

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ СТВОРЕННЯ ТА

(назва теми)

КЕРУВАННЯ РОЗКЛАДОМ. BACKEND ЧАСТИНА
DEVELOPMENT OF A WEB APPLICATION TO CREATE AND MANAGE A
SCHEDULE. BACKEND PART

Виконав(ла): студент(ка) 4 курсу, групи КНТ-219

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

ПРИМАКОВ В.В.

(ПРИЗВИЩЕ та ініціали)

Керівник СТЕПАНЕНКО О.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет ФКНТ

Кафедра програмних засобів

Ступінь вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки

(код і найменування)

Освітня програма (спеціалізація) Комп'ютерні науки

(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.

Сергій СУББОТІН

«_____» _____ 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ПРИМАКОВА Владислава Володимировича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка вебзастосунку для створення та керування розкладом. Backend частина

Development of a Web Application to Create and Manage a Schedule. Backend Part

керівник проєкту (роботи) к.т.н., доцент, СТЕПАНЕНКО Олександр Олексійович,

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від «4» квітня 2023 року № 87.

2. Строк подання студентом проєкту (роботи) 1 червня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної області. 2. Матеріали і методи. 3 Опис програми. 4. Тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	СТЕПАНЕНКО О.О., доцент		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання «4» квітня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

_____ Владислав ПРИМАКОВ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

_____ Олександр СТЕПАНЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 168 с., 37 рис., 19 табл., 3 дод., 25 джерел.

СКЛАДАННЯ РОЗКЛАДУ, РОЗКЛАД ЗАНЯТЬ, ГЕНЕТИЧНИЙ АЛГОРИТМ, ОСТРІВНА МОДЕЛЬ, НЕЙРОНА МЕРЕЖА, LSTM, EXPRESS, MYSQL, GRAPHQL, SEQUELIZE, C++, JAVASCRIPT.

Об'єкт дослідження – процес генерації розкладу занять для навчального закладу.

Предмет дослідження – програмні засоби для генерації та керування розкладу занять.

Метою роботи є дослідження та розробка програмного забезпечення для генерації за допомогою різноманітних алгоритмів та керування розкладом.

Матеріали, методи та технічні засоби: мова програмування JavaScript, C++, фреймворк Express, QT.

Результати. Розроблено серверну частину вебзастосунку для автоматизованої генерації та керування розкладу занять в навчальному закладі, яка дозволяє оброблювати HTTP запити для керування базою даних, а також алгоритмами, які генерують розклад.

Висновки. Створено програмний продукт для генерації та керування розкладом.

Галузь використання – може бути корисними для навчальних закладів, починаючи з дошкільних установ і закінчуючи вищими навчальними закладами. Програмний продукт допоможе забезпечити оптимальний розподіл розкладу уроків, уникнути перекриття занять та вікон між ними.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelors: 168 pages, 37 pictures, 19 tables, 3 appendixes, 25 sources.

SCHEDULING, CLASS SCHEDULE, GENETIC ALGORITHM, ISLAND MODEL, NEURON NETWORK, LSTM, EXPRESS, MYSQL, GRAPHQL, SEQUELIZE, C++, JAVASCRIPT.

The object of the study is the process of generating a schedule of classes for an educational institution.

The subject of the research is software tools for generating and managing the class schedule.

The aim of the work is research and development of software for generation using various algorithms and schedule management.

Materials, methods and technical means: JavaScript programming language, C++, Express framework, QT.

The results. The server part of the web application was developed for the automated generation and management of the schedule of classes in the educational institution. Which allows you to process HTTP requests to manage the database, as well as algorithms that generate the schedule.

Conclusions. Created a software product for schedule generation and management.

Field of use – can be useful for educational institutions, starting from preschool institutions and ending with higher educational institutions. The software product will help ensure the optimal distribution of the lesson schedule, avoid overlap of classes and windows between them.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	9
Вступ.....	11
1 Аналіз предметної області.....	13
1.1 Основні поняття	13
1.2 Огляд наявних методів	14
1.2.1 Алгоритм табу пошуку	19
1.2.2 Алгоритм імітації відпалу	20
1.2.3 Генетичний алгоритм та його модифікації	22
1.2.4 Нейронні мережі.....	25
1.3 Порівняльний аналіз наявних методів вирішення завдання.....	29
1.4 Огляд наявних систем.....	30
1.4.1 Огляд програми Untis	30
1.4.2 Огляд програми TimeTabler	32
1.4.3 Огляд програми TimeEdit.....	33
1.5 Порівняльна характеристики наявних систем	35
1.6 Формулювання завдання до роботи	36
1.7 Висновок	38
2 Матеріали і методи.....	39
2.1 Вибір мови програмування для серверної частини	39
2.1.1 JavaScript	39
2.1.1.1 Фреймворк ExpressJS.....	40
2.1.1.2 Фреймворк NestJS	41
2.1.2 Python	42
2.1.2.1 Фреймворк Django	45
2.1.2.2 Фреймворк Flask	46
2.1.3 C#	47
2.1.3.1 Фреймворк .NET Framework.....	48

2.1.3.2 Фреймворк .NET Core.....	48
2.2 Порівняння мов програмування та їх фреймворків. Вибір архітектури системи для розробки серверної частини вебзастосунку	49
2.3 Вибір мови програмування для розробки алгоритмів.....	56
2.3.1 C++.....	56
2.4 Порівняння мов програмування. Вибір архітектури системи для розробки алгоритмів	58
2.5 Вибір та порівняння архітектури та шаблонів проєктування вебзастосунку	60
2.6 Вибір бази даних	70
2.5.1 MySQL.....	70
2.5.2 PostgreSQL	71
2.5.3 MongoDB.....	72
2.5.3 Порівняння та аналіз баз даних	73
2.7 Розробка моделі взаємодії користувача з системою	76
2.8 Висновок	78
3 Опис програми.....	80
3.1 Структура бази даних MySQL	80
3.2 Структура вебсерверу ExpressJS	87
3.2.1 Підключення до БД та Models	87
3.2.2 Схема GraphQL.....	93
3.2.3 Схема вебсерверу Express	107
3.3 Опис алгоритмів	109
3.3.1 Алгоритм простого перебору.....	109
3.3.3 Алгоритм табу пошуку	110
3.3.4 Алгоритм імітації відпалу	111
3.3.5 Генетичний алгоритм	112
3.3.6 Модифікація генетичного алгоритму – Острівна модель.....	114
3.3.7 Нейронна мережа	116
3.4 Висновок	119

4 Тестування та експериментальне дослідження програми	120
4.1 Загальні дані для тестування алгоритмів.....	120
4.2 Результати тестів генетичного алгоритму	122
4.3 Результати тестів модифікації генетичного алгоритму – острівної моделі.....	125
4.4 Результати тестів алгоритму імітації відпалу	126
4.5 Результати тестів табу пошуку	127
4.6 Результати тестів нейронної мережі	128
4.7 Висновок	130
Висновки	132
Перелік джерел посилання	134
Додаток А Дані роботи алгоритмів	137
Додаток Б Текст програми	147
Додаток В Слайди презентації.....	158

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
CLI	– Command Line Programm;
CLR	– Common Language Runtime;
DNS	– Domain Name System;
HTTP	– HyperText Transfer Protocol;
IP	– Internet Protocol;
JIT	– Just-In-Time;
JS	– JavaScript;
LSTM	– Long Short-Term Memory;
MVC	– Model-View-Controller;
MVP	– Model-View-Presenter;
MVT	– Model-View-Template;
MVVM	– Model-View-ViewModel;
NP	– Non-Polynomial;
ORM	– Object-Ralational Mapping;
PWA	– Progressive Web Application;
REST	– Representational State Transfer;
RNN	– Recurrent Neural Network;
SOAP	– Simple Object Access Protocol;
SPA	– Single Page Application;
SQL	– Structed Query Language;
UML	– Unified Modeling Language;
URL	– Uniform Resource Locator;
WSGI	– Web Server Gateway Interface;
АГР	– автоматизована генерація розкладу;
БД	– база даних;
ГА	– генетичний алгоритм;

ІВ	— імітація відпалу;
НЗ	— навчальний заклад;
НМ	— нейронна мережа;
ОМ	— острівна модель;
ООП	— об'єктно-орієнтовне програмування;
ОС	— операційна система;
ТП	— табу пошук.

ВСТУП

Майже всі навчальні заклади мають проблеми з розкладом, особливо університети. Для того, щоб скласти розклад, потрібно врахувати багато речей. Одна з них є доступність викладачів. Не всі викладачі доступні в будь-який час. Деякі з них доступні лише в певний час. Тому, коли складається розклад, це треба враховувати. Інша річ – це кількість занять і курсів, що пропонуються. Кількість занять і курсів у розкладі університету дуже велика. Інша річ – наявність аудиторій, бюджетування та багато іншого [1].

Задача складання розкладу навчальних курсів полягає у пошуку найкращого розкладу, що задовольняє заданим критеріям, як набору заданих ресурсів, які можуть містити викладачів, групи студентів, аудиторії або лабораторії. Розробка розкладу є фундаментальною вимогою для здорового функціонування всіх навчальних та адміністративних підрозділів навчального закладу. Однак такі фактори, як наявність годин, кількість предметів і розподіл викладачів, роблять проблему розкладу дуже складною. Це робота має на меті розглянути декілька алгоритмів оптимізації, які можуть бути застосовані як можливі рішення для проблеми складання розкладу занять студентів навчальних закладів (НЗ). Розглянуті алгоритми враховують вимоги інституційних обмежень щодо управління розкладом занять.

Об'єктом дослідження є процес генерації розкладу в НЗ за допомогою алгоритмів оптимізації.

Предметом дослідження є розробка серверної частини вебзастосунку для автоматизованого складання та керування розкладом, а також розробка алгоритмів, які формують розклад згідно з вказаними критеріями.

Метою роботи є розробка та впровадження системи для автоматизованої генерації розкладу (АГР) для НЗ з метою поліпшення швидкості та якості генерації, уникнення людських помилок.

Для досягнення поставленої мети у роботі вирішувалися такі завдання:

- аналіз процесу генерації розкладу занять НЗ;
- розробка вимог до системи АГР;
- розробка архітектури та функціональних можливостей системи;
- розробка та реалізація програмного забезпечення системи;
- проведення тестування та оцінка ефективності системи.

Методика дослідження охоплює аналіз літературних джерел, розробку прототипу системи, її тестування та оцінку ефективності.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основні поняття

Генерація розкладу – це процес, пов'язаний зі складанням розкладу, в якому події, розташовані відповідно до часу, коли вони відбуваються, повинні відповідати часовим обмеженням кожного об'єкта, розміщеного в таблиці. Ці курси зазвичай викладаються різними лекторами в різних аудиторіях, щоб вказати деякі часові умови в своїх лекціях. Враховуючи всі обмеження та деталі для кожного курсу, класична ручна система складання розкладу займає багато часу, вимагає великих ресурсів, включає багато кроків і потребує повторної обробки одних і тих же даних кілька разів [2].

Розклад – це впорядкована в часі множина навчальних занять.

Заняття – це плановані академічні заняття, які відбуваються в рамках навчального процесу. А також мають містити список груп, вчителів, аудиторій, кількість разів на тиждень та рекомендований час для проведення. Заняття закріплені за дисциплінами.

Пара – це часовий інтервал, під час якого викладач проводить навчальний процес зі студентами. Воно повинно містити наступну інформацію: день тижня, номер пари, тип пари (чисельник, знаменник, загальна), та номер аудиторії. Пара закріплена за заняттям.

Аудиторія – є приміщення, де проводяться заняття. Для кожної аудиторії відомо кількість місць. Аудиторія закріплена за певною кафедрою або є загальною для НЗ.

Кафедра – це структурний підрозділ вищого навчального закладу. Кафедри мають перелік спеціальностей, за якими навчають студентів, перелік аудиторій закріплених за кафедрами та складаються з викладачів.

Спеціальність – це конкретна галузь знань або дисципліна, в якій студенти здобувають фахову освіту. Вона прикріплена до кафедри та має назву та код.

Дисципліна – це окремий предмет або курс, який закріплений за певною спеціальністю та має назву та семестр.

Група – це набір студентів, які закріплені до однієї спеціальності. Складаються з назви, кількості студентів.

Викладач – це професійний педагог, який працює в НЗ і має на меті надати якісну освіту студентам. Викладач закріплений за кафедрою та має ПІБ.

Розклад генерується на тиждень в залежності від кількості проведення занять. Так, наприклад якщо потрібно проводити заняття один раз на два тижні – то його можна один раз вставити по чисельнику або знаменнику. Тобто через кожен тиждень змінюється час проведення (чисельник або знаменник). А якщо пара загальна, то вона проводиться, як по чисельнику, так і по знаменнику.

При генерації розкладу мають враховуватися наступні умови:

- вікна між парами у груп та вчителів;
- накладання пар у груп, вчителів та аудиторій;
- побажання на проведення заняття в конкретний час;
- проведення заняття у вказаних аудиторіях;
- врахування кількості студентів та типу занять при виборі аудиторії;
- генерація розкладу в межах встановленої кількості днів та занять.

1.2 Огляд наявних методів

Через складність цієї проблеми координатори розкладу витрачають значну кількість часу на пошук найкращого рішення. Однак, навіть якщо вони мають великий досвід, знайдене рішення може не бути оптимальним через велику кількість можливих комбінацій. Таким чином, розподіл навантаження між різними викладачами в академічному закладі є проблемою комбінаторної природи. Загалом, розв'язання таких задач та отримання точних оптимальних розв'язків є обчислювальною нерозв'язною проблемою.

Тому задача складання розкладу занять в університеті є прикладом Non-Polynomial (NP) – важкої задачі. Це задачі, які не мають конкретного ефективного розв'язку. У випадку з розкладом занять не існує конкретного алгоритму, який можна було б використати для складання розкладу, оскільки кожен навчальний заклад має свої особливі обмеження. Водночас коли це робиться вручну, отриманий результат залежить як від початкового підходу, так і від досвіду комітету зі складання розкладу [3].

Цільова функція для цієї задачі повинна відображати основні критерії для оптимізації. Основними критеріями оцінки для розкладу є:

- мінімізація кількості вікон між парами: зменшити простій часу викладачів та груп;
- мінімізація накладань занять: передбачає уникнення перекриття між заняттями, які мають спільних груп, викладачів, або проводяться в одній аудиторії;
- врахування пропозицій щодо часу проведення занять.

Отже, цільова функція для задачі складання розкладу – це сума штрафних функцій, які показують наскільки порушується обмеження при генерації розкладу. Штрафні функції можуть мати різну вагомість, в залежності від того, наскільки важливим є дотримання певного обмеження, Ведемо наступні позначення:

- n_g – кількість груп;
- n_t – кількість викладачів;
- n_a – кількість аудиторій;
- n_s – кількість занять;
- D – кількість навчальних днів;
- P – кількість занять у день;
- Y – тип заняття;
- $G = [g_1, g_2, \dots, g_{n_g}]$ – розклад відносно груп;
- $T = [t_1, t_2, \dots, t_{n_t}]$ – розклад відносно викладачів;
- $A = [a_1, a_2, \dots, a_{n_a}]$ – розклад відносно аудиторій;

- $C = [c_1, c_2, \dots, c_{nc}]$ – загальний розклад;
- $R = [r_{i,1}, r_{i,2}, \dots, r_{i,nr}]$ – масив рекомендованого часу проведення занять, де i – номер заняття, а nr – кількість рекомендованого часу;
- $S = [s_{i,1}, s_{i,2}, \dots, s_{i,ns}]$ – масив пар для заняття, де i – номер заняття, а ns – кількість пар у занятті;
- $X = [x_1, x_2, \dots, x_{nx}]$ – розклад окремої групи, викладача або аудиторії, де nx – кількість занять в окремій групі, викладача або аудиторії.

Введемо наступні функції для більшого комфорту (1.1 – 1.3).

$$day(x) = \text{день тижня пари } x, \quad (1.1)$$

$$pair(x) = \text{номер пари } x, \quad (1.2)$$

$$type(x) = \text{тип пари } x. \quad (1.3)$$

Отже, ці функції (1.1 – 1.3) повертають певні параметри у переданій парі.

Тепер приведемо саме штрафні функції.

$f_w(X, y)$ – функція, яка розраховує вікна між парами та має відповідний ваговий коефіцієнт k_1 . Функція розраховує різницю між наступною парою та попередньою в межах одного дня. Спочатку треба порахувати вікна окремо по чисельнику, знаменник та загальній. домовимося так, що чисельнику відповідає значення 1, знаменнику 2, загальній 3. Слід також зазначити, що перед запуском цієї функції, обов'язково потрібно відсортувати масив пар за таким принципом – спочатку за днем тижня, потім за номер пари, потім за типом пари. Отримуємо таку формулу для підрахунку вікон (1.4):

$$f_w(X, y) = \sum_{i=1}^{nx-1} \begin{cases} k_1 \cdot (pair(x_{i+1}) - pair(x_i)), \text{ якщо} \\ day(x_{i+1}) = day(x_i) \text{ та } type(x_{i+1}) = type(x_i) = y, \\ 0, \text{ інакше.} \end{cases} \quad (1.4)$$

$f_1(X)$ – функція, яка підраховує вікна по всім типам пар (1.5):

$$f_1(X) = f_w(X, 1) + f_w(X, 2) + f_w(X, 3), \quad (1.5)$$

$f_2(X)$ – функція, яка розраховує накладання занять та має відповідний ваговий коефіцієнт k_2 . Накладанням занять можна вважати тоді, коли в один і той же час встановлено дві або більше пари. Виняткові ситуації це коли ці дві пари чисельник та знаменник. Функція, яка розраховує накладання занять (1.6):

$$f_2(X) = k_2 \cdot \sum_{i=1}^{nx} \sum_{j=1}^{nx} \begin{cases} 1, \text{якщо } type(x_i) = 1 \text{ та } type(x_j) = 1 \text{ або } 3, \\ 1, \text{якщо } type(x_i) = 2 \text{ та } type(x_j) = 2 \text{ або } 3, \\ 1, \text{якщо } type(x_i) = 3 \text{ та } type(x_j) = 1 \text{ або } 2 \text{ або } 3, \\ 0, \text{інакше.} \end{cases} \quad (1.6)$$

f_3 – функція, яка розраховує дотримання рекомендацій щодо часу проведення заняття та має відповідний ваговий коефіцієнт k_3 . Ця функція перевіряє день тижню та номер пари на співпадіння. Функція виглядає наступним чином (1.7):

$$f_3 = k_3 \cdot \sum_{i=1}^{nc} \sum_{j=1}^{nr} \begin{cases} 1, \text{якщо } day(r_{i,j}) \neq day(s_{i,j}) \\ \text{або } pair(r_{i,j}) \neq pair(s_{i,j}), \\ 0, \text{інакше.} \end{cases} \quad (1.7)$$

F – це загальна формула підрахунку штрафу для розкладу. Було описано всі штрафні формули (1.5 – 1.7), а отже їх можна звести в загальну формулу підрахунку штрафу (1.8):

$$F = \sum_{i=1}^{ng} f_1(g_i) + f_2(g_i) + \sum_{i=1}^{nt} f_1(t_i) + f_2(t_i) + \sum_{i=1}^{na} f_2(a_i) + f_3, \quad (1.8)$$

Змінюючи вагові коефіцієнти можна коригувати в яку сторону приділяти більше уваги чи вікнам, чи накладанню, чи рекомендованому часу.

Задача оптимального планування навчальних занять є задачею комбінаторного типу, і її характерною особливістю є велика розмірність. Крім того, завдання ускладняється наявністю великої кількості обмежень складної форми. Наразі не існує універсальних методів розв'язання таких задач. Класична теорія розкладів включає лише вузьке коло добре формалізованих задач, які зазвичай зводяться до задачі комівояжера, транспортної тощо. Пряме застосування методів до задачі планування навчальних занять неможливе. Однак існує ряд евристичних методів і методів грубої сили, які цілком піддаються програмуванню [4].

Як правило, застосовуються евристичні методи, які базуються на використанні різних евристик або евристичних алгоритмів для пошуку оптимального розв'язання задачі планування. Евристики використовують інтуїтивні припущення, які не підкріплені відповідним математичним обґрунтуванням. Такий підхід дозволяє дещо прискорити пошук найкращого розкладу. Однак він не гарантує досягнення глобальних екстремумів. Тому виникає проблема оцінки близькості локального екстремуму до глобального. У деяких роботах така оцінка здійснюється шляхом порівняння розкладу, отриманого евристичним методом, з розкладом, отриманим методом грубої сили. Попри цей недолік, даний метод є ефективним для пошуку розв'язку у випадку, коли неможливо або важко знайти найкращий розв'язок [4].

Іншим підходом до розв'язання задачі оптимального складання розкладу є використання методів, заснованих на обмеженнях, які зводяться до задачі задовільнення обмежень. Необхідно розподілити час і аудиторію в комірках розкладу так, щоб задовольнити множину обмежень. Як правило, для цього встановлюється багато правил. Цей метод передбачає "жорсткі" і "м'які" обмеження. Дотримання жорстких обмежень є обов'язковим, а м'яких – бажаним. На кожній ітерації розміщується одна група або курс. Якщо групі

або курсу неможливо призначити час або аудиторію для ітерації, одне з м'яких обмежень послаблюється. Процес продовжується до тих пір, поки не стане неможливо призначити час або аудиторію. Недоліком методу є те, що процес доводиться завершувати вручну, оскільки не завжди є можливість організувати групу або курс. Перевагою цього методу є зменшення простору пошуку [4].

