не розв’язувати старі проблеми за допомогою нових інструментів, а надати функціонал, котрий покращить процес розробки ПО. Через зрозумілу документацію та велику кількість супроводжуючих матеріалів та прикладів, можна дуже легко почати працювати із Spring Boot.

У порівнянні з Express та Laravel, Spring Boot виграє за декількома параметрами, а саме: він використовує мову програмування Java, котра має жорстку типізацію у порівнянні з javascript та php, та більшу документацію створену завдячуючи спільноті, а можливість виконання готового продукту на віртуальній машині Java робить можливим його розгортання на будь-якому хост-сервері.

### 3.4 Опис Vue.js

Vue.js є однією з трьох провідних бібліотек для розробки реактивних клієнтських застосунків (React, Vue.js, Angular). Дана бібліотека має модульну архітектуру, та може бути розширена за допомогою встановлення нових модулів, котрі можуть бути розроблені як спільнотою так і самим розробником.

Переваги Vue.js порівняно з React або Angular можна побачити вже безпосередньо працюючи з цією бібліотекою. На відміну від двох інших рішень вона від початку «рекомендує» підходи щодо того, як необхідно створювати застосунок, на відміну від React'у котрий надає потужний інструментарій, проте не розповідає що з ним робити, тож у ситуаціях коли остаточного бачення проблеми, або її рішення не має на даний момент, та задля економії часу доцільнішим є використання саме Vue.js. Якщо ж порівнювати із Angular'ом, то Vue не ставить на меті створення великої кількості надмірних абстракцій, а навпаки прагне їх мінімізувати, щоб розробник мав чітке бачення процесів, що відбуваються «під капотом», а з боку клієнтського досвіду не було проблем з продуктивністю, котрі є притаманними для застосунків, що використовують Angular.js.

На додачу, Vue.js надає широкий спектр можливостей одразу «із коробки», через що не потребує великої кількості сторонніх модулів, для розширення можливостей даної бібліотеки. Саме через перелік цих переваг дана бібліотека є більш привабливою для використання.

# 4. Практична частина

## 4.1 Опис практичної задачі

Практичним завданням даної курсової роботи було створення програмного продукту для методистів факультетів Києво-Могилянської академії, котрий допоміг би їм у створені навчального розкладу та зробив би можливим його переведення у цифровий вигляд.

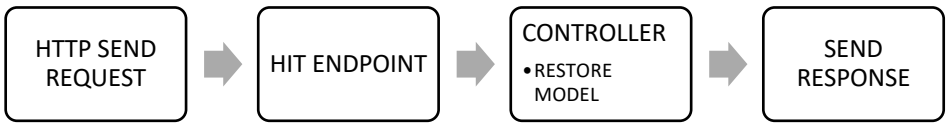
У загальному вигляді це повинен бути веб-сайт або веб-застосунок, що надає змогу передивлятись розклад, редагувати та додавати нові заняття до загального розкладу певної спеціальності.

Тож було обрано створення MVC(Model-View-Controller) веб-застосунку з відокремленням відображення(View) від моделі та контролеру(Model & Controller).

### 4.2 Структура back-end частини застосунку

У даному застосунку back-end частина використовує підхід REST API та є обробником усіх запитів від клієнтської частини. Таким чином з'являється можливість відокремити серверний та клієнтський застосунки, і виразити їх спілкування у вигляді HTTP запитів, в залежності від котрих буде змінюватись поведінка інтерфейсу, що надається користувачеві.

В даному випадку ми маємо уявлення, що запит користувача проходить наступний шлях:




Спочатку клієнт ініціює HTTP запит до сервера, після того він відправляється та потрапляє на певну кінцеву точку, за обробку котрої відповідає контролер. Кожна кінцева точка слугує для виконання конкретної дії на даними, як то отримання даних, додавання нових, оновлення та видалення. Під час обробки запиту контролер може відновити модель об'єкту, якщо був надісланий його ідентифікатор з метою оновлення чи видалення. Обробивши запит контролер надсилає клієнту

відповідь із HTTP кодом, котрий може свідчити про статус цього запиту, чи був він виконаний без помилок, чи все ж таки помилки трапилися. Не зважаючи на статус запиту, додатково у відповіді будуть міститись або дані, котрі запитував користувач(для оновлення, додавання, видалення), або повідомлення про помилку з деталями.