Існує також підхід до розв'язання складних комбінаторних задач цілочисельного програмування, до класу яких відноситься задача планування тренувальних занять, описаний в роботах. Розв'язання цієї задачі здійснюється за допомогою нейронних мереж. Недоліком цього методу є складність вибору початкового стану нейронної мережі [4].

1.2.1 Алгоритм табу пошуку

Табу пошук (ТП) – це метаевристичний підхід для дослідження простору розв'язків за межами локальної оптимальності. Основні ідеї табу-пошуку базуються на ідеях, запропонованих Фредом Гловером (1977, 1986). За останні 2 десятиліття з'явилося більше сотні робіт, присвячених застосуванню методу пошуку за заборонами (Tabu Search). Цей метод став дуже популярним завдяки своїй здатності надавати рішення, дуже близькі до найкращих або майже найкращих оптимальних. ТП забезпечують дуже гнучку поведінку пошуку завдяки використанню адаптивної пам'яті [5].

Основи ТП:

- спосіб використання пам'яті в ТП полягає в тому, щоб класифікувати підмножину ходів у сусідній околиці як заборонені;
- околиця будується для визначення суміжних рішень, які можуть бути досягнуті з поточного рішення;
- класифікація, зокрема, залежить від того, наскільки давно певні компоненти ходу або рішення, які називаються атрибутами, брали участь у генеруванні минулих рішень;

– список табу, в якому записані заборонені ходи, називається ходами табу;

– якщо табу хід має більш задовільну оцінку, коли він призводить до кращого розв'язку, ніж будь-який з відвіданих до цього часу, то його табу класифікацію можна скасувати [5].

Алгоритм ТП отримує високоякісне рішення в короткий час. Якщо він не може знайти найкраще оптимальне рішення, він запропонує рішення, яке є найближчим до найкращого. Завдяки цьому дослідженню, використовуючи алгоритм пошуку за допомогою табу, можна легко реалізувати генератор університетського розкладу [5].

1.2.2 Алгоритм імітації відпалу

Метод імітаційного відпалу (ІВ) був вперше запропонований Кіркпатріком. Метод відпалу – це спроба досягти низькотемпературного стану шляхом розплавлення речовини, а потім повільного зниження температури. Наступне рівняння дає ймовірність будь-якого фактичного стану:

$$p(x) = e^{\frac{-Df \cdot (x)}{k \cdot T}}, \quad (1.9)$$

де x – конфігурація енергії;

k – стала Больцмана;

T – температура [6].

У випадку оптимізаційної задачі стандартна процедура оптимізації СА складається з наступних кроків:

– згенеруйте вектор початкових розв'язків – виберіть випадковим чином початковий розв'язок для системи, що оптимізується, а потім обчисліть цільову функцію;

– ініціалізуйте температуру – початкове значення температури T є важливим параметром для успішного застосування ІВ. Якщо значення занадто велике, то для збіжності потрібно більше зменшення. Якщо занадто мале, процес пошуку може бути не досконалим, так що точки потенційного глобального оптимуму можуть бути перевищені;

– виберіть новий розв'язок в околі поточного розв'язку – новий розв'язок $x_0 + \Delta x$ приймається як новий останній розв'язок в залежності від T . Цільова функція для $x_0 + \Delta x$ подається у вигляді $f(x_0 + \Delta x)$ та $f(x_0)$ відповідно;

– оцінка нового розв'язку – якщо $f(x_0 + \Delta x) \leq f(x_0)$, $x_0 + \Delta x$ приймається і замінює x_0 , оновити наявний оптимальний розв'язок і переходимо до останнього кроку. З іншого боку, якщо $f(x_0 + \Delta x) \geq f(x_0)$, $x_0 + \Delta x$ також може бути прийнято з ймовірнісним базисом на основі рівняння 2.9;

– періодичне зниження температури – на час пошуку процесу температура T знижується, в результаті чого на початку пошуку ймовірність прийняття погіршених ходів висока, і вона поступово зменшується;

– повторюйте кроки, доки не буде виконано критерій зупинки – обчислення припиняється, коли критерій зупинки виконано. В іншому випадку кроки повторюються [6].

Переваги алгоритму імітації відпалу це простота реалізації та забезпечення оптимальні рішення широкого кола проблем. Недоліками цього алгоритму є для складних задач процес симуляції може зайняти багато часу, потреба у налаштуванні параметрів, відсутність гарантії знаходження глобальної оптимальності.

Алгоритм імітації відпалу можна застосувати для задачі складання розкладу занять, щоб отримати оптимальне рішення яке врахує всі потреби та обмеження.

1.2.3 Генетичний алгоритм та його модифікації

Генетичні алгоритми (ГА) – це потужні інструменти оптимізації загального призначення, які моделюють принципи еволюції. Вони часто здатні знаходити глобально оптимальні рішення навіть у найскладніших пошукових просторах. Вони оперують популяцією закодованих рішень, які відбираються відповідно до їх якості, а потім використовуються як основа для нового покоління рішень, знайдених шляхом комбінування (кросинговеру) або зміни (мутації) поточних особин. Традиційно механізм пошуку був незалежним від домену, тобто оператори кросинговеру та мутації не знають, яким буде найкраще рішення [1].

Принцип роботи канонічного ГА. Основними етапами є генерація популяції розв'язків, знаходження цільової функції та функції пристосованості, а також застосування генетичних операторів [1].

ГА опрацьовує декілька розв'язків одночасно. Отже, на першому кроці за допомогою псевдовипадкових генераторів генерується популяція з P хромосом, які називаються особинами, кожна з яких представляє можливий розв'язок. Це представлення вектора розв'язку у просторі розв'язків і називається початковим розв'язком. Це забезпечує надійність та незміщеність пошуку, оскільки він починається з широкого діапазону точок у просторі розв'язків [1].

На наступному кроці оцінюються окремі члени, хромосоми популяції, представлені рядком, щоб знайти значення цільової функції. Це виключно специфікація задачі. Цільова функція відображається у фітнес-функцію, яка обчислює значення пристосованості для кожної хромосоми. Після цього застосовуються оператори ГА [1].

Розмноження або відбір зазвичай є першим оператором, який застосовується до популяції. Це операція, яка створює більше копій кращих хромосом у новій популяції. Таким чином, в операції розмноження процес природного відбору призводить до того, що ті хромосоми, які кодують

успішні структури, виробляють копії частіше. Для того, щоб підтримувати покоління нової популяції, необхідно відтворювати хромосоми в поточній популяції. Щоб отримати кращі хромосоми, вони повинні бути створені з найбільш пристосованих хромосом попередньої популяції [1].

В літературі з ГА існує декілька операторів розмноження, але основна ідея в них полягає в тому, що з поточної популяції вибираються рядки з вищим за середнє значенням придатності, і їх кілька копій вставляються в пул спарювання ймовірнісним чином [1].

Операція кросинговеру використовується для рекомбінації двох хромосом для отримання кращої. При кросинговері процес рекомбінації створює різні хромосоми в наступних поколіннях шляхом об'єднання матеріалу з двох хромосом попереднього покоління. Під час розмноження хороші хромосоми в популяції ймовірно отримують більшу кількість копій, і формується пул спарювання. Важливо зазначити, що на етапі розмноження нові хромосоми зазвичай не утворюються. В операторі кросинговеру нові хромосоми створюються шляхом обміну інформацією між ланцюжками пулу спарювання [1].

У літературі з ГА існує багато операторів кросинговеру. Найпоширенішими з них є кросинговер за одним та двома точками. У більшості операторів кросинговеру випадковим чином вибираються два рядки з пулу спарювання і відбувається обмін деякими частинами рядків між ними. Операція кросинговеру виконується на рівні рядків шляхом випадкового вибору двох рядків для операцій кросинговеру [1].

Мутація додає нову інформацію у випадковий спосіб до процесу генетичного пошуку і, зрештою, допомагає уникнути зациклення на локальних оптимумах. Це оператор, який вносить різноманітність у популяцію, коли популяція має тенденцію до однорідності через багаторазове використання операторів розмноження та кросинговеру. Мутація може призвести до того, що хромосоми будуть відрізнятися від хромосом їхніх батьків. Мутація – це процес випадкового порушення

генетичної інформації. Вони діють на рівні бітів. Коли біти копіюються з поточного рядка в нові хромосоми, існує ймовірність того, що кожен біт може стати мутований. Ця ймовірність зазвичай є досить малою величиною, яка називається ймовірністю мутації p_m . Необхідність мутації полягає у створенні точки в околиці поточної точки. Мутація також використовується для підтримки різноманітності в популяції [1].

Ці три оператори прості і зрозумілі. Оператор розмноження вибирає хороші хромосоми, а оператор кросинговеру рекомбінє хороші підрядки з хороших підрядків разом, сподіваючись створити кращу підрядкову хромосому. Оператор мутації змінює рядок локально, сподіваючись отримати кращу хромосому. Хоча жодне з цих тверджень не гарантується і/або не перевіряється при створенні хромосоми, очікується, що якщо будуть створені погані хромосоми, вони будуть усунуті оператором розмноження в наступному поколінні, а якщо будуть створені хороші хромосоми, то вони будуть все більше підкреслюватись [1].

Переваги генетичного алгоритму:

- знаходження оптимальних рішень для складних задач, які не можуть бути розв'язані традиційними методами;
- може застосовуватися для різних типів проблем та задач, що робить його універсальним;
- алгоритм може бути модифікованим та адаптованим для конкретних задач;
- знаходження оптимального рішення швидше, ніж інші алгоритми оптимізації;
- шукають найкраще рішення у широкому просторі можливих рішень.

Недоліки ГА:

- складність налаштування параметрів, через це залежить швидкість та ефективність роботи алгоритму;
- можуть потрапити в локальні оптимуми;
- вимагають значних обчислювальних ресурсів та часу.

Відбір Больцмана заснований на методах ентропії та вибірки, які використовуються в моделюванні Монте-Карло. Це допомагає у вирішенні проблеми передчасної конвергенції. Імовірність вибору найкращого рядка дуже висока, при цьому вона виконується за дуже менший час. Однак існує ймовірність втрати інформації. Нею можна керувати за допомогою елітарності. Вибір елітарності був запропонований К. Д. Джонгом (1975) для поліпшення характеристик вибору колеса рулетки. Це гарантує, що елітарна людина в поколінні завжди передається наступному поколінню. Якщо індивід, що має найвищу пристосованість, не присутній у наступному поколінні після звичайної процедури відбору, то елітарний також включається до наступного покоління автоматично [7].

Мотивація паралельних ГА полягає в поліпшенні обчислювального часу та якості рішень за допомогою розподілених осіб. Паралельні ГА поділяються на три великі категорії, такі як паралельні ГА master-slave, fine grained та coarse grained. У паралельному ГА master-slave обчислення фітнес-функцій розподіляється між кількома процесорами. У fine grained ГА паралельні комп'ютери використовуються для вирішення реальних проблем. Генетичні оператори пов'язані зі своїм сусідством. Однак допускається взаємодія між особинами. У coarse grained ГА здійснюється обмін особинами між субпопуляціями. Контрольні параметри також передаються під час міграції. Основними проблемами паралельних ГА є максимізація пропускну здатності пам'яті та організація потоків для використання потужності графічних процесорів [7].

1.2.4 Нейронні мережі

Штучні нейронні мережі – це математичні моделі, які імітують роботу людського мозку. Вони здатні розв'язувати складні задачі шляхом визначення залежностей між великою кількістю вхідних та вихідних даних. Нейронні мережі можуть бути застосовані для розв'язання задач у різних

галузях: класифікації, регресії, кластеризації, розпізнавання образів, обробка мови, аналіз даних тощо [8]-[10].

Нейронна мережа складається з нейронів та зв'язків між ними. Кожен нейрон приймає вхідні сигнали від інших нейронів чи зовнішнього середовища, обробляє їх та генерує вихідний сигнал. Зв'язки між нейронами відповідають за передачу сигналів між ними, а їх ваги визначають внесок кожного нейрона у формування вихідного сигналу [8]-[10].

Математична модель нейронної мережі базується на використанні лінійної алгебри та математичного аналізу. Кожен нейрон у мережі можна описати наступною формулою (1.10):

$$z = \sum_{i=1}^n w_i \cdot x_i + b, \quad (1.10)$$

де w_i – вага, яка присвоюється вхідному сигналу x_i ;

x_i – значення вхідного сигналу;

n – кількість вхідних сигналів;

b – зсув [14]-[16].

Далі до результату z застосовується функція активації, яка дозволяє моделювати нелінійні залежності між вхідними та вихідними даними (1.11):

$$y = f(z), \quad (1.11)$$

де y – вихідний сигнал нейрона;

f – функція активації [8]-[10].

Нейрони групують у шари. Кожен шар містить групу нейронів, які взаємодіють з нейронами попереднього та наступного шарів. Виділяють наступні види шарів:

– вхідний шар: приймає вхідні дані;

– приховані шари: обробляють вхідні дані та передають результати до наступного шару;

– вихідний шар: видає результати роботи мережі [8]-[10].

Нейронні мережі можна класифікувати за різними ознаками. Основні класифікації НМ:

а) за кількістю шарів:

- 1) одношарові;
- 2) багатошарові;

б) за способом навчання:

- 1) НМ з учителем;
- 2) НМ без учителя;
- 3) НМ з підсиленням;

в) за типом зв'язків між нейронами:

- 1) повнозв'язні;
- 2) неповнозв'язні;
- 3) ієрархічно інтерактивні;
- 4) слабозв'язні;

г) за типом функції активації:

- 1) лінійна;
- 2) порогова;
- 3) гіперболічний тангенс;
- 4) сигмоїдна логістична;
- 5) косинусоїдальна;
- 6) синусоїдальна [8]-[10].

Найбільш відомі архітектури НМ:

- прямолінійні нейронні мережі;
- згорткові нейронні мережі;
- рекурентна нейронна мережа [8]-[10].

Довга короткочасна пам'ять (Long short-term memor) – це вид рекурентних нейронних мереж, який дозволяє моделювати та працювати з

послідовностями даних довільної довжини та здатна зберігати інформацію про довготривалі залежності вхідних даних [11].

Стандартні RNN можуть мати проблеми зі збереженням довготривалих залежностей, які виникають між далекими моментами часу в послідовності вхідних даних. LSTM розв'язує цю проблему за допомогою спеціальної структури, яка дозволяє мережі зберігати попередні значення стану та використовувати їх при обчисленні наступних значень [11].

LSTM, використовуючи спеціальний механізм, дозволяє контролювати потік інформації через різні шари мережі. Він складається з трьох основних блоків: блоку забування, блоку входу та блоку виводу [11].

Блок забування відповідає за розв'язання проблеми зниклого градієнту, дозволяючи мережі забувати непотрібну інформацію. Блок входу контролює, яка нова інформація повинна бути додана до внутрішнього стану мережі, а блок виводу контролює, яку інформацію мережа повинна виводити [11].

LSTM використовується в багатьох задачах обробки природних мов (наприклад, у машинному перекладі, розпізнаванні мовлення), а також в задачах прогнозування часових рядів та аналізу даних [11].

До переваг нейронних мереж можна віднести:

- нейронні мережі можуть навчатися на основі вхідних даних та знаходити неявні взаємозв'язки між ними;
- НМ можуть адаптуватися до змін у вхідних даних;
- можливість ефективно працювати з великими обсягами даних;
- можуть бути застосовані до розв'язання складних задач;
- нейронні мережі можуть робити узагальнення
- нейронні мережі можуть обробляти нечітку та неповну інформацію;
- можуть обробляти багато даних одночасно та забезпечувати високу продуктивність [8]-[10].

Недоліки НМ:

- необхідність великої кількості даних для навчання;

- високі вимоги до обчислювальних ресурсів;
- наявність великої кількості параметрів та складність їх налаштування;
- складність вибору архітектури
- вразливість до шуму даних;
- НМ можуть бути складними для інтерпретації та розуміння [8]-[10].

1.3 Порівняльний аналіз наявних методів вирішення завдання

Було складено порівняльну таблицю розглянутих алгоритмів (табл. 1.1).

Таблиця 1.1 – Порівняльна характеристика розглянутих алгоритмів

Характеристика / Назва алгоритму	Табу пошук	Імітація відпалу	Генетичний алгоритм	Штучна нейронна мережа (LSTM)
1	2	3	4	5
Швидкість знаходження оптимального рішення	Залежить від кількості обмежень	Залежить від функції прийняття	Залежить від параметрів	Залежить від розміру вхідних даних
Потрібність у вибірці даних для навчання та тренування	Ні	Ні	Ні	Так
Залежність від початкового наближення	Висока	Висока	Середня	Низька

Кінець таблиці 1.1

1	2	3	4	5
Складність реалізації	Низька	Низька	Середня	Висока
Потреба в налаштуванні параметрів	Так	Так	Так	Так

Можна зробити висновок, що генетичний алгоритм безсумнівно швидший та точніший в ході роботи аніж табу пошук та імітація відпалу. Проте ГА потребує більше обчислювальних ресурсів та набагато складніший в реалізації. Якщо порівняти ГА зі штучною нейронною мережею (LSTM), то якщо питання стоїть у виборці даних для навчання, то ГА не потребує їх. А якщо немає проблем з вибіркою даних, то нейрона мережа (LSTM) виграє в часі роботи.

1.4 Огляд наявних систем

1.4.1 Огляд програми Untis

Створення та адміністрування розкладу є складним завданням. Untis – це наше базове програмне забезпечення, яке автоматично створює розклади відповідно до ваших специфікацій. Untis допоможе вам створити розклад, відповідний для вашої школи [12].

Наші модулі розширюють можливості базового пакета, підтримуючи вас у різних завданнях навіть за межами створення розкладу (наприклад, у плануванні заміни) [12].

А Untis Express пропонує вам компактну, просту у використанні програму розкладу, розроблену спеціально для невеликих шкіл з меншими складними вимогами [12].

Гнучкий генератор розкладу. Планувальник розкладу визначає та зважає критерії (наприклад, наявність або часові запити класів, вчителів, кімнат та уроків), відповідно до яких має бути створений розклад:

- коли заплановані заняття, а які не планувати;
- які номери доступні в який час;
- скільки днів тижня який предмет викладати в якому класі;
- який учитель може викладати в який день.

Untis генерує різні розклади відповідно до цих та інших критеріїв завдяки унікальному алгоритму, а планувальник вибирає найкращий розклад з декількох можливостей [12].

Ручне редагування. До або після завершення розкладу ви можете запланувати або змінити періоди вручну в будь-який час. Таким чином ви ще краще оптимізуєте свій розклад відповідно до ваших індивідуальних вимог або реагуєте на зміни в короткі терміни [12].

Інструмент діагностики. Інтегрований інструмент діагностики показує будь-які розбіжності в розкладі, які можуть виникнути, особливо після ручного редагування [12].

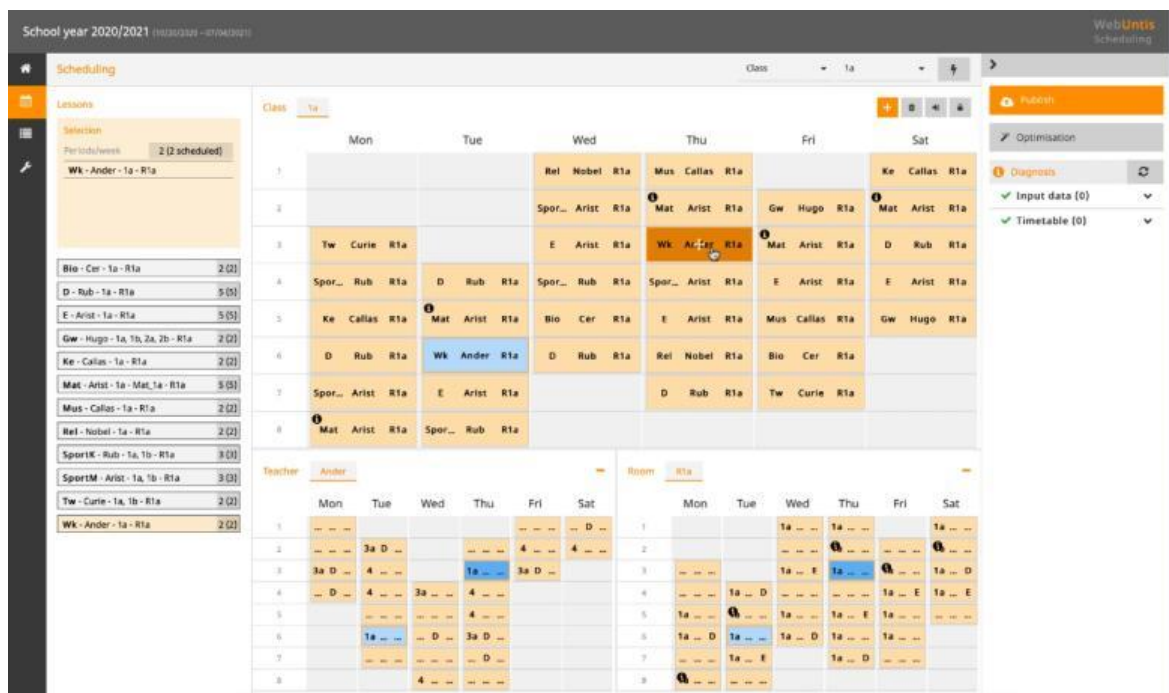


Рисунок 1.1 – Вікно програми Untis [12]

Переваги програми – це унікальний алгоритм генерації розкладу та можливість ручного редагування розкладу.

1.4.2 Огляд програми TimeTabler

TimeTabler – це швидка і зручна комп'ютерна програма, заснована на багаторічному досвіді розкладу, ретельно розроблена, щоб допомогти вам швидко і точно запланувати свій розклад [13].

TimeTabler повністю само перевіряється. Наприклад: ніколи не дозволить помилково виділити якогось одного вчителя в два класи одночасно; і завжди попередить вас, якщо ви почнете ставити два класи в одному приміщенні; або якщо ви вирішили поставити два окремих уроки математики в один день тощо. Тощо. Використовуючи високу швидкість комп'ютера в поєднанні з ефективними та перевіреними алгоритмами розкладу, TimeTabler дивиться вперед і показує вам на кожному етапі рекомендований список пріоритетів, які ви повинні розглянути [13].

Тобто він рекомендує, який урок ви повинні розмістити в розкладі, і де ви повинні його розмістити. На кожному етапі ви можете слідувати порадам програми або прийняти власне рішення [13].

Але на додаток до цієї механічної перевірки та визначення пріоритетів, TimeTabler зробить набагато більше – він зробить фактичне планування розкладу за вас:

- або інтерактивно (ви і машина працюєте разом);
- або автоматично (ви розслабляєтеся, поки машина виконує роботу);
- або обидва (мінняєтеся місцями між ними, коли забажаєте) [13].

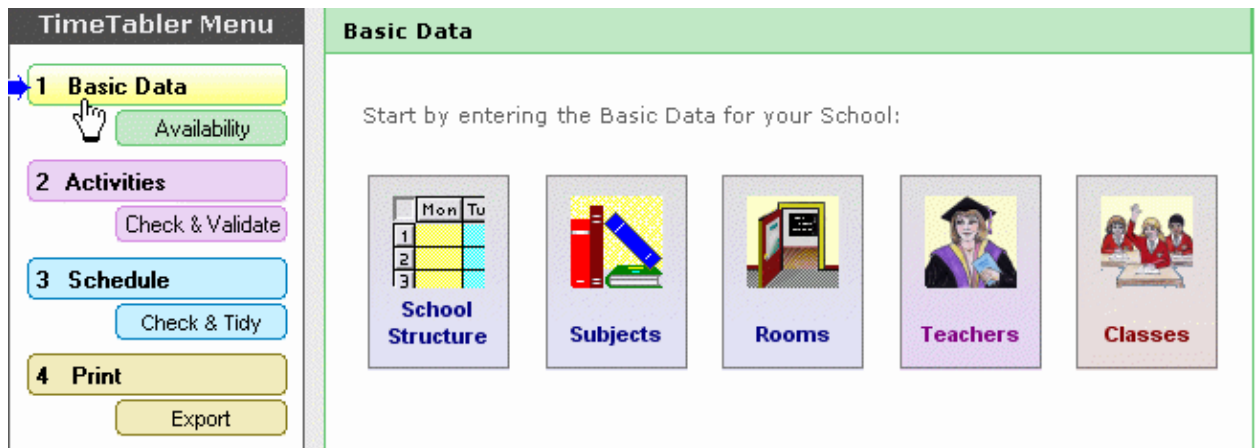


Рисунок 1.2 – Інтерфейс програми TimeTabler [13]

Переваги програми – вибір стратегії генерації розкладу, показує список пріоритетів та рекомендує як розмістити заняття.

1.4.3 Огляд програми TimeEdit

TimeEdit – інструмент для планування. Ми знаємо, що кожна установа різна. Ось чому ми створили найгнучкіший у світі інструмент для планування. Розклад курсу. Гнучкий розклад і розклад курсів для будь-яких типів курсів і програм на індивідуальному, груповому, класному та курсовому рівнях [14].

Генеральне планування. Забезпечте плавний процес, спланувавши приміщення, обладнання та доступність викладачів до того, як розпочнеться розклад курсу [14].

Призначення кімнати. Керуйте своїм навчальним розкладом і простором у кампусі в одному рішенні. Призначте кімнати до, під час або після планування курсу [14].

Гнучке управління ресурсами. У TE Core будь-що може бути ресурсом, тобто ви можете планувати та керувати чим завгодно, від кімнат чи обладнання до курсів і ліцензій на програмне забезпечення [14].

Вносьте зміни на ходу. Всяке трапляється. Вчителі хворіють, кімнати стають незамінними, а групи потрібно налаштовувати. TE Core створено для цієї реальності та вносить зміни в останню хвилину без проблем [14].

Гуртом або поштучно. Економте час завдяки потужній системі масового керування TE Core, яка дозволяє масово копіювати, змінювати та видаляти резервування [14].

Більше ніяких зіткнень. TE Core постачається як хмарна багатокористувацька система, що означає, що всі користувачі завжди мають доступ до оновлених даних у реальному часі. Іншими словами; привітайтеся зі 100% безконфліктним графіком [14].

Найкраща у своєму класі система дозволів. Детальна система дозволів і правила часу TE Core дозволяють вирішувати, що можна зарезервувати, коли, ким і як часто [14].

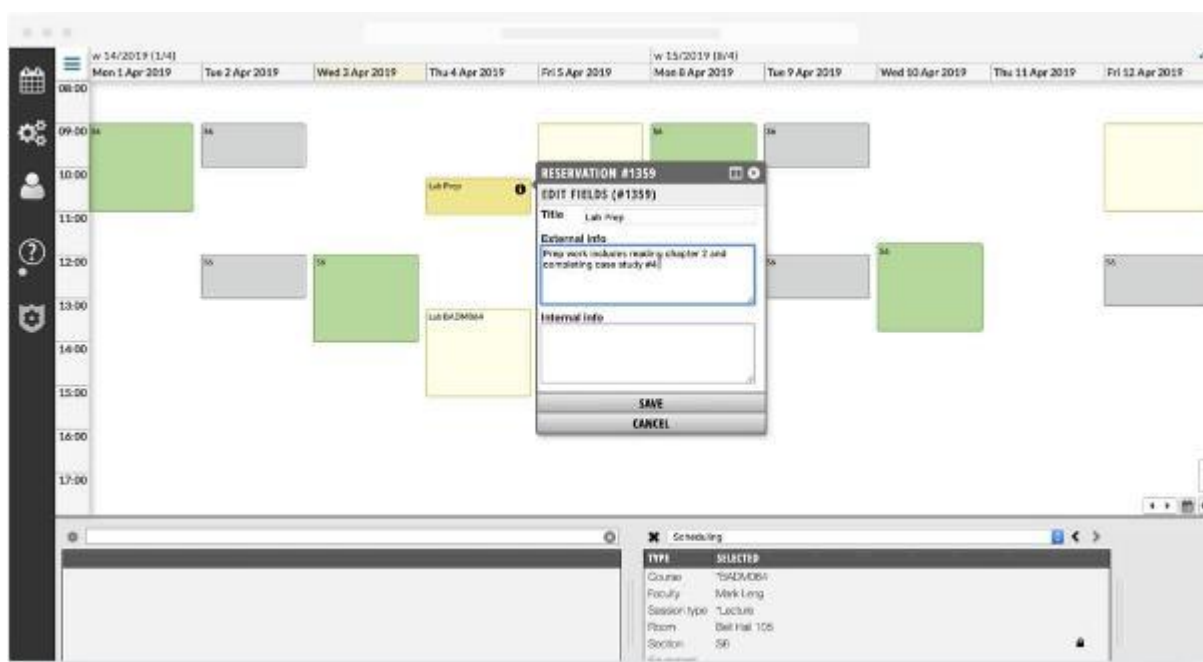


Рисунок 1.3 – Інтерфейс програми TimeEdit [14]

Переваги програми – дуже потужне керування розкладом, надійність та захист.

1.5 Порівняльна характеристики наявних систем

Ознайомившись з декількома програмами для автоматизованого складання розкладу, було створено їх порівняльну характеристику (табл. 1.2).

Таблиця 1.2 – Порівняльна характеристика наявних програмних засобів

Характеристика	Untis	TimeTabler	TimeEdit
1	2	3	4
Відкритий код	Ні	Ні	Ні
Операційна система	Кросплатформна	Кросплатформна	Кросплатформна
Підтримка різних мов	Багатомовний	Багатомовний	Багатомовний
Наявність ліцензії	Так	Так	Так
Адаптивність до різних типів установ	Так	Так	Так
Тип програми	Десктопна	Десктопна	Вебзастосунок
Автоматичне створення розкладу	Так	Так	Так
Можливість ручного редагування розкладу	Так	Так	Так

Кінець таблиці 1.2

1	2	3	4
Можливість імпорту/експорту даних	Так	Так	Так
Формат збереження даних	Різні формати (.unt, .xlsx, .xml)	Різні формати (.ttf або .ttb, .xlsx, .xml)	Різні формати (.xml, .csv, sql)
Редагування даних	Так	Так	Так
Фільтрація та сортування даних	Так	Так	Так
Формування звітів	Так	Так	Так

1.6 Формулювання завдання до роботи

Метою роботи є дослідження та реалізація серверної частини вебзастосунку для генерації та керуванням розкладу.

Вхідними даними для генерації розкладу за певною кафедрою будуть:

- дані про кафедру: код кафедри;
- базовий розклад: тобто масив занять груп, викладачів та аудиторій, якщо у них є пари на інших кафедрах;
- розмір штрафів: штраф за накладання, штраф за вікно для групи та вчителя, штраф за рекомендоване заняття;
- параметри розкладу: кількість навчальних днів та кількість занять на тиждень;
- параметри алгоритму: назва алгоритму та параметри його запуску;
- дані про аудиторії: код аудиторії, тип, місткість, закріплені кафедри;

– дані про заняття: код заняття, кількість раз на тиждень, тип, закріплені групи (код групи, кількість студентів), закріплені викладачі (код викладача), рекомендовані аудиторії (код аудиторії), рекомендований час (день тижня, номер пари).

Вихідними даними буде оптимальний розклад занять, складений для обраної кафедри, який мінімізує:

- накладання занять для груп, вчителів та аудиторій;
- вікна між заняттями для груп та вчителів.

Алгоритм для складання розкладу повинен мінімізувати значення визначеної цільової функції (1.8). Роботу алгоритму можна представити у вигляді наступної формули (1.12):

$$F \rightarrow \min, \quad (1.12)$$

де F – функція підрахунку штрафів у розкладі. Тобто показує фітнес-значення.

Для оцінювання якості результатів роботи алгоритму буде використовуватися описана вище цільова функція, яка показує кількість порушень заданих обмежень.

До системи керування розкладом наступні вимоги:

- перегляд та фільтрація даних;
- ручне створення даних;
- редагування та видалення даних;
- можливість автоматичного створення даних за допомогою вже готових відомостей;
- запуск та налаштування параметрів алгоритмів;
- можливість збереження та завантаження розкладу;
- перегляд графіків результатів роботи алгоритмів.

1.7 Висновок

У цьому розділі було проведено аналіз предметної області, визначено основні поняття, пов'язані з предметною областю й об'єктом дослідження та розробки. Було розглянуто проблему генерації та обмеження для розкладу занять.

Було досліджено наступні методи генерації розкладу: табу пошук, симуляція відпалу, генетичний алгоритм та його модифікації, нейронну мережу (LSTM). Було порівняно переваги та недоліки цих алгоритмів.

Проведено порівняльну характеристику наявних програмних засобів для генерації розкладу та було виявлено основні вимоги до власної системи.

Було сформульовано та поставлено завдання до роботи.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Вибір мови програмування для серверної частини

Для розробки вебсервера можна використовувати різні мови програмування залежно від ваших вимог, знань та уподобань. Далі будуть розглянуті деякі популярні мови програмування та фреймворки під них, які часто використовуються для створення вебсерверів.

2.1.1 JavaScript

JavaScript (JS) – це перш за все клієнтська мова, яка виконується у браузері. Але що робить його унікальним, так це те, що він також може працювати на стороні сервера з такими платформами, як Node.js [15].

Оскільки браузерні сценарії є найпопулярнішим використанням JS, як саме це працює? Вебсайти зазвичай містять вбудований код JavaScript, який запускається лише тоді, коли користувач фактично відвідує цю сторінку. JavaScript – це інтерпретована мова, а не компільована мова [15].

Компільовані мови, такі як Java і C ++, повинні бути скомпільовані, перш ніж вони зможуть працювати. Вихідний код проходить через компілятор, який переводить код в байт-код, зрозумілий машині [15].

Однак, оскільки JavaScript є інтерпретованою мовою, цей "компіляційний" крок повністю обходиться. Замість цього перекладач у вашому браузері (наприклад, Chrome, Safari, FireFox) може "інтерпретувати" код JavaScript і виконати його. З цієї причини JS часто називають скомпільованою мовою програмування «just-in-time» [15].

Переваги JS:

- невелике налаштування;
- ви можете почати кодувати прямо у своєму браузері. Працює в будь-якому веббраузері;

- не потрібно нічого завантажувати додатково. Працює на будь-якому пристрої (наприклад, мобільному телефоні, настільному комп'ютері, планшеті), тому ви можете створювати кросплатформні програми, які працюватимуть незалежно від того, який пристрій має користувач;

- на стороні клієнта може запускатися відразу в браузері, оскільки це інтерпретована мова і не потребує проходження через компілятор;

- оскільки це дуже популярна мова, є безліч ресурсів, місць, де можна поставити запитання та відповісти на них, отримати допомогу тощо. Stack Overflow наразі має 2 199 603 запитання лише щодо JavaScript [15].

Недоліки JS:

- старіші браузери можуть не підтримувати JavaScript;
- безпека: оскільки JS-код працює на комп'ютері користувача та доступний користувачеві для перегляду, існує ймовірність його використання для зловмисних цілей;

- кожен браузер може інтерпретувати код JS дещо по-різному. Користувачі можуть бачити різні речі залежно від браузера, який вони використовують [15].

2.1.1.1 Фреймворк ExpressJS

Фреймворк ExpressJS – це швидкий, неперевірений та мінімалістичний вебфреймворк, який забезпечує надійний набір функцій для розробки вебдодатків в екосистемі NodeJS / JavaScript [16].

ExpressJS – це фреймворк, подібний до Сінатри, який починається зі створення простих функцій, додаючи та розробляючи більш надійні компоненти та функції залежно від випадків використання вашої програми [16].

ExpressJS API займається вебзастосунком, HTTP-запитами та відповідями, маршрутизацією та проміжним програмним забезпеченням [16].

Особливості ExpressJS:

- пропонує потужну, надійну і високорозвинену систему маршрутизації, яка допомагає вашій програмі відповісти на запит клієнта через певну кінцеву точку;
- дозволяє розділяти довгі та роздуті маршрути на різні маршрутні файли та керувати ними незалежно через маршрутизатор ExpressJS, повне проміжне програмне забезпечення та систему маршрутизації;
- проміжне програмне забезпечення не має конкретного визначення. Це може бути що завгодно, залежно від варіанту використання та функціональних можливостей;
- проміжне програмне забезпечення виконується до того, як HTTP-запит досягне обробника маршруту або до того, як клієнт отримав відповідь;
- expressJS складається з серії проміжного програмного забезпечення. Насправді фреймворк по суті є серією викликів функцій проміжного програмного забезпечення. Це також дозволило дуже легко створювати та використовувати проміжне програмне забезпечення;
- забезпечує простий механізм налагодження, який полегшує налагодження додатків ExpressJS, визначаючи точну частину вебсторінки з помилкою [16].

2.1.1.2 Фреймворк NestJS

NestJS виявився найбільш швидкозростаючим фреймворком TypeScript для створення гнучких, масштабованих, великомасштабних і корпоративних серверних додатків за допомогою Node.js [17].

При створенні бекенд-додатків, які мають високий рівень тестування, добре структуровані і підтримувані за допомогою TypeScript, NestJS є найкращим вибором для провідних організацій [17].

Як повноцінний вебфреймворк з високою структурою та добре структурованими архітектурами, фреймворк NestJS має чудові функції, які роблять його масштабованим, тестованим та підтримуваним [17].

Особливості NestJS:

- побудований за допомогою TypeScript, і він є найпопулярнішим у цій екосистемі. Ця інтеграція значно зменшує кількість помилок, пов'язаних із перевіркою типів та невідповідностями типів під час використання JavaScript для створення великомасштабних та корпоративних програм;
- надає найпотужніший інструмент командного рядка. Інструмент може створювати та керувати будь-якою частиною фреймворку NestJS, вводячи команди терміналу. Command line program (CLI) є єдиною точкою істини для командних рядків для розробки додатків з NestJS без особливої взаємодії графічного інтерфейсу. За допомогою CLI можна створювати бази даних для створення модулів, контролерів і службових файлів, ввівши кілька команд;
- надає дуже зрозумілі та добре задокументовані посібники для початківців для створення простих і складних додатків за допомогою фреймворку NestJS;
- підтримує стиль розробки архітектури мікросервісів прямо з коробки, забезпечуючи належну інтеграцію багатьох популярних інструментів мікросервісів, таких як Kafka, gRPC, RabbitMQ тощо;
- підтримує різні популярні інструменти з коробки, що зменшує клопоти з розробкою та збільшує швидку розробку додатків. Найкращі інструменти розробника вже інтегровані, дотримуючись найкращих практик та галузевих стандартів. NestJS підтримує TypeORM, Mongoose, GraphQL, журналювання, перевірку, кешування, WebSockets та багато іншого прямо з коробки, і додаткова конфігурація не потрібна [17].

2.1.2 Python

Python вперше випущений у 1991 році, Python є об'єктно-орієнтованою мовою програмування, відомою тим, що вона проста у

вивченні, але надзвичайно потужна та є однією з найбільш швидкозростаючих мов у світі [18].

Оскільки це мова загального призначення, її можна використовувати практично для чого завгодно, що робить її чудовою першою мовою для початківців програмістів, науковців даних, інженерів робототехніки, веброзробників, інженерів DevOps тощо [18].

Існує широкий спектр проєктів, над якими ви можете працювати з Python. Наприклад, Python має понад чверть мільйона сторонніх пакетів, доступних в Python Package Index, що дозволяє знаходити та встановлювати бібліотеки та фреймворки Python, розроблені та спільно використовувані спільнотою Python [18].

Особливості Python:

- є простим і потужним відразу, але ви не переростаете його. Це означає, що розпочати роботу відносно швидко і легко, але є так багато речей, які ви можете з ним зробити. З такою кількістю бібліотек, модулів, фреймворків та варіантів використання для вивчення, він універсальний та гнучкий. Іншими словами, Python чудово підходить для початківців, але також чудово підходить для найбільш просунутих розробників та експертів;

- все, що потрібно для запуску програм Python це текстовий редактор (наприклад, VS Code, PyCharm Community Edition та інші) та Python, встановлені на вашому комп'ютері;

- код Python легко читати. Він чимось нагадує англійську мову, і є відступи, щоб полегшити відстеження речей. Не доведеться витратити багато часу, намагаючись зрозуміти складний синтаксис;

- безліч варіантів кар'єри доступні, коли ви знаєте Python. Вони охоплюють від веброзробки до робототехніки, науки про дані тощо;

- існує безліч ресурсів та підтримки спільноти. Можна ставити питання на Stack Overflow (або переглядати понад мільйон питань, на які вже були отримані відповіді) і брати участь у різних форумах (наприклад,

<https://python-forum.io/>), конференціях (наприклад, PyCon) та читати документацію Python [18].

Що можна зробити з Python:.

- веброзробка: оскільки існує так багато вебфреймворків Python (наприклад, Django, Flask, web2py), створення вебсайтів за допомогою Python може бути швидким і простим. Ці фреймворки Python можуть допомогти вам створювати надійні, масштабовані та підтримувані вебпрограми;

- вилучення даних: Python має багато інструментів вебскрейпінгу, таких як Selenium, і фреймворків, таких як Scrapy, які дозволяють отримувати та організовувати дані з різних вебсторінок. Це можна використовувати для порівняння цін, збору адрес електронної пошти, з'ясування того, що в тренді тощо;

- тестування: Python має вбудовану структуру модульного тестування під назвою PyUnit, що дозволяє легко переконатися, що ваш код працюватиме належним чином. І, звичайно, Python має сторонній фреймворк pytest, що робить тестування ще простішим;

- автоматизація: можна використовувати Python для написання програм, які виконують виснажливі завдання за лічені хвилини, а не години. Наприклад, ви можете автоматизувати такі завдання, як перейменування файлів і директорій, надсилання електронних листів, заповнення онлайн-форм та багато іншого;

- програми машинного навчання: можна використовувати Python для навчання моделей машинного навчання та створення таких речей, як рекомендаційні системи та програмне забезпечення для розпізнавання обличчя або голосу. Дві найпопулярніші бібліотеки Python і фреймворки для машинного навчання – це scikit-learn і TensorFlow [18].

2.1.2.1 Фреймворк Django

Django – це високорівневий вебфреймворк Python MVT (Model View Template) з відкритим вихідним кодом, який дозволяє швидко розробляти безпечні та підтримувані вебзастосунки [19].