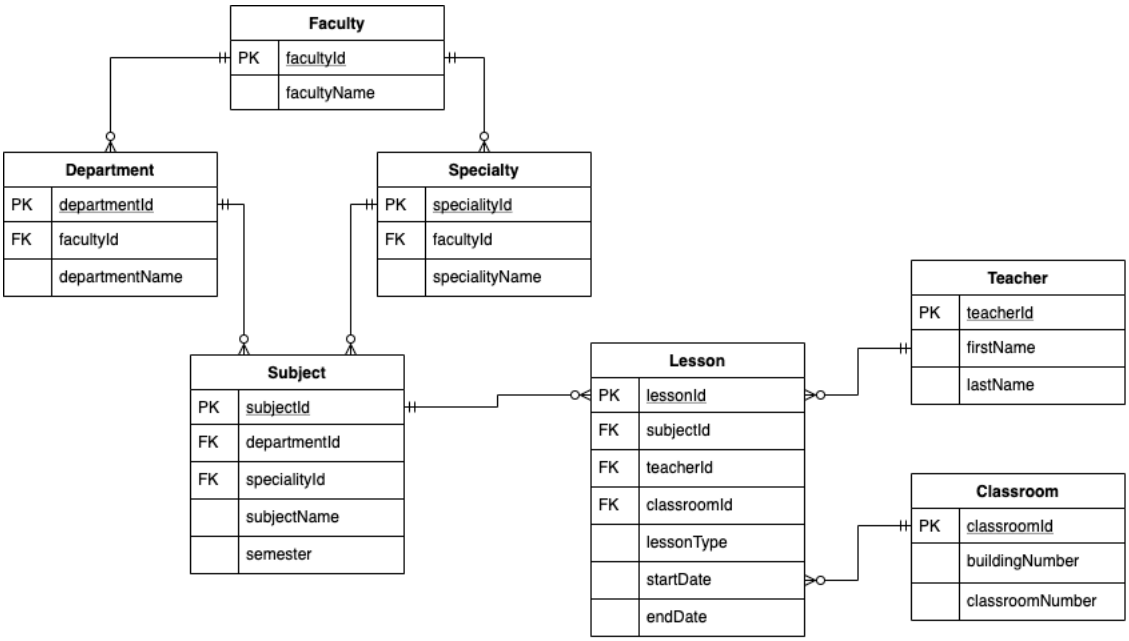
### 4.2.1 Структура бази даних

Як і будь-яка інша інформаційна система, даний застосунок повинен мати постійне сховище даних. В якості системи керування баз даних була обрана MySQL.

Також завдяки підтримці ORM(object-relation model) всередині Spring Boot, була можливість задати моделі для кожного суб'єкту програмним способом. Усі моделі зберігаються всередині пакету ua.edu.ukma.schedule.model.

Для організації лімітованого доступу до полів моделей були створені DTO(data transfer object). Вони слугують своєрідними конвертами, що пересилаються до клієнта, та мають лише певний перелік даних, котрі необхідні користувачеві, в той час як решта інформації, що належить моделі не надсилається. DTO зберігаються поруч з моделями у пакеті ua.edu.ukma.schedule.model.dto.

Для моделі даних була створена наступна ER-модель:




Виокремлено наступні сутності:

- Факультет (Faculty)
- Кафедра (Department)
- Спеціальність (Specialty)
- Предмет (Subject)
- Викладач (Teacher)
- Аудиторія (Classroom)
- Та сутність «Заняття» (Lesson), котра поєднує відношення викладача, аудиторії та предмету.

Відповідно до кожної сутності з ER-схеми було створено моделі та DTO-об'єкти всередині застосунку.

### 4.2.2 Доступ до даних

Можливість отримання та взаємодії з даними, що зберігаються у БД є важливою складовою будь-якого застосування, при неправильній організації передачі параметрів можливе утворення небезпечних багів у застосунку.

Для доступу до даних використовується декілька абстракцій. Першою такою абстракцією, котру надає Spring Boot є Repository (в нашому випадку одна з його реалізацій CrudRepository). Цей підхід дозволяє мати заздалегідь визначений перелік операцій над певною сутністю бази даних, таким чином забороняючи виконання будь-яких несанкціонованих операцій з БД.

Другою абстракцією у переліку є Service, це такі класи, котрі дозволяють реалізовувати бізнес логіку застосунку, не порушуючи цільове призначення репозиторію. Таким чином сервіс може дістати з репозиторію необхідні данні, привести до певного вигляду та повернути назад до контролеру.

Третьою й найвищою абстракцією є контролер, котрий отримує запит користувача щодо даних, та запускає виконання сервісів щодо отримання даних зі сховища, їх внесення та/або модифікацію.

Завдяки наявності цих шарів абстракції, будь-який несанкціонований доступ або зміна даних застосунку з метою пошкодження, або через недбалість виключається.

### 4.3 Структура front-end частини застосунку

Клієнтський інтерфейс є основним способом для користувача, щоб отримувати, вносити та модифікувати інформацію у системі. Через це до клієнтського застосунку повинні бути висунуті такі ж вимоги щодо якості та надійності як і до back-end’у.

Зважаючи на необхідність отримання потужного інтерфейсу користувача, було обрано створення SPA (single page application) з використанням Vue.js.

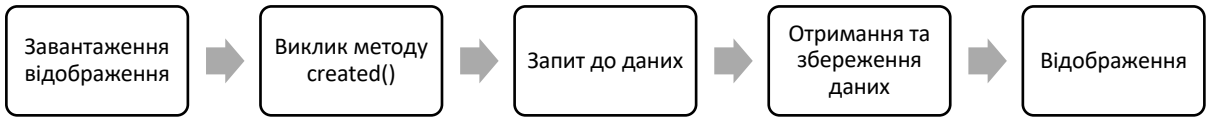
### 4.3.1 Відображення даних

Інтерфейс користувача напряму залежить від переліку даних, що зберігаються у застосуванні. Це чинник також впливає на структуру відображень та способи отримання інформації.