Django слідує популярній архітектурі MVC (Model View Controller) з невеликими варіаціями. У той час як MVC розшифровується як контролер перегляду моделі, MVT Django означає шаблон перегляду моделі [19].

Особливості Django:

- відмінна інтерактивна документація Django дозволяє будь-якому новому розробнику легко і цікаво вибрати проєкт і закінчити його в рекордно короткі терміни;
- Django – це фреймворк, побудований на Python. Будучи найпростішою мовою програмування з численними пакетами та бібліотеками, ви можете швидко створювати вебсайти, вебзастосунки, науки про дані, машинне навчання та проєкти штучного інтелекту за допомогою Django та інших пакетів Python;
- за замовчуванням оптимізований для пошукових систем;
- має високу масштабованість;
- дуже захищений;
- з більш ніж десятирічною продуктивністю та послідовністю, не жарт, що Django ретельно протестований;
- забезпечує швидкий розвиток;
- поставляється з попередньо вбудованою областю адміністратора. Ця функція означає, що вам не обов'язково створювати нову область адміністратора для керування вебзастосунком [19].

2.1.2.2 Фреймворк Flask

Flask – це фреймворк вебзастосунків, написаний на Python. Його розробив Армін Роначер, який очолював групу міжнародних ентузіастів Python під назвою Рооссо. Flask базується на наборі інструментів Werkzeug WSGI та механізмі шаблонів Jinja2 [20].

Інтерфейс шлюзу вебсервера Web Server Gateway Interface (WSGI) використовувався як стандарт для розробки вебдодатків на Python. WSGI – це специфікація загального інтерфейсу між вебсерверами та вебдодатками [20].

Werkzeug – це набір інструментів WSGI, який реалізує запити, об'єкти відповіді та службові функції. Це дозволяє створити на ньому вебфрейм. Фреймворк Flask використовує Werkzeug як одну зі своїх основ [20].

jinja2 – популярний механізм шаблонів для Python. Система вебшаблонів поєднує шаблон із певним джерелом даних для відтворення динамічної вебсторінки [20].

Особливості Flask:

- маршрутизація. Flask використовує підхід на основі декоратора для маршрутизації URL. Ви можете визначати маршрути та пов'язувати їх із функціями, які обробляють запити;
- шаблони. Flask підтримує шаблони Jinja2, що дозволяє відокремити логіку презентації від коду програми. Шаблони Jinja2 надають такі потужні функції, як успадкування шаблонів, макроси та фільтри;
- розширення Flask. Flask має багату екосистему розширень, які додають додаткові функції до фреймворку. Ці розширення охоплюють різні сфери, такі як інтеграція бази даних, обробка форм, авторизація тощо;
- вбудований сервер розробки. Flask надає вбудований сервер розробки, який полегшує запуск і тестування вашої програми під час розробки;

– Flask CLI. Flask містить інтерфейс командного рядка, який надає корисні команди для таких завдань, як запуск сервера розробки, створення міграцій бази даних і виконання тестів [20].

2.1.3 C#

C# – об'єктно-орієнтована мова програмування, розроблена компанією Microsoft. Це проста і потужна мова, яка дозволяє розробнику створювати різноманітні програми, які працюють на фреймворках .NET Framework і .NET Core. Його можна використовувати для створення різних типів додатків за допомогою Visual Studio, таких як:

- мобільні додатки;
- застосунки для настільних комп'ютерів;
- вебдодатки;
- вебсервіси;
- ігри;
- додатки до баз даних тощо [21].

C# – це проста і легка для вивчення мова, і вона буде миттєво впізнавана кожним, хто знайомий з C, C++ або Java. Ті розробники, які працювали на будь-якій з цих мов, можуть легко почати працювати на C# протягом короткого часу [21].

Синтаксис C# спрощує багато складнощів C++ і надає потужні функції, такі як нульові типи, перерахування, делегати, лямбда-вирази та прямий доступ до пам'яті [21].

Він підтримує загальні методи та типи, які дозволяють розробникам визначати власну поведінку ітерацій, які прості у використанні клієнтським кодом. Це об'єктно-орієнтована мова програмування, яка підтримує концепції інкапсуляції, успадкування та поліморфізму [21].

2.1.3.1 Фреймворк .NET Framework

Мова C# не поширюється як окремих продукт, вона є частиною платформи Microsoft .NET Framework. Вона, як правило, складається з середовища для розробки та виконання програм, написаних на C# або будь-якій іншій мові [21].

Особливості .NET Framework:

- містить мови програмування .NET, такі як C#, VB.NET, C++, F# тощо;
- має середовище для виконання Common Language Runtime (CLR), яке виконує програми на C# контрольованим чином;
- містить набір інструментів розробки, таких як компілятор, який перетворює програми C# в проміжний код, який може зрозуміти CLR;
- має набір стандартних бібліотек, таких як ADO.NET, що дозволяє отримати доступ до баз даних, таких як MS SQL Server або MySQL, і WCF, який з'єднує додатки через стандартні комунікаційні рамки та протоколи, такі як HTTP, REST, JSON, SOAP та TCP сокети [21].

2.1.3.2 Фреймворк .NET Core

.NET Core – це безплатна платформа керованого програмування з відкритим кодом для операційних систем Windows, Linux та macOS. Це кросплатформний наступник .NET Framework [21].

Особливості .NET Core:

- можна використовувати для створення різних типів додатків, таких як: мобільні, настільні, веб, хмарні, машинне навчання, мікросервіси, ігри тощо;
- написаний з нуля, щоб зробити його модульним, легким, швидким і кросплатформним фреймворком;

- включає основні функції, необхідні для запуску базових програм .NET Core;
- всі інші можливості надаються у вигляді пакетів NuGet, які ви можете додати до своєї програми за потреби.

2.2 Порівняння мов програмування та їх фреймворків. Вибір архітектури системи для розробки серверної частини вебзастосунку

Додамо описану вище інформацію про особливості мов програмування JavaScript, Python, C# до таблиці 2.1.

Таблиця 2.1 – Порівняльна характеристика мов програмування для вебсерверу

Критерії порівняння	JavaScript	Python	C#
1	2	3	4
Складність синтаксису	Середня	Простий	Складна
Швидкість виконання	Помірна	Повільна	Висока
Екосистема та бібліотеки	Широкий спектр фреймворків	Широкий спектр фреймворків	Пов'язана з платформою .NET
Система типів	Слабо типізована	Сильна типізація	Сильна типізація
ООП	Відсутні класи, проте має прототипи і функції-конструктори	Повноцінне ООП	Повноцінне ООП

Кінець таблиці 2.1

1	2	3	4
Багатопоточність	Однопотокова модель виконання, однак підтримує асинхронне програмування	Вбудована підтримка багатопоточності	Вбудована підтримка багатопоточності
Підтримка платформ	Більшість вебплатформ	Різні операційні системи	Windows, інші операційні системи з .NET Core
Спільнота розробників	Одна з найбільших у світі	Дуже активна та залучена	Значний розмір і активність
Масштабованість і розширюваність	Добре масштабується для вебпроектів будь-яких розмірів	Високий рівень	Потужна і розширювана екосистема
Безпека	Відкритий доступ до клієнтського коду	Має вбудовані заходи безпеки	Потужні механізми безпеки з .NET

Порівнявши ці мови в таблиці 2.1 можна зробити такі висновки:

- всі мови мають велику кількість фреймворків, спільноту розробників та документацію;
- підтримують усі платформи;

- JS поступається у безпеці аніж Python та C#;
- усі мають можливість асинхронного програмування;
- найкраща швидкість виконання у C#, проте і складність написання велика. Python має просту складність синтаксису, проте і найповільніший. А JS знаходиться посередині інших мов.

Далі у таблиці 2.2 порівняю особливості розглянутих фреймворків.

Таблиця 2.2 – Порівняльна характеристика фреймворків

Характеристика	Фреймворк	Опис
1	2	3
Мова програмування	ExpressJS	JS
	NestJS	JS
	Jango	Python
	Flask	Python
	.NET Framework	C#
	.NET Core	C#
Функціональність	ExpressJS	– легкість використання та навчання; – простота створення вебсерверів та API; – гнучкість та можливість налаштування.
	NestJS	– повна підтримка TypeScript та структурована архітектура; – модульність та повторне використання коду; – вбудована підтримка мікросервісної архітектури.

Продовження таблиці 2.2

1	2	3
Функціональність	Jango	– повна функціональність для швидкої розробки вебдодатків; – вбудована адміністративна панель для керування даними; – вбудована підтримка ORM та баз даних.
	Flask	– мінімалістичний фреймворк з простотою використання; – підтримка шаблонів та маршрутизації; – відмінний вибір для невеликих та середніх проєктів.
	.NET Framework	– повна платформа для розробки вебдодатків та десктопних програм; – підтримка мов програмування C# та Visual Basic; – вбудована підтримка Windows-серверів та інших Microsoft-технологій.
	.NET Core	– перехідна платформа для розробки хмарних та кросплатформених додатків; – підтримка мов програмування C# та F#; – легковага та швидка робота; – підтримка контейнеризації та мікросервісної архітектури.
Спільнота та підтримка	ExpressJS	– велика та активна спільнота розробників; – багато документації, статей та підручників; – багато розширень та пакетів розроблених спільнотою; – активне оновлення та підтримка фреймворку.

Продовження таблиці 2.2

1	2	3
Спільнота та підтримка	NestJS	– швидко зростаюча спільнота розробників; – постійно оновлювана документація та приклади коду; – активний репозиторій на GitHub з численними внесками від спільноти; – засновники та команда проєкту активно спілкуються зі спільнотою.
	Jango	– велика та досвідчена спільнота розробників; – офіційна документація високої якості та чіткість; – регулярні випуски оновлень та підтримка від розробників Django.
	Flask	– активна та зростаюча спільнота розробників; – багато розширень та пакетів розроблених спільнотою; – форуми, групи в соціальних мережах та блоги, де можна знайти допомогу та поради; – регулярні оновлення та підтримка від авторів фреймворку.
	.NET Framework	– велика спільнота розробників, особливо в середовищі Microsoft; – офіційна документація високої якості та численні ресурси для навчання; – активні форуми та спільноти, де можна отримати підтримку та поради; – майкрософт забезпечує підтримку та оновлення фреймворку.

Продовження таблиці 2.2

1	2	3
Спільнота та підтримка	.NET Core	– швидко зростаюча спільнота розробників, зокрема після випуску .NET Core; – офіційна документація високої якості та багато практичних прикладів; – активність на GitHub та форумах;
Швидкість та продуктивність	ExpressJS	Висока продуктивність, особливо при роботі зі статичними файлами та маленькими проєктами
	NestJS	Базується на Node.js та TypeScript, що дозволяє використовувати потужності JavaScript двигуна V8. Швидкодія залежить від ефективного використання асинхронного програмування та оптимізації коду
	Jango	Використовує Python, що може бути дещо повільнішим за інші мови програмування
	Flask	Також використовує Python, тому має подібну швидкодію до Django. Однак, завдяки своїй легковагості та простоті, Flask може мати більшу швидкодію при роботі з меншими проєктами
	.NET Framework	Має добре оптимізований JIT-компілятор, що сприяє високій швидкодії виконання програм
	.NET Core	Використовує той самий JIT-компілятор, що й .NET Framework, тому швидкодія може бути схожою. Проте .NET Core пропонує оптимізації та покращення продуктивності, особливо при роботі з вебзастосунками
Безпека	ExpressJS	Є мінімалістичним фреймворком, який не надає вбудованих захисних механізмів

Кінець таблиці 2.2

1	2	3
Безпека	NestJS	Забезпечує вбудовану підтримку для багатьох важливих аспектів безпеки, таких як обробка форм, аутентифікація, авторизація та захист від загроз безпеки
	Jango	Має вбудовані інструменти для захисту сесій, авторизації, контролю доступу та безпеки форм
	Flask	Для забезпечення безпеки можна використовувати різні пакети, такі як Flask-WTF для захисту форм, Flask-Security для авторизації та захисту сесій
	.NET Framework	Широкий спектр інструментів безпеки, включаючи вбудовану підтримку автентифікації та авторизації, контроль доступу, шифрування даних та захист від загроз
	.NET Core	Пропонує покращену безпеку та властивості, зокрема захист від кроссайтових скриптів та SQL Injection

Порівнявши фреймворки в таблиці 2.2 можна зробити наступні висновки:

- ExpressJS та Flask мають мінімалістичне забезпечення, що робить їх дуже простими в освоєнні. Інші фреймворки мають повну підтримку для розробки вебзастосунків;

- усі фреймворки займають перше або друге місце серед популярних у відповідних мовах, тому логічно, що всі вони мають спільноту та підтримку на дуже високому рівні;

- що стосується швидкодії, то це напряду залежить від мови програмування. Так JS фреймворки залежить від якості написаного асинхронного коду. Python фреймворки дещо програють в швидкості через

особливість мови програмування. А С# фреймворки мають високу швидкість роботи завдяки JIT-компілятору;

– ExpressJS та Flask мають мінімальний набір захисту. NestJS та Jango мають багато вбудованих систем захисту. А .NET фреймворки мають найкращі набори захисту.

Отже, взявши за увагу те, що під час навчання в університеті вже була можливість працювати як з JS та Express, так із Python та Django. А з .NET не працював ніколи. Та Express має низький ступінь входу, тому було обрано стек технологій для розробки серверної частини вебзастосунку – JavaScript та ExpressJS.

2.3 Вибір мови програмування для розробки алгоритмів

Для написання алгоритмів однозначно повинна бути розглянута мова – Python, через свою актуальність в науковій сфері та великій кількості бібліотек. Оскільки ця мова та її особливості були розглянуті в підрозділі 2.1.2, то її не буду повторно описувати. Окрім Python буде розглянута мова – C++, як низькорівнева мова програм у протилежність Python.

2.3.1 C++

C++ – це мова програмування загального призначення, яка була розроблена як вдосконалення мови С для включення об'єктно-орієнтованої парадигми. Це імперативна і скомпільована мова. C++ має ряд особливостей, включаючи:

– об'єктно-орієнтоване програмування (ООП): C++ є об'єктно-орієнтованою мовою програмування, на відміну від С, яка є процедурною мовою програмування. Це найважливіша особливість C++. Він може створювати/знищувати об'єкти під час програмування. Крім того, він може створювати креслення, за допомогою яких можна створювати об'єкти.

Поняття об'єктно-орієнтованої мови програмування: інкапсуляція, поліморфізм, успадкування, абстракція;

- незалежний від машини: виконуваний файл C++ не є незалежним від платформи (скомпільовані програми на Linux не працюватимуть на Windows), однак вони не залежать від машини;

- простий: це проста мова в тому сенсі, що програми можуть бути розбиті на логічні блоки й частини, має багату підтримку бібліотек і має різноманітні типи даних. Крім того, автоматичне ключове слово C++ полегшує життя;

- автоматичне ключове слово: ідея автоматичного ключового слова полягала в тому, щоб сформувати компілятор C++ для виведення типу даних під час компіляції, а не змушувати вас оголошувати тип даних кожен страшний час. Майте на увазі, що ви не можете заявити про щось без ініціалізатора. Повинен бути якийсь спосіб, щоб компілятор вивів ваш тип;

- мова високого рівня: C++ є мовою високого рівня, на відміну від C, яка є мовою програмування середнього рівня. Це полегшує життя в C++, оскільки це мова високого рівня, вона тісно пов'язана зі зрозумілою людині англійською мовою;

- популярна: C++ може бути базовою мовою для багатьох інших мов програмування, яка підтримує функцію об'єктно-орієнтованого програмування. Б'ярне Страуструп виявив, що Simula 67, перша об'єктно-орієнтована мова в історії, не має симуляцій, і вирішив розробити C++;

- чутлива до регістру: зрозуміло, що C++ є чутливою до регістру мовою програмування. Наприклад, `cin` використовується для прийому вхідних даних з вхідного потоку. Але якщо «`Cin`» не спрацює. Інші мови, такі як HTML і MySQL, не є регістровими мовами;

- на основі компілятора: C++ є компіляторною мовою, на відміну від Python. Тобто програми на C++ раніше компілювалися, а для їх запуску використовувався виконуваний файл. C++ є відносно швидшою мовою, ніж Java та Python;

- динамічний розподіл пам'яті: коли програма виконується в C++, змінним виділяється динамічний простір купи. Усередині функцій змінні виділяються в стековому просторі. Багато разів ми не знаємо заздалегідь, скільки пам'яті потрібно для зберігання певної інформації у визначеній змінній, і розмір необхідної пам'яті може бути визначений під час виконання;
- управління пам'яттю: C++ дозволяє виділити пам'ять змінної або масиву за часом виконання. Це відоме як динамічне виділення пам'яті. В інших мовах програмування, таких як Java і Python, компілятор автоматично управляє пам'яттю, виділеною для змінних. Але це не так у C++. У C++ пам'ять повинна бути розподілена динамічно розподілена пам'ять вручну після того, як вона не буде корисною;
- багатопоточність – це спеціалізована форма багатозадачності, а багатозадачність – це функція, яка дозволяє вашій системі виконувати дві або більше програм одночасно. Загалом, існує два види багатозадачності: процесна та потокова. Багатозадачність на основі процесів обробляє одночасне виконання програм. Багатозадачність на основі потоків має справу з мультипрограмуванням частин еквівалентної програми. Багатопотокова програма містить дві або більше частин, які будуть працювати одночасно. Кожна частина такої програми називається потоком, і кожен потік визначає окремий шлях виконання. C++ не містить вбудованої підтримки багатопотокових програм. Натомість він повністю покладається на ОС для надання цієї функції [22].

2.4 Порівняння мов програмування. Вибір архітектури системи для розробки алгоритмів

Для розробки алгоритмів буде розглянуто та порівняно тільки дві мови програмування – Python та C++. Основні особливості цих мов представлено в таблиці 2.3.

Таблиця 2.3 – Порівняльна характеристика мов програмування для алгоритмів

Критерії порівняння	Python	C++
1	2	3
Складність синтаксису	Простий	Складна
Швидкість виконання	Повільна	Висока
Екосистема та бібліотеки	Широкий спектр фреймворків	Широкий спектр бібліотек
Система типів	Сильна типізація	Сильна типізація
ООП	Повноцінне ООП	Повноцінне ООП
Багатопоточність	Вбудована підтримка багатопоточності	Вбудована підтримка багатопоточності
Підтримка платформ	Різні операційні системи	Різні операційні системи
Спільнота розробників	Дуже активна та залучена	Дуже активна і велика кількість ресурсів
Масштабованість і розширюваність	Високий рівень	Високий рівень
Безпека	Має вбудовані заходи безпеки	Не має вбудованих заходів безпеки

Порівнявши мови Python та C++ в таблиці 2.3 можна зробити такі висновки:

– обидві мови підтримують парадигму ООП, строгу типізацію даних, багатопоточність, різні платформи, велику кількість спільноти, бібліотек та масштабованість;

- різниця в синтаксису: у Python він простий та повільна робота, у C++ навпаки синтаксис складний та висока швидкість роботи;
- C++ надає більше можливостей користувачу щодо керування системною пам'яттю;
- Python зараз – актуальна мова для наукових досліджень, через це має велику кількість високорівневих бібліотек.

Отже, відносно прості алгоритми (генетичний, табу пошук, симуляція відпалу) для генерації розкладу будуть написані на швидкій мові C++ з використанням багатопоточності. А складний алгоритм нейрона мережа вже буде написаний на Python з використанням відповідних бібліотек.

2.5 Вибір та порівняння архітектури та шаблонів проєктування вебзастосунку

Спочатку розглянемо можливі архітектури вебзастосунку.

Клієнтські компоненти: фронтенд-частина, яка з'являється в браузері і служить полем для взаємодії з вебдодатком. Клієнтські компоненти розробляються за допомогою HTML, CSS і Javascript і не потребують підключення операційної системи або пристрою для роботи. Простіше кажучи, ці компоненти є представленням вебдодатків [23].

Серверні компоненти: серверна частина архітектури вебдодатків, побудована з використанням Java, Python, .Net, NodeJS, PHP та Ruby on Rails для створення логіки програми. Також серверні компоненти складають базу даних, маючи на увазі центр управління для зберігання даних [23].

На рис. 3.1 відображена 3-рівнева архітектура.

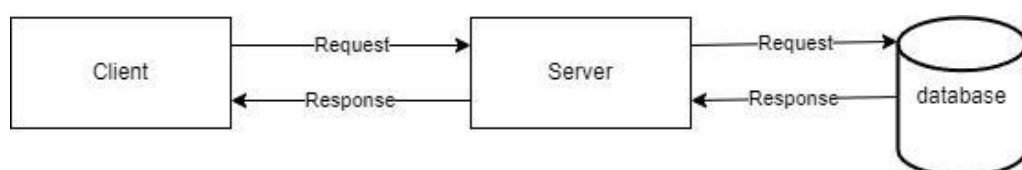


Рисунок 2.1 – Схема 3-рівневої архітектури

Більшість вебдодатків створюються шляхом поділу їх основної функції на шари/яруси. Відповідно, це дозволяє швидко і без зусиль замінювати або оновлювати ці шари самостійно. Її називають багато- або 3-рівневою архітектурою [23].

У 3-рівневій архітектурі виділяють три шари/яруси:

- презентаційний (клієнтський) шар;
- прикладний (бізнес) шар;
- рівень доступу до даних [23].

Можна сказати, що ця архітектура є найбезпечнішою. Це можна пояснити тим, що клієнт не отримує доступ до даних безпосередньо. Сервери додатків можуть бути розгорнуті на декількох постачальниках машин, що забезпечує більш високу масштабованість, підвищену продуктивність і кращу ефективність [23].

Кожен ярус можна масштабувати самостійно, отже, таку архітектуру можна масштабувати по горизонталі. Крім того, це також значно покращує загальну цілісність даних, оскільки дані будуть проходити через сервер додатків, який саме вирішує, як і ким будуть доступні дані. Ця дивовижна перевага є простим та економічно ефективним рішенням у разі нового керівництва компанії [23].

На рис. 2.2 відображено Single Page Application (SPA) архітектуру.

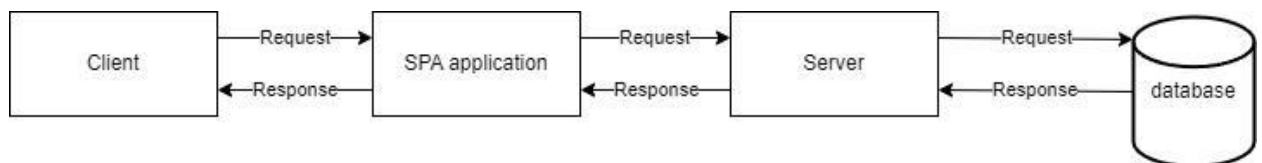


Рисунок 2.2 – Схема SPA архітектури

У цій схемі архітектури вебзастосунку SPA клієнт включає в себе SPA-додаток, який виконується безпосередньо у браузері. SPA-додаток завантажується одноразово при першому завантаженні сторінки. Він взаємодіє з сервером через HTTP запити, надсилавні асинхронно з

використанням AJAX або Fetch API. Клієнтський код, написаний на JavaScript, виконується безпосередньо в браузері та контролює відображення та поведінку сторінки [23].

Клієнт (браузер) отримує від сервера лише необхідні дані (наприклад, JSON або XML) і використовує їх для відображення контенту на сторінці. Логіка обробки запитів та оновлення сторінки відбувається на клієнтській стороні, без повного перезавантаження сторінки [23].

Сервер відповідає на HTTP запити від клієнта, обробляє їх та надсилає необхідні дані назад до клієнта. Сервер може виконувати різні функції, такі як обробка бізнес-логіки, взаємодія з базою даних або надання статичних ресурсів [23].

Ця архітектура дозволяє створювати динамічні та інтерактивні вебзастосунки, які забезпечують плавну та швидку взаємодію з користувачем без перезавантаження сторінки [23].

На рис. 2.3 відображено схему мікросервісної архітектури.

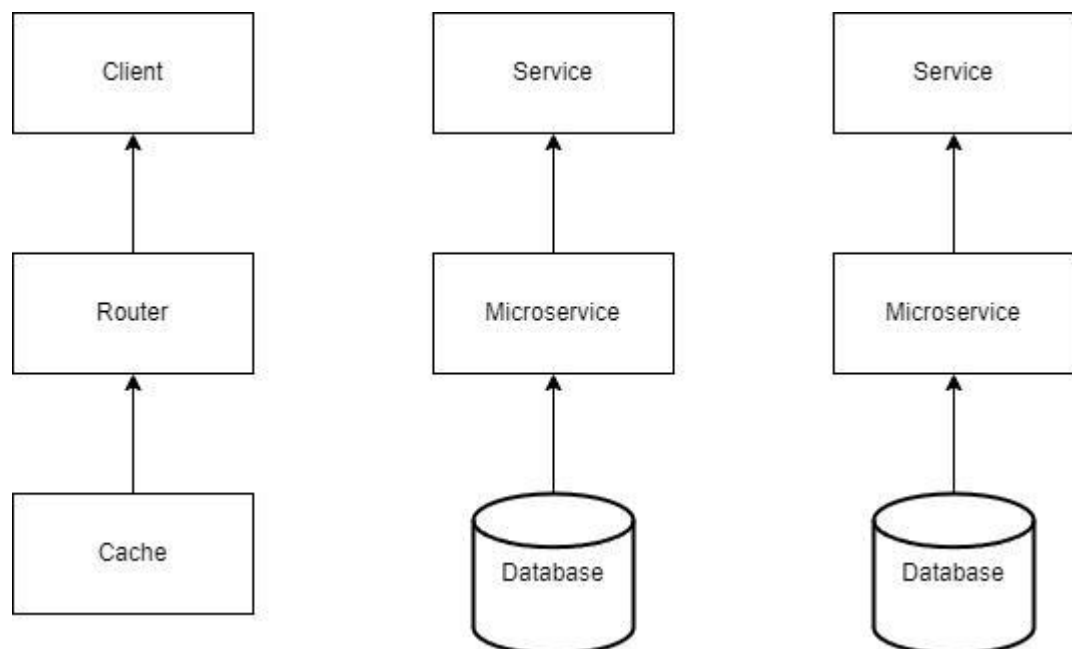


Рисунок 2.3 – Схема мікросервісної архітектури

Будучи частою альтернативою ненадійній монолітній архітектурі вебдодатків, мікросервіси розосереджують функціональність для надання невеликих і легких послуг окремо [23].

Зокрема, вони слабо пов'язані між собою і використовують Application Programming Interface (API) для спілкування, якщо виникає складна бізнес-проблема. Ця особливість полегшує життя розробнику, оскільки окремі компоненти сервісу можуть бути побудовані на різних мовах програмування [23].

Завдяки своїй гнучкості та стабільності архітектура мікросервісів набула популярності в наші дні, і такі компанії, як Amazon, eBay та Netflix, використовують її для своїх складних потреб [23].

Плюси мікросервісної архітектури:

- простіше масштабування;
- краща відмовостійкість;
- проста для розуміння кодова база;
- незалежне розгортання модулів [23].

Мінуси мікросервісної архітектури:

- труднощі з тестуванням і налагодженням;
- розгортання комплексу [23].

На рис. 2.4 відображено схему Progressive Web Application (PWA) архітектури.

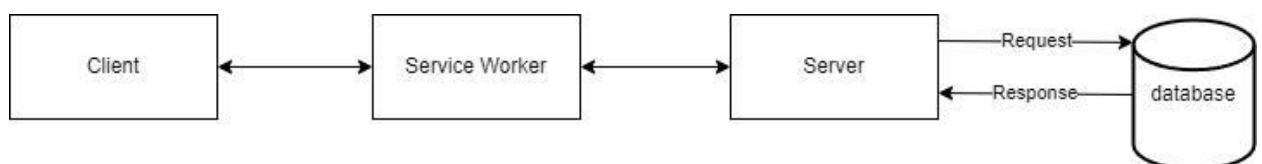


Рисунок 2.4 – Схема PWA архітектури

У схемі архітектури вебзастосунку PWA клієнт здатний працювати в офлайн режимі та надає користувачам досвід, схожий на нативні додатки. Клієнт включає в себе JavaScript-додаток, який виконується в браузері. Він

може зберігати дані локально за допомогою локального сховища (LocalStorage, IndexedDB) та використовувати Service Worker – проксі-сервер, який керує кешуванням ресурсів та забезпечує офлайн режим роботи додатку [23].

Сервер виконує функції, пов'язані з обробкою запитів від клієнта та доступом до бази даних. Взаємодія між клієнтом і сервером здійснюється за допомогою протоколу, такого як HTTP або WebSocket [23].

У схемі архітектури вебзастосунку REST (Representational State Transfer) клієнт і сервер взаємодіють за допомогою стандартних HTTP-запитів [23].

А схема виглядає так само, як на рис. 2.1.

Клієнт використовує HTTP-методи, такі як GET, POST, PUT, DELETE, для взаємодії з сервером. Клієнт формулює запити до сервера, включаючи URL-шлях до ресурсу та необхідні параметри. Сервер обробляє ці запити, виконує необхідні операції і повертає відповідь клієнту, яка може бути у форматі JSON, XML або іншому [23].

Сервер відповідає на запити клієнта, оброблює їх і взаємодіє з базою даних, яка зберігає дані. Сервер також може реалізовувати бізнес-логіку, обробляти авторизацію та аутентифікацію користувачів, ініціювати операції збереження та оновлення даних тощо [23].

На рис. 2.5 відображено схему GraphQL архітектури.

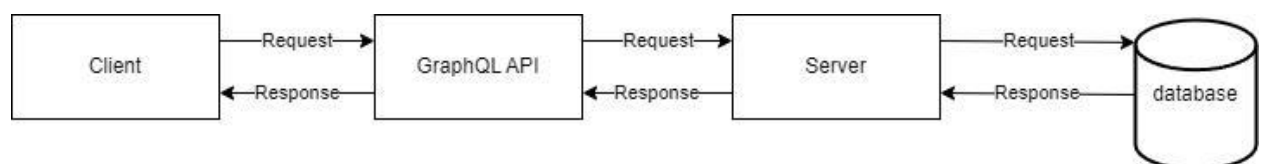


Рисунок 2.5 – Схема GraphQL архітектури

У схемі архітектури вебзастосунку GraphQL клієнт і сервер взаємодіють за допомогою запитів та відповідей у форматі GraphQL [23].

Клієнт відправляє запити до сервера, що містять структурований запит GraphQL. Запити можуть включати поля, фільтри, змінні та інші параметри. Клієнт описує, які дані йому потрібні і як їх організувати, і сервер надає відповідь, яка точно відповідає цим вимогам [23].

Сервер приймає запити GraphQL, обробляє їх та виконує необхідні операції, щоб зберегти, отримати або оновити дані. Він взаємодіє з базою даних або іншими джерелами даних для отримання потрібної інформації. Сервер виконує розв'язувач GraphQL, який аналізує запити, вирішує, які дані необхідно відправити клієнту, і забезпечує ці дані відповідно до специфікації GraphQL [23].

Отже, було розглянуто такі схема архітектури вебзастосунку:

- 3-рівна;
- SPA;
- мікросервісна;
- PWA;
- REST;
- GraphQL.

Вебзастосунки можуть використовувати комбінацію різних архітектур та технологій в залежності від їхніх вимог та потреб [23].

Для розробки вебзастосунку було обрано такі архітектури: 3-рівнева, SPA, GraphQL. Загальна схема обраних архітектур відображена на рис. 2.6.

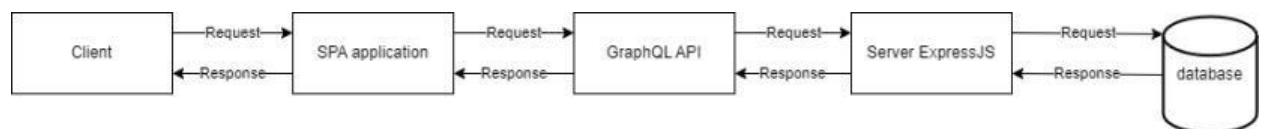


Рисунок 2.6 – Загальна схема архітектур вебзастосунку

Шаблон проєктування – це формалізована найкраща практика, яку можна використовувати для вирішення типових проблем під час проєктування програми або системи. Терміни, з яких вони складаються:

- Model: зберігає дані та безпосередньо спілкується з базою даних. Модель – це частина, яка представляє ваші дані та логіку програми. Вона визначає бізнес-правила, які керують обробкою, модифікацією даних;
- View: відображає дані моделі та відповідає за представлення даних в інтерфейсі користувача;
- ViewModel: є ексклюзивним для шаблону MVVM. Це абстракція шару перегляду, а також діє як обгортка для даної моделі;
- Controller: компонент, який інтегрує вид і модель;
- Presenter: компонент, який існує тільки в моделі MVP. Доповідач отримує вхідні дані від компонента view і обробляє дані за допомогою моделі [24].

Шаблон Model-View-Controller (MVC).

Архітектурний шаблон MVC був першим, і він популярний сьогодні в області вебдодатків. Він був представлений в 1970-х роках. Цей шаблон дозволяє створити додаток відокремлення інтересів (Separation of Concerns). Це полегшує зусилля, необхідні для тестування, підтримки та розробки вашої програми [24].

У шаблоні MVC model не має розуміння view або controller. Спостерігач моделі отримуватиме сповіщення щоразу, коли змінюється view і controller. Controller допомагає процесу маршрутизації підключити model до відповідного view [24].

Деякі з переваг шаблону MVC:

- поділ проблем (більш цілеспрямовано);
- полегшує тестування коду та керування ним;
- сприяє роз'єднанню шарів програми;
- краща організація коду та можливість повторного використання [24].

Мінуси:

- при неадекватній архітектурі може призвести до складності в управлінні потоком даних між компонентами;

– можлива зайва складність при малих та простих проєктах [24].

На рис. 2.7 відображено схему роботи шаблону MVC.

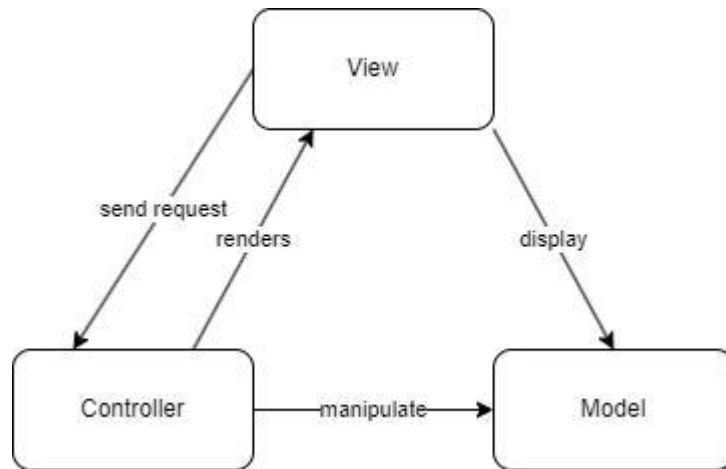


Рисунок 2.7 – Схема шаблону MVC

Шаблон Model-View-Presenter (MVP).

Шаблон MVP має два компоненти з MVC: model і view. Він замінює controller на presenter. Presenter – як випливає з його назви – використовується для того, щоб представити щось. Це дозволяє легше знущатися над view [24].

У MVP presenter має функціонал «посередника», тому що вся логіка презентації підштовхується до нього. View і presenter в MVP також незалежні один від одного і взаємодіють через інтерфейс [24].

Плюси MVP:

- легше тестування, оскільки presenter можна тестувати незалежно від інтерфейсу користувача;
- зменшує залежність між компонентами;
- дозволяє більшу контрольованість над взаємодією між компонентами.

Мінуси:

- вимагає додаткових зусиль для розробки та підтримки коду;
- може збільшити кількість класів та складність архітектури.

На рис. 2.8 відображено схему роботи шаблону MVP.

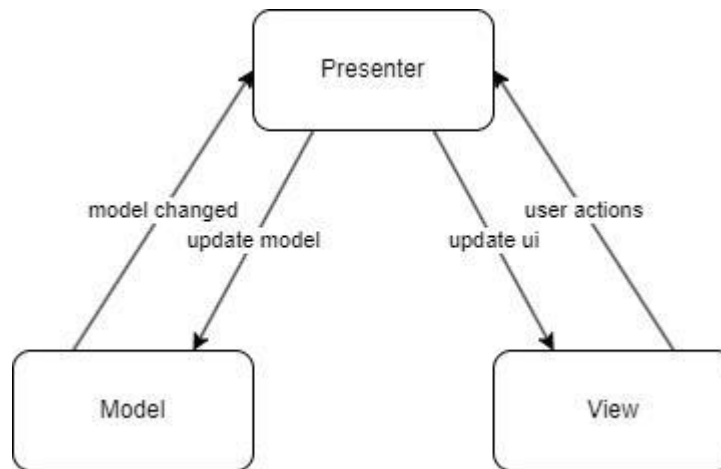


Рисунок 2.8 – Схема шаблону MVP

Presenter отримує вхідні дані від користувача через view. Потім він обробляє дії користувача за допомогою model, передаючи результати назад в view. Presenter спілкується з view через інтерфейси [24].

Шаблон Model-View-ViewModel (MVVM).

MVVM – це сучасна еволюція MVC. Основна мета MVVM – забезпечити чіткий поділ між логікою домену і рівнем представлення. MVVM підтримує двосторонню прив'язку даних між view та viewmodel [24].

Шаблон MVVM дозволяє відокремити code's view та model. Це означає, що при зміні моделі view робити не потрібно, і навпаки. Використовуючи viewmodel, ви можете проводити модульне тестування та перевіряти свою логічну поведінку без залучення вашого view [24].

Плюси MVVM:

- простота використання та вивчення, особливо для розробників, знайомих з Django;

- розділення відповідальностей між компонентами, полегшує розробку та підтримку коду.

Мінуси:

- вимагає дотримання специфічних правил та конвенцій фреймворку Django;
- може бути обмежена у плані гнучкості для деяких складних сценаріїв.

На рис. 2.9 відображено схему роботи шаблону MVVM.

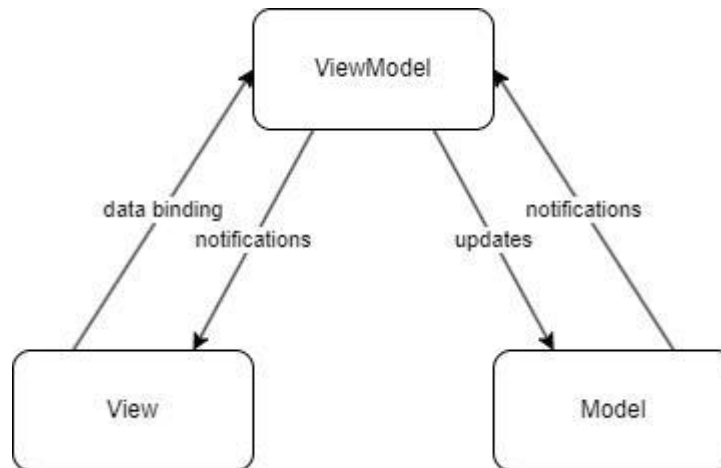


Рисунок 2.9 – Схема шаблону MVVM

Коли використовувати MVC: MVC – це просто реалізація поділу проблем. Якщо вашому додатку потрібно розділити дані (Model), обробку даних (Controller) і представлення даних (View), MVC буде добре працювати. MVC також добре працює в додатку, де джерело даних та / або представлення даних може змінитися в будь-який час [24].

Коли використовувати MVP: можна використовувати MVP, коли ваша програма має двонаправлений потік. Якщо взаємодії з користувачем потрібно щось запитати у моделі, і результат цього запиту негайно змінить інтерфейс користувача, розгляньте MVP [24].

Коли використовувати MVVM:

- потрібно поділитися проектом з дизайнером і проектно-конструкторські роботи можуть відбуватися самостійно;
- потрібно модульне тестування для ваших рішень;

- потрібно мати компоненти для повторного використання як усередині, так і в межах проєктів в організації;
- потрібна більша гнучкість, щоб змінювати свої views без необхідності рефакторінгу іншої логіки в кодовій базі [24].

При розробці вебзастосунку було прийнято рішення використовувати шаблон проєктування MVP. Оскільки він дозволяє зробити більшу контрольованість над взаємодією між компонентами.

2.6 Вибір бази даних

Реляційні бази даних (БД): пари ключ-значення використовуються для зберігання структурованих даних у таблицях, що складаються з стовпців і рядків. Реляційні бази даних, які включають MySQL, PostgreSQL та багато інших, є найбільш часто використовуваними базами даних з відкритим вихідним кодом. Їх часто віддають перевагу для зберігання та обробки даних бізнес-аналітики організаціями, яким потрібні швидкі SQL-запити [25].

Нереляційні бази даних: замість таблиць нереляційні (NoSQL) бази даних використовують сховище даних на основі документів, стовпчасто-орієнтоване сховище та графові бази даних. Нереляційні бази даних, до яких відносяться MongoDB, Apache Cassandra та інші, віддають перевагу для зберігання та обробки неструктурованих даних. Наприклад, аналітична програма найкраще працюватиме з неструктурованими файлами зображень, що зберігаються в нереляційній графічній базі даних [25].

2.5.1 MySQL

MySQL – частина SQL розшифровується як Structured Query Language – є однією з найпопулярніших реляційних баз даних з відкритим вихідним кодом. Він широко використовується для вебдодатків, сховищ даних та

обробки онлайн транзакцій. Популярність MySQL пояснюється багатьма факторами:

- це надійна, швидка та надійна система;
- MySQL не має крутої кривої навчання;
- він сумісний майже з кожною операційною системою (ОС), яку використовуватиме DBA або розробник; а середовище MySQL сприяє масштабованості [25].

MySQL – це система управління реляційними базами даних з типовими функціями СУБД, включаючи таблиці, представлення, зовнішні ключі та збережені процедури. Він добре підходить для більшості робочих навантажень з обробки онлайн-транзакцій і працює з деякими робочими навантаженнями онлайн-аналітичної обробки. MySQL особливо ефективний зі стандартними реляційними схемами та вебробочими навантаженнями. Проста асинхронна реплікація дозволяє легко масштабувати читання та розвантажувати запити звітів. Синхронна реплікація підтримує високу доступність. MySQL зазвичай використовується компаніями, які розміщують вебтранзакції – тобто банківські операції, покупки, замовлення та реєстрація. MySQL посідає друге місце за популярністю системи керування базами даних (DB-Engines, березень 2023) [25].