Зважаючи на попередні фактори, було створено модульну систему. Вона дозволяє створювати віртуальні шляхи всередині застосунку (так звані routes), котрі в свою чергу динамічно завантажують компоненти із структурою відображення. Кожен компонент має модель даних, що співпадає з моделлю котру надає back-end частина у вигляді DTO.

Після активації компоненту (відвідування певного шляху) відбувається виконання сценаріїв сторінки. Першим таким сценарієм є метод created(), котрий викликається після того, як DOM модель сторінки вже була створена. Всередині цього методу відбувається HTTP запит до back-end'у з метою отримання даних. Усі дані поточного відображення зберігаються усередині контейнеру стану, котрий дозволяє контролювати, які саме компоненти, в якій послідовності, та яким чином мають з'являтись перед користувачем.

Ілюстрація процесу створення компоненту:



### 4.3.2 Збереження поточного стану користувача

Кожна дія користувача на веб-сторінці застосунку супроводжується перезавантаженням певної кількості інформації, залежно від компонентів, що змінюються. Оскільки кожен компонент надсилає запит щодо даних у момент свого створення, було визначено необхідність збереження стану користувача разом із отриманими даними, аби оптимізувати роботу застосунку та не надсилати надмірну кількість запитів.

Таким чином було створено глобальний компонент тимчасового сховища даних для користувача store. Даний компонент дозволяє зберігати будь-які формати даних, та мати змогу динамічно оновлювати їх поміж декількох незалежних компонентів. Завдяки цьому метод запиту до даних був модифікований, та додалася умова відсутності інформації. Якщо така умова задовольняється лише в такому випадку надсилається запит до серверу.

### 4.3.3 Компоненти користувача

Кожен з існуючих компонентів обслуговує певний віртуальний шлях(route). Далі наводиться відношення існуючих компонентів до моделей та даних, котрі вони оброблюють.

| **Назва компоненту** | **За що відповідає** |
|---|---|
| MainCalendar.vue | Основний компонент, котрий дозволяє переглядати розклад за певними параметрами, такими як факультет, спеціальність, семестр. Також він надає змогу створювати нові події(заняття) та заносити їх до бази даних. |

| | |
|---|---|
| ClassroomsPage.vue | Компонент для відображення, редагування інформації про аудиторії. |
| DepartmentsPage.vue | Компонент для відображення, редагування інформації про кафедри. |
| FacultiesPage.vue | Компонент для відображення, редагування факультетів. |
| SpecialitiesPage.vue | Компонент для відображення, редагування спеціальностей. |
| SubjectsPage.vue | Компонент для відображення, редагування предметів, що викладаються. |
| TeachersPage.vue | Компонент для відображення, редагування викладачів та їх інформації. |
| NewItemModal.vue | Компонент модального вікна для відображення інтерфейсу створення або редагування інформації про певну сутність. Присутній на кожній з попередніх сторінок, та змінює свою модель даних відповідно до них. |

# Висновки

Працюючи над виконанням даної курсової роботи я отримав змогу детальніше розібратись в проблемі створення університетського розкладу, зрозуміти необхідність вирішення даної задачі, ознайомився з прикладами та комбінаціями алгоритмів, що використовуються на сьогоднішній день для вирішення проблеми. Практичне завдання допомогло мені ознайомитись та використати для вирішення справжніх проблем такі бібліотеки та фреймворки як Spring Boot, Vue.js, покращило розуміння підходів до створення REST API серверів, та клієнтської взаємодії. Загалом дана курсова робота допомогла мені структурувати теоретичні знання та підкріпила їх практикою.

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ


1. Burke, E.K. A University Timetabling System Based on Graph Colouring and Constraint Manipulation.
2. J. Thompson, K. Dowsland. Variants of simulated annealing for the examination timetabling problem.
3. H. Babei, J. Karimpour, A. Hadidi. A survey of approaches for university course timetabling problem.
4. B. Naderi. Modeling and Scheduling University Course Timetabling Problems.
5. A. Nanda, Manisha P. Pai, Abhijeet Gole. An Algorithm to Automatically Generate Schedule for School Lectures Using a Heuristic Approach
6. Constraint satisfaction problems. http://aima.cs.berkeley.edu/newchap05.pdf [електронний ресурс]
7. Jean-Charles Regin, Sophia Antipolis. Modeling Problems in Constraint Programming.
8. Fred Glover. Tabu Search
9. Kirkpatrick S., Gelatt C. D, Vecchi M.P. Optimization by Simulated Annealing
10. https://spring.io/projects/spring-boot [електронний ресурс]
11. https://expressjs.com/ [електронний ресурс]
12. https://vuejs.org/v2/guide/ [електронний ресурс]
13. https://reactjs.org/docs/getting-started.html [електронний ресурс]
14. https://angular.io/docs [електронний ресурс]