2.5.2 PostgreSQL

PostgreSQL – це нереляційна система баз даних, швидко завоювала популярність серед професійних розробників. Статистика StackOverflow показує, що 26% розробників віддали перевагу йому в 2017 році, 34% в 2019 році і 40% в 2021 році. Зовсім недавно, в опитуванні розробників стеків StackOverflow 2022 року, PostgreSQL трохи випередив MySQL (46,48% до 45,68%) як найпопулярнішу платформу баз даних серед професійних розробників [25].

PostgreSQL віддають перевагу за його складний аналіз даних, науку про дані, графіки та можливості, пов'язані з штучним інтелектом. PostgreSQL відомий потужними та розширеними функціями, включаючи асинхронну реплікацію, повнотекстовий пошук у базі даних та вбудовану підтримку сховища в стилі JSON, зберігання ключів-значень та XML. PostgreSQL також дуже розширюваний, що дозволяє користувачам додавати власні функціональні можливості за допомогою плагінів і розширень [25].

PostgreSQL – це об'єктно-реляційна система управління базами даних з усіма стандартними функціями СУБД плюс підтримка складних об'єктів, успадкування таблиць і додаткових типів даних за межами JSON. Ці доповнення дозволяють PostgreSQL підтримувати складніші робочі навантаження та конструкції схем. PostgreSQL має подібну реплікацію до MySQL для побудови архітектур. Будучи більш просунутою системою управління базами даних, PostgreSQL добре підходить для швидкого виконання складних запитів у великому середовищі. Оскільки він легко підтримує відмовостійкість та повне резервування, фінансові установи та виробники часто віддають йому перевагу. Він також є кращим для використання з геоінформаційними системами та геопросторовими даними. PostgreSQL займає четверте місце за популярністю системи керування базами даних (DB-Engines, березень 2023) [25].

2.5.3 MongoDB

MongoDB – популярна доступна нереляційна (NoSQL) база даних. Вона зберігає дані в документах і колекціях, дизайн ідеально підходить для обробки великих обсягів неструктурованих даних. Відомий своєю продуктивністю та масштабованістю, вона часто використовується для великих обсягів даних та вебдодатків у режимі реального часу [25].

У той час як багато пропозицій програмного забезпечення з відкритим вихідним кодом, такі як версія спільноти MySQL, використовують ліцензію

GPL, MongoDB була під ліцензією AGPL і останнім часом під ліцензією SSPL (представлена самою MongoDB). Багато прихильників відкритого вихідного коду, включаючи Open Source Initiative, не вважають програмне забезпечення в рамках SSPL відкритим вихідним кодом [25].

MongoDB – це нереляційна (NoSQL) програма баз даних, доступна для джерел, яка зберігає дані в JSON-подібних документах з додатковими схемами. Вона дуже масштабована і ідеально підходить для аналітики в режимі реального часу та високошвидкісного ведення журналу. MongoDB є кращою системою для аналізу даних, оскільки документи легко обмінюються кількома вузлами, а також завдяки своїм можливостям індексації, запиту на вимогу та агрегації в режимі реального часу. Набори реплік MongoDB дозволяють надмірність даних та автоматичну відмову, встановлюючи основу для високої доступності. MongoDB також забезпечує надійне шифрування та безпеку брандмауера. MongoDB вимагає значного місця для зберігання. MongoDB краще підходить для роботи з системами управління контентом, мобільними додатками і аналітикою в режимі реального часу [25].

MongoDB популярна в різних галузях – опитування 90 240 компаній, що використовують MongoDB, показало, що провідні способи використання технологій та послуг (23%), комп'ютерного програмного забезпечення (16%) та Інтернету (6%). MongoDB займає п'яте місце за популярністю системи управління базами даних і є No1 серед нереляційних систем (DB-Engines, березень 2023) [25].

2.5.3 Порівняння та аналіз баз даних

Зведемо основні параметри баз даних до таблиці 2.4.

Таблиця 2.4 – Порівняльна характеристика баз даних

Критерії порівняння	MySQL	PostgreSQL	MongoDB
1	2	3	4
Тип	Реляційна	Об'єктно-реляційна	Документоорієнтовна
Мова запитів	SQL	SQL, PL/pgSQL	MongoDB (JSON подібна)
Масштабованість	Вертикальне та горизонтальне масштабування	Вертикальне та горизонтальне масштабування, включаючи реплікацію та розподілені бази даних	Горизонтальне масштабування
Підтримка операцій	Широкий набір операцій для створення, модифікації, запитів і адміністрування бази даних	Надає розширені можливості для роботи з базою даних, включаючи транзакції, засіб повернення до попереднього стану (rollback), точки відновлення (savepoints) та багато інших	Широкий набір операцій для роботи з документами, включаючи створення, оновлення, видалення та запити, а також можливості для реплікації та розподілення

Кінець таблиці 2.4

1	2	3	4
Підтримка інструментів	Розширена підтримка інструментів для адміністрування бази даних, розробки, моніторингу та налагодження	Розширена підтримка інструментів для адміністрування бази даних, розробки, моніторингу та налагодження	Власний набір інструментів для роботи з базою даних, таких як MongoDB Compass, який надає графічний інтерфейс для взаємодії з даними

Порівнявши особливості баз даних з таблиці 2.4 можна зробити такі висновки:

- усі наведені бази даних дуже популярні на цей час, що дає дуже високу функціональність кожної з них;
- якщо порівнювати MySQL з PostgreSQL, то однозначно PostgreSQL має більше можливостей, таких як нові типи даних, транзакцій точки відновлення;
- усі наведені БД мають відкритий код, тобто являються безкоштовними та мають велику кількість документації.

Обрати БД з наведених можна виходячи із поставленої задачі. PostgreSQL слід використати для професійної розробки та складної схеми БД. MongoDB слід використати для програм у реальному часі. А MySQL надає базовий функціонал, якого достатнього для студентського проєкту. Тому несумніно вибір пав на MySQL.

2.7 Розробка моделі взаємодії користувача з системою

Основні терміни предметної області були наведені в заголовку 1.1. Тому наступним кроком потрібно розподілити функції (сценарії) між користувачами.

В проєкті передбачені наступні ролі користувачів та їх функції:

- а) гість – неавторизований користувач, який може переглядати, фільтрувати та завантажувати розклад по групам, викладачам, аудиторіям;
- б) адміністратор – авторизований користувач, якому доступні наступні можливості:

- 1) авторизація;
- 2) керування даними (перегляд, фільтрація, додавання, редагування, видалення) таких сутностей:
 - аудиторії;
 - кафедри;
 - спеціальності;
 - дисципліни;
 - заняття;
 - групи;
 - викладачів.
- 3) видалення всіх даних;
- 4) завантаження даних з відомостями кафедри по шаблону;
- 5) перегляд та запуск фітнес-функції поточного розкладу;
- 6) редагування параметрів розкладу та значеннями штрафів;
- 7) керування алгоритмами: запуск, редагування параметрів;
- 8) перегляд результатів роботи алгоритмів.

Для графічного опису взаємодії між системою та користувачами була розроблена діаграма прецедентів (рис. 2.10).

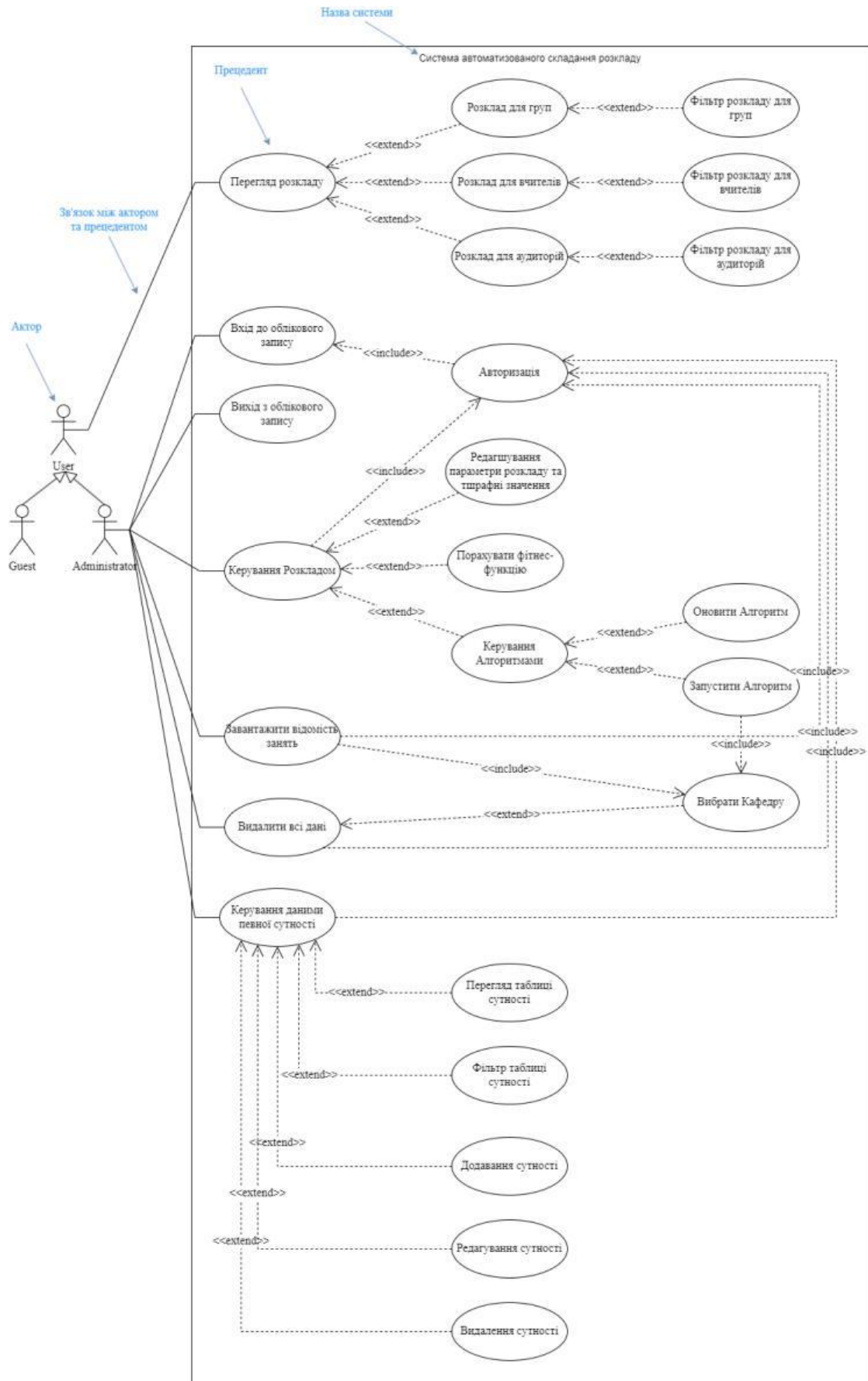


Рисунок 2.10 – Діаграма прецедентів для системи автоматизованого складання розкладу

На рис. 2.10 відображено ті самі функція, що були наведені у списку. Оскільки як гість так і адміністратор можуть переглядати розклад, та було створено додаткового актора «User». Щоб адміністратор міг керувати системою, він повинен авторизуватися. Прецедент «керування даними певної сутності» підрозуміває сутності наведені у пункті 6.2 списку ролей та їх функціях.

Отже, була створена модель, яка описує, як користувачі будуть взаємодіяти з системою для автоматичного складання розкладу. Ця модель була представлена у вигляді діаграми прецедентів, яка використовує стандартну нотацію UML.

2.8 Висновок

В цьому розділі було проведено аналіз та вибір інструментарію розробки вебзастосунку.

Було проведено порівняльний аналіз мов програмування JS, Python, C# та їх фреймворків ExpressJS, NestJS, Jango, Flask, .NET Framework, .NET Core для створення серверу та вирішено розробити вебзастосунок на мові JavaScript з фреймворком ExpressJS.

Було проведено порівняльний аналіз мов програмування Python, C++ для побудови алгоритмів та вирішено використовувати для алгоритмів (генетичний, табу пошук, симуляція відпалу) C++, а для нейронної мережі Python.

Було проведено аналіз архітектур вебзастосунку: 3-рівневий, SPA, мікросервісний, PWA, GraphQL. В результаті була обрана комбінація архітектур client-SPA-GraphQL-server(expressJS)-database.

Було проведено аналіз шаблонів проєктування вебзастосунку: MVC, MVP, MVVM. В результаті було обрано використовувати шаблон Model-View-Presenter.

Було проведено аналіз баз даних: MySQL, PostgreSQL, MongoDB. В результаті обрано використовувати БД в системі MySQL.

Створено модель взаємодії користувача з системою та розподіл функцій, яка подана у вигляді діаграми прецедентів у нотації UML.

3 ОПИС ПРОГРАМИ

3.1 Структура бази даних MySQL

Спершу потрібно описати структуру бази даних, як вона функціонує. На рис. 3.1 зображена схема БД, розроблена в програмі MySQL Workbench, інструментом Reverse Engineer.

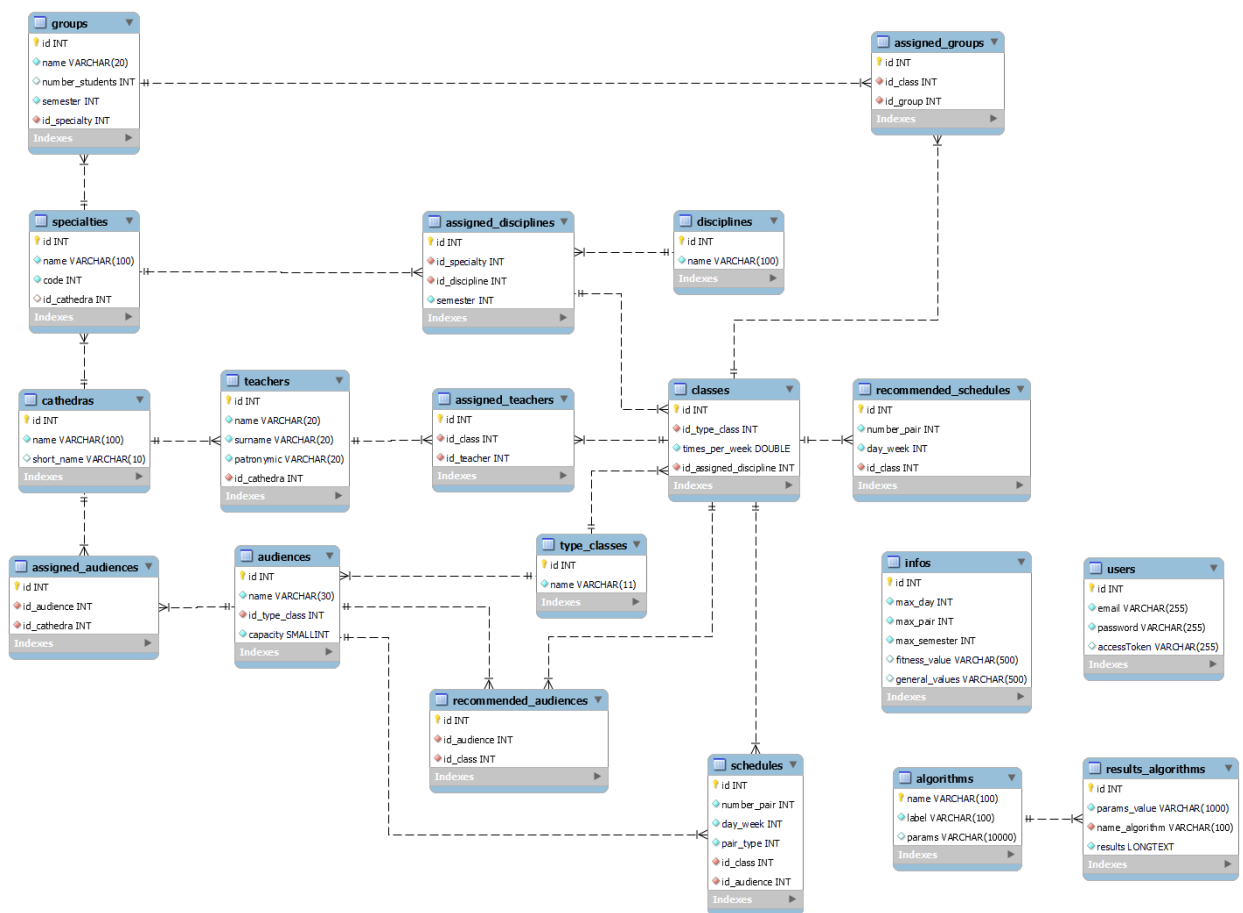


Рисунок 3.1 – Схема бази даних

Спочатку слід провести опис таблиць та визначити залежності між ними:

а) *cathedras*. Являється узагальнюючою одиницею, оскільки розклад та інші таблиці поділяється за кафедрами;

б) *audiences*. В аудиторії проводяться заняття. Мають зв'язки:

1) багато до багатьох з audiences, через це існує додаткова таблиця assigned_audiences;

2) один до багатьох з type_classes;

в) teachers. Викладачі викладають заняття. Має зв'язки:

1) один до багатьох з cathedras;

г) specialties. Також узагальнююча одиниця, яка поділяє кафедри на спеціальності. Має зв'язки:

1) один до багатьох з cathedras;

д) groups: групи, яким проводять заняття. Має зв'язки:

1) один до багатьох з specialties;

е) disciplines. Дисципліни, які закріплені за спеціальностями та викладаються. Має зв'язки:

1) багато до багатьох з specialties, оскільки можуть проводитися у різні семестри. Через це існує додаткова таблиця assigned_disciplines;

є) classes. Заняття, які проводяться у дисципліні з певними групами, викладачами та аудиторіями. Має зв'язки:

1) багато до багатьох з audiences, через це існує додаткова таблиця recommended_audiences;

2) один до багатьох з type_classes;

3) багато до багатьох з teachers, через це існує додаткова таблиця assigned_teachers;

4) один до багатьох з assigned_disciplines;

5) багато до багатьох з groups, через це існує додаткова таблиця assigned_groups;

ж) recommended_schedules. Рекомендований час проведення занять (день тижня, номер пари). Має зв'язки:

1) один до багатьох з classes;

з) schedules. Являється парою, яка проводиться у заняття. Має зв'язки:

1) один до багатьох з audiences;

2) один до багатьох з classes;

и) algorithms. Певний алгоритм зі своїми параметрами;

і) results_algorithms. Результати роботи алгоритмів. Має зв'язки:

1) один до багатьох з algorithms;

ї) users. Адміністратори системи, які мають вебтокен;

й) infos. Інформація про кількість днів, кількість пар, кількість семестрів, фітнес-значення та штрафні значення.

Слід скласти опис таблиць та їх параметрів (табл. 3.1) зі схеми на рис.

3.1.

В колонці примітки можуть бути такі значення:

- PK: primary key;
- NN: not null;
- UQ: unique;
- AI: auto increment;
- FK: foreign key.

В колонці тип даних можуть бути такі значення:

- INT: 4 байти, [-2147483648; 2147483647];
- SMALLINT: 2 байти, [-32768; 32767];
- DOUBLE: 8 байт;
- VARCHAR(size). Рядок змінної довжини (може містити літери, цифри та спеціальні символи). Параметр size визначає максимальну довжину стовпця в символах – може бути від 0 до 65535;
- LONGTEXT. Містить рядок максимальною довжиною 4 294 967 295 символів.

Таблиця 3.1 – Таблиці та їх параметри зі схеми БД

Назва таблиці	Назва параметру	Тип даних	Примітки	Опис
1	2	3	4	5
algorithms	name	VARCHAR (100)	PK, NN	Назва алгоритму
	label	VARCHAR (100)	NN	Назва алгоритму на українському
	params	VARCHAR (10000)		Параметри алгоритму у форматі JSON
assigned_audiences	id	INT	PK, NN, AI	Номер запису
	id_audience	INT	NN, FK	Номер аудиторії
	id_cathedra	INT	NN, FK	Номер кафедри
assigned_disciplines	id	INT	PK, NN, AI	Номер запису
	id_specialty	INT	NN, FK	Номер спеціальності
	id_discipline	INT	NN, FK	Номер дисципліни
	semester	INT	NN	Семестр
assigned_groups	id	INT	PK, NN, AI	Номер запису
	id_class	INT	NN, FK	Номер заняття
	id_group	INT	NN, FK	Номер групи

Продовження таблиці 3.1

1	2	3	4	5
assigned _teachers	id	INT	PK, NN, AI	Номер запису
	id_class	INT	NN, FK	Номер заняття
	id_teacher	INT	NN, FK	Номер викладача
audience s	id	INT	PK, NN, AI	Номер запису
	name	VARCHAR(30)	NN	Назва аудиторії
	id_type_class	INT	NN, FK	Номер типу заняття
	capacity	SMALLINT	NN	Місткість аудиторії
cathedras	id	INT	PK, NN, AI	Номер запису
	name	VARCHAR(100)	NN	Назва кафедри
	short_name	VARCHAR(10)		Коротка назва кафедри
classes	id	INT	PK, NN, AI	Номер запису
	id_type_class	INT	NN, FK	Номер типу заняття
	times_per_week	DOUBLE	NN	Кількість занять на тиждень
	id_assigned_discipline	INT	NN, FK	Номер закріпленої дисципліни
disciplin es	id	INT	PK, NN, AI	Номер запису
	name	VARCHAR(100)	NN	Назва дисципліни

Продовження таблиці 3.1

1	2	3	4	5
groups	id	INT	PK, NN, AI	Номер запису
	name	VARCHAR(20)	NN	Назва групи
	number_students	INT		Кількість студентів
	semester	INT	NN	Семестр
	id_specialty	INT	NN, FK	Номер спеціальності
infos	id	INT	PK, NN, AI	Номер запису
	max_day	INT	NN	Кількість днів в розкладі
	max_pair	INT	NN	Кількість пар в розкладі
	max_semester	INT	NN	Кількість семестрів
	fitness_value	VARCHAR(500)		Фітнес значення в JSON
	general_values	VARCHAR(500)		Штрафи в JSON
recommended_audiences	id	INT	PK, NN, AI	Номер запису
	id_audience	INT	NN, FK	Номер аудиторії
	id_class	INT	NN, FK	Номер заняття
recommended_schedules	id	INT	PK, NN, AI	Номер запису
	number_pair	INT	NN	Номер пари
	day_week	INT	NN	День тижня
	id_class	INT	NN, FK	Номер заняття

Продовження таблиці 3.1

1	2	3	4	5
results_algorithms	id	INT	PK, NN, AI	Номер запису
	params_value	VARCHAR(1000)	NN	Параметри запуску в JSON
	name_algorithm	VARCHAR(100)	NN, FK	Назва алгоритму
	results	LONGTEXT	NN	Результати в JSON
schedules	id	INT	PK, NN, AI	Номер запису
	number_pair	INT	NN	Номер пари
	day_week	INT	NN	День тижня
	pair_type	INT	NN	Тип пари
	id_class	INT	NN, FK	Номер заняття
	id_audience	INT	NN, FK	Номер аудиторії
specialties	id	INT	PK, NN, AI	Номер запису
	name	VARCHAR(100)	NN	Назва спеціальності
	code	INT	NN	Код спеціальності
	id_cathedra	INT	NN, FK	Номер кафедри
teachers	id	INT	PK, NN, AI	Номер запису
	name	VARCHAR(20)	NN	Ім'я
	surname	VARCHAR(20)	NN	Прізвище
	patronymic	VARCHAR(20)	NN	По батькові

Кінець таблиці 3.1

1	2	3	4	5
teachers	id_cathedra	INT	NN, FK	Номер кафедри
type_classes	id	INT	PK, NN, AI	Номер запису
	name	VARCHAR(11)	NN	Назва типу заняття
users	id	INT	PK, NN, AI	Номер запису
	email	VARCHAR(255)	NN, UQ	Почта
	password	VARCHAR(255)	NN	Пароль
	accessToken	VARCHAR(255)		Вебтокен

Отже, було розписано для чого кожна таблиця існує та особливості їх структури.

3.2 Структура вебсерверу ExpressJS

3.2.1 Підключення до БД та Models

Для підключення та роботою з БД, було обрано використовувати фреймворк Sequelize.

Sequelize – це Object-Relation Mapping (ORM) для Node.js, який дозволяє зручно взаємодіяти з реляційними базами даних, такими як MySQL, PostgreSQL, SQLite та інші.

Основні функції та особливості Sequelize:

- відображення об'єктів JavaScript на таблиці бази даних. Sequelize дозволяє описувати моделі даних за допомогою класів JavaScript, які автоматично відображаються на таблиці в базі даних;

- підтримка реляційних зв'язків. Sequelize надає можливість встановлювати різні типи реляційних зв'язків між моделями даних, такі як один до одного, один до багатьох та багато до багатьох;
- підтримка міграцій. Sequelize надає інструменти для створення та управління міграціями бази даних, що дозволяє контролювати версії схеми бази даних та здійснювати зміни безпечно та систематично;
- підтримка запитів і операцій з базою даних. Sequelize надає потужні можливості для створення складних запитів до бази даних, включаючи фільтрацію, сортування, об'єднання, агрегацію та інші операції;
- підтримка транзакцій. Sequelize дозволяє працювати з транзакціями бази даних, забезпечуючи консистентність та цілісність даних під час виконання групи операцій.

Плюси Sequelize:

- легкість використання. Sequelize має простий та зрозумілий API, який дозволяє швидко розпочати роботу з базою даних;
- підтримка різних баз даних. Sequelize підтримує різні реляційні бази даних, що дозволяє переносити код між різними середовищами без значних змін;
- міграції бази даних. Sequelize має вбудовану підтримку міграцій, що дозволяє керувати структурою бази даних та змінами схеми;
- активна спільнота. Sequelize має велику та активну спільноту розробників, що забезпечує швидку підтримку, документацію та регулярні оновлення.

Мінуси Sequelize:

- накладні витрати на виконання запитів. Оскільки Sequelize використовує ORM-підхід, це може викликати деяку накладну витрату на виконання запитів порівняно з прямим використанням SQL-запитів;
- складність при роботі зі складними схемами. При роботі зі складними схемами бази даних, з великою кількістю реляційних зв'язків, може виникати деяка складність та перевантаження коду;

– обмежена підтримка NoSQL баз даних. Sequelize спеціалізується на реляційних базах даних і має обмежену підтримку для NoSQL баз даних, таких як MongoDB.

У лістингу 3.1 відображено побудову моделі бази даних та її підключення.

Лістинг 3.1 «Фрагмент файлу database.js»

```
const db = {};

let Connection = new Sequelize(config.DB_NAME, config.DB_USER,
config.DB_PASS, {
  dialect: "mysql",
  host: config.HOST,
  define: { timestamps: false },
});
await Connection.sync({});

models.forEach((model) => {
  const seqModel = model(Connection, Sequelize);
  console.log(seqModel.name);
  db[seqModel.name] = seqModel;
});

Object.keys(db).forEach((key) => {
  if ("associate" in db[key]) {
    db[key].associate(db);
  }
});

db.Connection = Connection;
db.Sequelize = Sequelize;
```

Як можна бачити в фрагменту файлу 3.1 створюється змінна Connection, яка приймає новий клас Sequelize з параметрами для підключення до БД. Це параметри:

- config.DB_NAME: назва схеми БД («scheduled»);
- config.DB_USER: назва користувача для підключення («admin»);
- config.DB_PASS: пароль користувача («351973»);
- config.HOST: по якій адресі звертатися для підключення («localhost»).

Після цього викликається «Connection.sync({})», який і підключається з даними параметрами.

Наступним кроком є створення моделей (таблиць) в БД. По черзі створюється кожна модель та заноситься до об'єкту db. Після цього для кожної моделі встановлюються асоціації, якщо вони є.

Далі у лістингах 3.2-3.4 будуть наведені приклади створення моделей Specialty, Discipline та різних асоціацій.

Лістинг 3.2 «Фрагмент файлу Models/Specialty.js»

```
export default (Sequelize, DataTypes) => {
  const Specialty = Sequelize.define("specialty", {
    id: {
      type: DataTypes.INTEGER,
      autoIncrement: true,
      primaryKey: true,
      allowNull: false,
    },
    name: {
      type: DataTypes.STRING(100),
      allowNull: false,
    },
    code: {
      type: DataTypes.INTEGER,
      allowNull: false,
    },
    id_cathedra: {
      type: DataTypes.INTEGER,
      allowNull: false,
    },
  });
  Specialty.associate = (models) => {
    Specialty.belongsTo(models.cathedra, {
      foreignKey: "id_cathedra",
      onDelete: "CASCADE",
      onUpdate: "CASCADE",
    });
    Specialty.hasMany(models.assigned_discipline, {
      foreignKey: "id_specialty",
    });
  };
  return Specialty;
};
```

Лістинг 3.3 «Фрагмент файлу Models/Discipline.js»

```
export default (Sequelize, DataTypes) => {
  const Discipline = Sequelize.define("discipline", {
    id: {
      type: DataTypes.INTEGER,
      autoIncrement: true,
      primaryKey: true,
```

```

        allowNull: false,
      },
      name: {
        type: DataTypes.STRING(100),
        allowNull: false,
      },
    });

Discipline.associate = (models) => {
  Discipline.hasMany(models.assigned_discipline, {
    foreignKey: "id_discipline",
  });
};

return Discipline;
};

```

Лістинг 3.4 «Фрагмент файлу Models/Assigned_discipline.js»

```

export default (Sequelize, DataTypes) => {
  const Assigned_discipline = Sequelize.define("assigned_discipline",
  {
    id: {
      type: DataTypes.INTEGER,
      autoIncrement: true,
      primaryKey: true,
      allowNull: false,
    },
    id_specialty: {
      type: DataTypes.INTEGER,
      allowNull: false,
      unique: false,
    },
    id_discipline: {
      type: DataTypes.INTEGER,
      allowNull: false,
      unique: false,
    },
    semester: {
      type: DataTypes.INTEGER,
      allowNull: false,
    },
  });
  Assigned_discipline.associate = (models) => {
    Assigned_discipline.belongsTo(models.specialty, {
      foreignKey: "id_specialty",
      onDelete: "CASCADE",
      onUpdate: "CASCADE",
    });
    Assigned_discipline.belongsTo(models.discipline, {
      foreignKey: "id_discipline",
      onDelete: "CASCADE",
      onUpdate: "CASCADE",
    });
    Assigned_discipline.hasMany(models.class, {
      foreignKey: "id_assigned_discipline",
    });
  };

  return Assigned_discipline;
};

```

```
};
```

Таким чином у кожній моделі у лістингах 3.2-3.4 спочатку викликається метод «Sequelize.define», до якого передається назва та параметри моделі. Після створення моделей у лістингу 3.1 викликаються функції `associate` у кожній моделі, якщо вони є.

У лістингу 3.2 викликається функція «`Specialty.hasMany`», яка вказує на модель «`assigned_discipline`» та її зовнішнім ключем.

У лістингу 3.4 викликається функція «`Assigned_discipline.belongsTo`», яка вказує вже зовнішній ключ вихідної моделі.

Таким чином за допомогою виклику цих двох функцій встановлюється зв'язок один до багатьох (одна `specialty` містить багато `assigned_discipline`).

Теж саме відбувається з `Discipline` у лістингу 3.3 та `assigned_discipline` (одна `discipline` містить багато `assigned_discipline`).

Отже, між моделями `specialty` та `discipline` була побудовано зв'язок багато до багатьох, за допомогою додаткової таблиці `assigned_discipline` та двох зв'язків один до багатьох.

Аналогічна логіка відбувається між асоціаціями й у інших моделях. В результаті можна бачити таку структуру рис. 3.2.

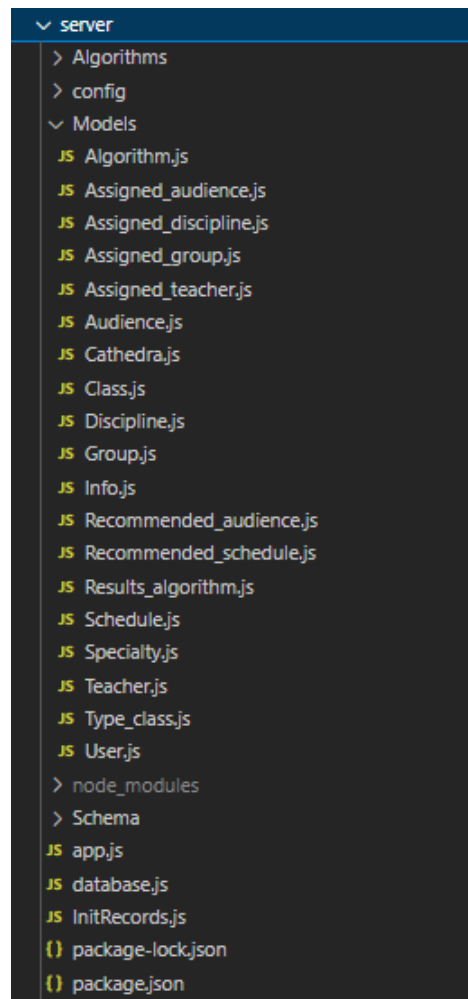


Рисунок 3.2 – Фізична структура моделей

3.2.2 Схема GraphQL

Оскільки GraphQL працює з моделями, параметри яких складаються тільки з типами GraphQL, було створено ще одні моделі даних на основі типів GraphQL. Приклад такої моделі показану у лістингу 3.5.

Лістинг 3.5 «Фрагмент файлу Schema/TypeDefs/ClassType.js»

```
export default new GraphQLObjectType({
  name: "Class",
  fields: () => ({
    id: { type: GraphQLID },
    type_class: { type: Type_classType },
    times_per_week: { type: GraphQLFloat },
    assigned_discipline: { type: Assigned_disciplineType },
    assigned_groups: { type: new GraphQLList(Assigned_groupType) },
```

```

        assigned_teachers: { type: new GraphQLList(Assigned_teacherType)
    },
    recommended_audiences: { type: new
GraphQLList(Recommended_audienceType) },
    recommended_schedules: { type: new
GraphQLList(Recommended_scheduleType) },
    schedules: { type: new GraphQLList(ScheduleType) },
  }},
})

```

Як можна бачити з лістингу 3.5, деякі параметри моделей вже складаються із списків, що полегшує роботу із даними. Таким чином були створені нові моделі GraphQL (рис. 3.3).

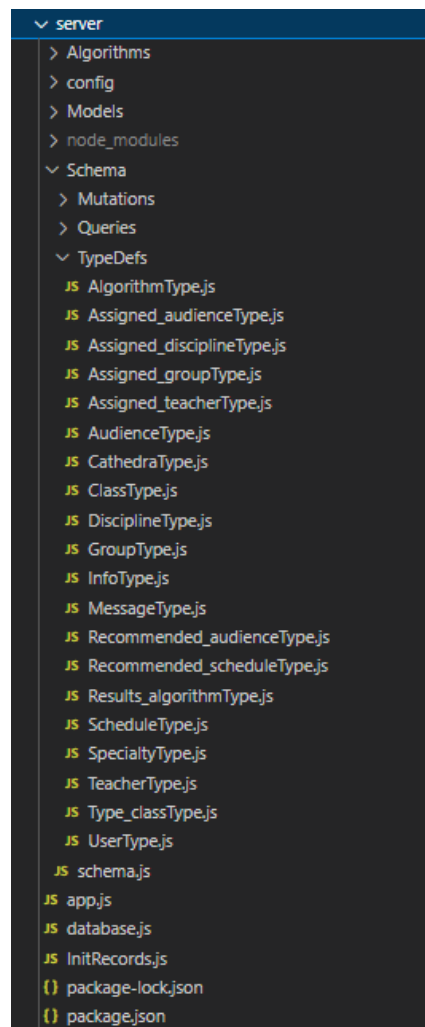


Рисунок 3.3 – Фізична структура моделей GraphQL

Наступним кроком було створення запитів. Було прийнято рішення, що усі запити будуть поділені на два типи – query та mutation. Query-запити

будуть тільки брати дані із БД. А mutation-запити вже будуть змінювати дані в БД. У лістингу 3.6 приклад обробки query-запиту.

Лістинг 3.6 «Фрагмент файлу Schema/Queries/Schedule.js»

```
export const GetAllScheduleGroups = {
  type: new GraphQLList(GroupType),
  args: {
    id_cathetra: { type: GraphQLInt },
    id_specialty: { type: GraphQLInt },
    id_group: { type: GraphQLInt },
    semester: { type: GraphQLInt },
  },
  async resolve(parent, { id_cathetra, id_specialty, id_group, semester }) {
    const FilterSemester = semester ? { semester } : {};
    const FilterGroup = id_group ? { id: id_group } : {};
    let FilterSpecialty = {},
        FilterCathetra = {};
    if (id_specialty) {
      FilterSpecialty = { id_specialty };
    } else if (id_cathetra) {
      FilterCathetra = { id_cathetra };
    }
    const res = await db.group.findAll({
      order: [
        ["semester", "ASC"],
        ["name", "ASC"],
      ],
      where: {
        [Op.and]: [FilterSemester, FilterGroup, FilterSpecialty],
      },
      include: [
        {
          model: db.specialty,
          where: FilterCathetra,
          include: {
            model: db.cathetra,
          },
        },
        {
          model: db.assigned_group,
          include: {
            model: db.class,
            include: [
              {
                model: db.schedule,
                include: {
                  model: db.audience,
                },
              },
            ],
          },
        },
        {
          model: db.type_class,
        },
        {
          model: db.assigned_discipline,
          include: {
```

```

        model: db.discipline,
    },
    {
        model: db.assigned_teacher,
        include: {
            model: db.teacher,
        },
    },
],
},
],
});
return res;
},
};

```

У лістингу 3.6 показано функцію «GetAllScheduleGroups», яка має 4 параметри: `id_cathedra`, `id_specialty`, `id_group`, `semester`. Тобто наявність параметрів означає певний фільтр розкладу. Для отримання даних з БД використовується об'єкт «db» з фрагменту 3.1. Sequelize має багатий набір функцій для роботи з пошуком в БД:

- `findAll`. Генерує стандартний запит SELECT, який буде отримувати всі записи з таблиці (якщо це не обмежено чимось на зразок пункту `where`, наприклад);
- `findByPk`. Отримує лише один запис із таблиці, використовуючи наданий первинний ключ;
- `findOne`. Отримує перший знайдений запис (який виконує додаткові параметри запиту, якщо вони надані);
- `findOrCreate`. Створить запис у таблиці, якщо він не знайде такий, що відповідає параметрам запиту. В обох випадках він поверне екземпляр (або знайдений екземпляр, або створений екземпляр) і логічне значення, яке вказує, чи цей екземпляр був створений чи вже існував;
- `findAndCountAll`. Зручний метод, який поєднує в собі `findAll` і `count`. Це корисно під час роботи із запитами, пов'язаними з розбиттям на сторінки, де потрібно отримати дані з обмеженням і зміщенням, але також потрібно знати загальну кількість записів, які відповідають запиту.

Також дуже комфортно те, що можна задати будь-які складні структури пошуку в залежності від створених моделей, можна їх сортувати, ставити умови.

Лістинг 3.7 «Фрагмент файлу Schema/Mutations/Class.js»

```
export const CreateClass = {
  type: MessageType,
  args: {
    id_type_class: { type: GraphQLInt },
    times_per_week: { type: GraphQLFloat },
    id_assigned_discipline: { type: GraphQLInt },
    assigned_teachers: { type: GraphQLString },
    assigned_groups: { type: GraphQLString },
    recommended_audiences: { type: GraphQLString },
    recommended_schedules: { type: GraphQLString },
  },
  async resolve(
    parent,
    {
      id_type_class,
      times_per_week,
      id_assigned_discipline,
      assigned_teachers,
      assigned_groups,
      recommended_audiences,
      recommended_schedules,
    }
  ) {
    let res = await db.class.create({
      id_type_class,
      times_per_week,
      id_assigned_discipline,
    });
    if (res) {
      if (assigned_teachers) {
        const atIDs = JSON.parse(assigned_teachers);
        await db.assigned_teacher.bulkCreate(
          atIDs.map((item) => {
            return { id_class: res.dataValues.id, id_teacher: item };
          })
        );
      }
      if (assigned_groups) {
        const agIDs = JSON.parse(assigned_groups);
        await db.assigned_group.bulkCreate(
          agIDs.map((item) => {
            return { id_class: res.dataValues.id, id_group: item };
          })
        );
      }
      if (recommended_audiences) {
        const raIDs = JSON.parse(recommended_audiences);
        await db.recommended_audience.bulkCreate(
          raIDs.map((item) => {
            return { id_class: res.dataValues.id, id_audience: item };
          })
        );
      }
    }
  }
}
```

```

        })
      );
    }
    if (recommended_schedules) {
      const raIDs = JSON.parse(recommended_schedules);
      await db.recommended_schedule.bulkCreate(
        raIDs.map((item) => {
          return {
            id_class: res.dataValues.id,
            day_week: item.day_week,
            number_pair: item.number_pair,
          };
        })
      );
    }
  }
  return res
    ? { successful: true, message: "Запис заняття успішно створено" }
    : { successful: false, message: "Помилка при створенні запису
заняття" };
},
);
};

```

Як можна бачити з лістингу 3.7 показано запит «CreateClass», який має 7 параметрів та створює заняття в розкладі. Для керування даними Sequelize має наступні інструменти:

- create. Використовується для створення нового запису в базі даних. Він приймає об'єкт з властивостями, які необхідно зберегти в базі даних;
- bulkCreate. Дозволяє створювати багато записів одночасно в базі даних. Використовуючи цей метод, можна передати масив об'єктів, які містять дані для створення записів;
- update. Дозволяє оновлювати записи в базі даних за заданими критеріями. Використовуючи цей метод, можна вказати нові значення для певних полів у вибраних записах;
- bulkUpdate. Масове оновлення записів в базі даних, використовуючи задані умови;
- destroy. Видаляє записи з таблиці бази даних, використовуючи задані умови;
- bulkDestroy. Масове видалення записів з таблиці бази даних, використовуючи задані умови;

– upsert. Оновлює існуючий запис або створює новий, якщо запис не існує.

Таким чином структура запитів виглядає так (рис. 3.4).

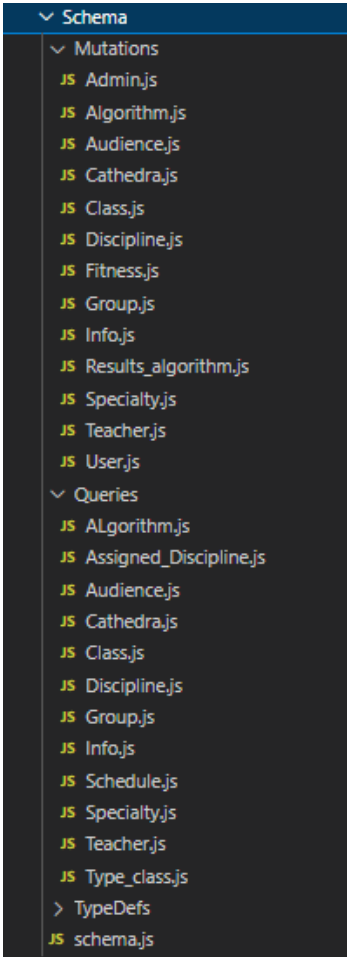


Рисунок 3.4 – Фізична структура запитів GraphQL

Далі у таблиці 3.2 наведено усі запити до вебсерверу та їх опис.

Таблиця 3.2 – Опис запитів GraphQL до вебсерверу

Назва запиту	Опис запиту	Параметри	Опис параметру
1	2	3	4
GetAllAlgorithms	Повертає всі алгоритми з їх результатами		

Продовження таблиці 3.2

1	2	3	4
GetAllAssignedDisciplines	Повертає закріплені дисципліни з дисципліною та спеціальністю		
GetAllAudiences	Повертає аудиторії із закріпленими кафедрами	name	Назва аудиторії
		id_cathedra	До якої кафедри закріплені аудиторії
GetAllCathedras	Повертає кафедри	name	Назва кафедри
GetAllClasses	Повертає заняття із закріпленими викладачами, групами, рекомендованими аудиторіями, часом проведення, дисципліною та спеціальністю	id_group	Які групи враховувати
		id_discipline	Які дисципліни враховувати
		id_teacher	Яких викладачів враховувати
		id_specialty	Які спеціальності враховувати
		semester	Який семестр враховувати
GetAllDisciplines	Повертає дисципліни з спеціальностями та кафедрами	name	Назва дисципліни
		id_specialty	Які спеціальності враховувати
GetAllGroups	Повертає групи з спеціальностями та кафедрами	– name	Назва групи
		id_specialty	Які спеціальності враховувати

Продовження таблиці 3.2

1	2	3	4
GetInfo	Повертає інформацію про параметри розкладу та штрафні значення		
GetAllScheduleGroups	Повертає розклад для груп	id_cathedra	Які кафедри враховувати
		id_specialty	Які спеціальності враховувати
		id_group	Які групи враховувати
		semester	Семестр
GetAllScheduleAudiences	Повертає розклад для аудиторій	id_cathedra	Які кафедри враховувати
		id_audience	Які аудиторії враховувати
GetAllScheduleTeachers	Повертає розклад для викладачів	id_cathedra	Які кафедри враховувати
		id_teacher	Яких викладачів враховувати
GetAllSchedules	Повертає весь розклад		
GetAllSpecialties	Повертає спеціальності з кафедрами	name	Назва спеціальності
		id_cathedra	Які кафедри враховувати

Продовження таблиці 3.2

1	2	3	4
GetAllTeachers	Повертає викладачів з кафедрами	surname	Прізвище
		id_cathedra	Які кафедри враховувати
GetAllTypeClasses	Повертає всі типи занять		
DeleteAllData	Видаляє усі дані або певної кафедри	id_cathedra	Які кафедри враховувати
SetClasses	Конвертує дані з форми відомості до БД	data	Масив занять
		id_cathedra	Які кафедри враховувати
RunAlgorithm	Запускає певний алгоритм для генерації розкладу за кафедрою	name	Назва алгоритму
		id_cathedra	Для якої кафедри скласти розклад
UpdateAlgoritm	Оновлює дані алгоритму	name	Назва алгоритму
		params	Параметри алгоритму
CreateAudience	Створює аудиторію з переданими параметрами	name	Назва аудиторії
		capacity	Місткість
		id_type_class	Тип заняття
		assigned_cathedras	Закріплені кафедри
UpdateAudience	Оновлює аудиторію з переданими параметрами	id	Номер аудиторії
		name	Назва алгоритму
		capacity	Місткість
		id_type_class	Тип заняття

Продовження таблиці 3.2

1	2	3	4
DeleteAudience	Видаляє аудиторію	id	Номер аудиторії
AddCathedraToAudience	Додає кафедру до аудиторії	id_audience	Номер аудиторії
		id_cathedra	Номер кафедри
DeleteCathedraFromAudience	Видаляє кафедру з аудиторії	id	Номер закріпленої кафедри
CreateCathedra	Створює кафедру з переданими параметрами	name	Назва кафедри
		short_name	Коротка назва
UpdateCathedra	Оновлює кафедру з переданими параметрами	id	Номер кафедри
		name	Назва кафедри
		short_name	Коротка назва
DeleteCathedra	Видаляє кафедру	id	Номер кафедри
CreateClass	Створює заняття з переданими параметрами	id_type_class	Тип заняття
		times_per_week	Кількість на тиждень
CreateClass		id_assigned_discipline	Номер закріпленої дисципліни
		assigned_teachers	Закріплені викладачі
		assigned_groups	Закріплені групи
		recommended_audiences	Рекомендовані аудиторії
		recommended_schedules	Рекомендований час

Продовження таблиці 3.2

1	2	3	4
UpdateClass	Оновлює заняття з переданими параметрами	id	Номер заняття
		id_type_class	Тип заняття
		times_per_week	Кількість на тиждень
		id_assigned_discipline	Номер закріпленої дисципліни
DeleteClass	Видаляє заняття	id	Номер кафедри
AddTeacherToClasses	Додає викладача до заняття	id_teacher	Номер викладача
		id_class	Номер заняття
AddRecAudienceToClass	Додає аудиторію до заняття	id_audience	Номер аудиторії
		id_class	Номер заняття;
AddRecScheduleToClass	Додає рекомендований час до заняття	number_pair	Номер пари
		day_week	День тижня
		id_class	Номер заняття
AddGroupToClass	Додає групу до заняття	id_group	Номер групи
		id_class	Номер заняття
DeleteTeacherFromClass	Видаляє викладача з заняття	id	Номер закріпленої викладача
DeleteRecAudienceFromClass	Видаляє аудиторію з заняття	id	Номер закріпленої аудиторії
DeleteRecScheduleFromClass	Видаляє рекомендований час з заняття	id	Номер рекомендованого часу
DeleteGroupFromClass	Видаляє групу з заняття	id	Номер закріпленої групи

Продовження таблиці 3.2

1	2	3	4
CreateDiscipline	Створює дисципліну з переданими параметрами	name	Назва дисципліни
		assigned_disciplines	Закріплені дисципліни
UpdateDiscipline	Оновлює дисципліну з переданими параметрами	id	Номер дисципліни
		name	Назва дисципліни
DeleteDiscipline	Видаляє дисципліну	id	Номер дисципліни
AddDisciplineToSpecialty	Додає дисципліну до спеціальності	id_discipline	Номер дисципліни
		id_specialty	Номер спеціальності
		semester	Семестр
DeleteDisciplineFromSpecialty	Видаляє дисципліну з спеціальності	id	Номер закріпленої дисципліни
CalcFitness	Підраховує фітнес-значення		
CreateGroup	Створює групу з переданими параметрами	name	Назва групи;
		number_students	Кількість студентів
		id_specialty	Номер спеціальності
		semester	Семестр

Продовження таблиці 3.2

1	2	3	4
UpdateGroup	Оновлює групу з переданими параметрами	id	Номер групи
		number_students	Кількість студентів
		id_specialty	Номер спеціальності
		semester	Семестр
		name	Назва групи
DeleteGroup	Видаляє групу	id	Номер групи
UpdateInfo	Оновлює дані розкладу та штрафні значення	data	Дані розкладу та штрафні значення
DeleteResults	Видаляє результати роботи певного алгоритму	name_algorithm	Назва алгоритму
CreateSpecialty	Створює спеціальність з переданими параметрами	name	Назва спеціальності
		id_cathedra	Номер кафедри
		code	Код спеціальності
UpdateSpecialty	Оновлює спеціальність з переданими параметрами	id	Номер групи
		name	Назва спеціальності
UpdateSpecialty		id_cathedra	Номер кафедри
		code	Код спеціальності
DeleteSpecialty	Видаляє спеціальність	id	Номер спеціальності

Кінець таблиці 3.2

1	2	3	4
CreateTeacher	Створює викладача з переданими параметрами	name	Ім'я
		surname	Прізвище
		patronymic	По батькові
		id_cathedra	Номер кафедри
UpdateTeacher	Оновлює викладача з переданими параметрами	id	Номер викладача
		name	Ім'я
		surname	Прізвище
		patronymic	По батькові
		id_cathedra	Номер кафедри
DeleteTeacher	Видаляє викладача	id	Номер викладача
LoginUser	Входить до облікового запису	email	Почта
		password	Пароль
LogoutUser	Виходить з облікового запису	email	Почта
ReloginUser	Оновлює вебтокен користувача	accessToken	Вебтокен

3.2.3 Схема вебсерверу Express

У лістингу 3.8 показано фрагмент самого серверу, його запуску та підключення інших частин.

Лістинг 3.8 «Фрагмент файлу app.js»

```
const main = async () => {
  const app = express();
  let corsOptions = {
    origin: "*",
    credentials: true,
  };
  app.use(cors(corsOptions));
```

```

await db.Connection.sync({})
  .then(async (result) => {
    await InitRecords(db);
    console.log("Connected to DB");
  })
  .catch((err) => console.log(err));

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(
  "/graphql",
  graphqlHTTP({
    schema: Schema,
    graphiql: true,
  })
);

http.createServer(app).listen(config.PORT, () => {
  console.log(`Server HTTP is running on ${config.PORT}`);
});

main().catch((err) => {
  console.log(err);
});

```

Спочатку в лістингу 3.8 йде створення асинхронної функції «main» в якій буде створюватися сервер.

Далі створюється екземпляр Express, який буде використовуватись для створення вебсервера. Налаштовується CORS для дозволу запитів з будь-якого джерела «origin: "*"» та передачі креденціалів «credentials: true».

Далі йде синхронізація моделей Sequelize з базою даних. Виконується метод «sync()», який створює таблиці в базі даних, якщо вони ще не існують. Після синхронізації виконується функція «InitRecords», яка ініціалізує початкові дані. Якщо все успішно, виводиться повідомлення «Connected to DB».

Далі налаштовується middleware для обробки даних, які надходять в форматі JSON. Використовується пакет body-parser для розбору запитів. Крім того, налаштовується шлях «/graphql», на якому будуть оброблятися запити GraphQL за допомогою GraphQL middleware. Вказується схема GraphQL «Schema» і вмикається інтерактивна консоль GraphiQL «graphiql: true», яка дозволяє тестувати та виконувати запити до GraphQL API.

Далі створюється HTTP-сервер з використанням Express-додатку «app». Сервер слухає певний порт «config.PORT», і коли сервер успішно запускається, виводиться повідомлення «Server HTTP is running on {PORT}».

Функцію `main` викликається, і якщо виникають помилки, вони обробляються в блоках `catch`.

3.3 Опис алгоритмів

3.3.1 Алгоритм простого перебору

Оскільки випадкова ініціалізація розкладу дає дуже високе початкове фітнес-значення, було прийнято рішення розробити простий алгоритм перебору. Його назва розкриває його суть – по черзі вставляти пари у заняті, вибираючи краще положення в розкладі.

Основні кроки алгоритма.

Крок 1.

Ініціалізація вхідних даних та створення пустого розкладу.

Крок 2.

Прохід по циклу занять та отримання пар для цього заняття.

Крок 3.

Перевірити чи є рекомендований час проведення у пари, якщо є – перейти до кроку 7.

Крок 4.

Пошук перетинів між вільними парами у груп, викладачів та потенційних аудиторій.

Крок 5.

Вибір перетину з найбільшим числом вільних пар у перетині з потенційною аудиторією.

Крок 6.

Якщо в перетині є пари – обрати випадкову з них, а якщо немає – знайти найбільш вигідне місце проведення пари шляхом перебору можливого часу проведення.

Крок 7.

Додати пару до розкладу та перейти до кроку 2.

Крок 8.

Зберегти результати роботи алгоритму до файлу.

3.3.3 Алгоритм табу пошуку

Основна ідея алгоритму полягає в тому, що він підтримує певний набір "заборонених" рухів або "табу", які допомагають уникнути застрягання в локальних оптимумах. Алгоритм постійно переходить в нові стани, роблячи заборонені рухи, якщо це дозволяє знайти краще рішення.

Основні кроки алгоритма.

Крок 1.

Ініціалізація початкового рішення.

Крок 2.

Ініціалізація списку Табу, що містить заборонені рухи.

Крок 3.

Основний цикл алгоритму, який працює поки фітнес-значення не опуститься до 0, або не буде досягнуто заданої кількості ітерацій.

Крок 4.

Ініціалізація сусідніх розв'язків, шляхом внесення змін в поточне рішення.

Крок 5.

Ініціалізація списків Табу сусідніх рішень.

Крок 6.

Вибір найкращого кандидата: обирається той, у кого менший фітнес та він не містить заборонені кроки.

Крок 7.

Якщо знайшли, то замінюємо поточний розклад на новий.

Крок 8.

Оновлюємо список заборон.

Крок 9.

Перехід до кроку 3.

Крок 10.

Зберегти результати роботи алгоритму до файлу.

3.3.4 Алгоритм імітації відпалу

Суть алгоритму полягає в пошуку найкращого рішення шляхом зміни початкового розкладу випадковим чином. Та можливістю прийняти гірше рішення для уникнення застрягання в локальних оптимумах.

Основні кроки алгоритма.

Крок 1.

Ініціалізація вхідних даних алгоритму та створення двох пустих розкладів.

Крок 2.

Ініціалізація першого розкладу початковим розкладом.

Крок 3.

Основний цикл роботи алгоритма умовою зупинки якого є досягнення максимальної кількості ітерацій або знаходження найкращого рішення.

Крок 4.

Копіювання першого розкладу до другого.

Крок 5.

Виклик функції мутації для зміни другого розкладу.

Крок 6.

Підрахунок різниці між другим та першим розкладом.

Крок 7.

Якщо різниця менше нуля, то перейти до кроку 10, інакше до кроку 8.

Крок 8.

Якщо різниця більше або дорівнює нулю то зробити перевірку «GetRndDouble() < exp(-difference / mainAlgorithm.temperature)», якщо перевірка правдива то перейти до кроку 9, інакше перейти до кроку 10.

Крок 9.

Замінити перший розклад на другий.

Крок 10.

Оновити значення температури та перейти.

Крок 11.

Кінець основного циклу роботи алгоритма.

Крок 12.

Зберегти результати роботи алгоритма до файлу.

3.3.5 Генетичний алгоритм

Суть алгоритму полягає в створенні масиву розкладів та виконання певних дій над ними:

а) мутація – змінює розклад відповідно до типу мутації:

1) мутація одного гена – змінює тільки час проведення однієї пари;

2) мутація усіх генів – змінює час проведення пари в залежності від ймовірності заміни гена;

б) схрещування – комбінує два розклади відповідно до типу схрещування:

1) схрещування одного гена – міняє місцями одну пару між двома розкладами;

2) k-точкове схрещування – міняє місцями пари між розкладами в залежності від кількості випадкових точок по черзі;

в) вибірка – тип відбору індивідів для формування нового покоління:

1) турнірна – бере 3 випадкових різних розклади з популяції та обирає з найменшим фітнес-значенням;

2) рулетка – підраховує шанс виграшу кожного індивіду через загальну суму фітнес-значень індивідів та генерує випадкове число після чого шукає розклад який випав у проміжок цього випадкового числа;

3) ранжування – логіка роботи схожа з рулеткою, проте ймовірність виграшу певного розкладу залежить не від фітнес-значення, а від номеру розкладу у рейтингу, так само генерує випадкове число та шукає проміжок в якому знаходиться розклад.

Основні кроки алгоритма.

Крок 1.

Ініціалізація вхідних даних алгоритму та створення пустих розкладів в залежності від переданого розміру популяції.

Крок 2.

Ініціалізація початкового розкладу для кожного індивіда.

Крок 3.

Основний цикл роботи алгоритму умовою зупинки якого є досягнення максимальної кількості ітерацій або знаходження найкращого рішення.

Крок 4.

Цикл проходу по кожному другому індивіду і якщо випадкове число від 0 до 1 менше або дорівнює ймовірності схрещування – перехід до кроку 5, інакше перехід до кроку 6.

Крок 5.

Виклик функції схрещування між індивідами в багатопоточному режимі.

Крок 6.

Кінець циклу схрещування.

Крок 7.

Цикл проходу по кожному індивіду і якщо випадкове число від 0 до 1 менше або дорівнює ймовірності мутації – перехід до кроку 7, інакше перехід до кроку 8.

Крок 7.

Виклик функції мутації індивіда в багатопоточному режимі.

Крок 8.

Кінець циклу мутації.

Крок 9.

Цикл проходу по кожному індивіду.

Крок 10.

Виклик функції підрахунку фітнес-значення певного розкладу в багатопоточному режимі.

Крок 11.

Кінець циклу фітнесу.

Крок 12.

Виклик функції відбору, яка згенерує нову популяцію.

Крок 13.

Кінець основного циклу роботи алгоритма.

Крок 14.

Зберегти результати роботи алгоритма до файлу.

3.3.6 Модифікація генетичного алгоритму – Острівна модель

Хід роботи алгоритму дуже схожий на хід роботи звичайного генетичного алгоритму. Цей алгоритм виконується в багатопоточному режимі, суть якого полягає в створенні декількох популяцій (островів) які працюють паралельно та можуть обмінюватися індивідами між собою. В схемі роботи є декілька нових операцій – ініціалізація островів з різними параметрами та можливість міграції індивідів між островами.

Основні кроки алгоритма.

Крок 1.

Ініціалізація вхідних даних алгоритму та створення пустих популяцій з різними параметрами в залежності від кроку розбіжності.

Крок 2.

Ініціалізація початкового розкладу в кожному острові для кожного індивіда.

Крок 3.

Основний цикл роботи алгоритму умовою зупинки якого є досягнення максимальної кількості ітерацій або знаходження найкращого рішення.

Крок 4.

Цикл проходу по кожному острову.

Крок 5.

Цикл схрещування аналогічно як в ГА.

Крок 5.

Цикл мутації аналогічно як в ГА.

Крок 6.

Цикл підрахунку фітнес-значень аналогічно як в ГА.

Крок 7.

Цикл відбору аналогічно як в ГА.

Крок 8.

Кінець циклу проходу по кожному острову.

Крок 9.

Якщо потрібно зробити міграцію між островами, то перехід до кроку 10, інакше перехід до кроку 17.

Крок 10.

Цикл проходу по кожному острову.

Крок 11.

Отримання найкращих індивідів з острова.

Крок 12.

Кінець циклу проходу по островам.

Крок 13.

Пошук масиву розкладів з найменшою сумою значень фітнес-функцій.

Крок 14.

Цикл проходу по островам.

Крок 15.

Заміна в острові найгірших індивідів на найкращих з острова.

Крок 16.

Кінець циклу проходу по островам.

Крок 17.

Початок циклу обходу островів.

Крок 18.

Отримання найкращого індивіда з популяції

Крок 19.

Кінець циклу обходу островів.

Крок 20.

Пошук найкращого індивіда серед отриманих з островів.

Крок 21.

Кінець основного циклу роботи алгоритма.

Крок 22.

Зберегти результати роботи алгоритму до файлу.

3.3.7 Нейронна мережа

Для алгоритму з використанням моделі LSTM використовувався модуль `tensorflow.keras`. `TensorFlow.keras` – це високорівневий API для глибокого навчання, що є складовою бібліотеки `TensorFlow`. Він надає зручний інтерфейс для побудови та навчання глибоких нейронних мереж. `TensorFlow.keras` дозволяє легко визначати архітектуру вашої нейронної мережі за допомогою послідовного або функціонального API і містить

широкий набір вбудованих шарів, функцій активації, оптимізаторів та функцій втрат для конструювання моделі.

Модель LSTM буде використовуватися для прогнозування дня тижня, номеру пари, на вхід якої буде подаватися дані про заняття. Модель нейронної мережі визначається за допомогою TensorFlow.keras з використанням Sequential API. Основною частиною моделі є LSTM шар, який має параметри: кількість нейронів ($\text{day_week} * \text{number_pair}$), dropout (відсічення), return_sequences (повертати послідовності) та input_shape (форма вхідних даних). Після LSTM шару додається Dense шар з кількістю нейронів $\text{day_week} * \text{number_pair}$ та активацією, яка вказується параметром activation. Далі модель компілюється з використанням визначеного втрати (loss), оптимізатора (optimizer) та метрик (metrics). Параметри моделі завантажуються з файлу modelParams.json.

Для навчання та тренування моделі була створена вибірка даних у вигляді файлів CSV, які містять таблицю з розкладом: день тижня, номер пари, тип пари, ідентифікатор заняття, тип заняття, ідентифікатор аудиторії, викладача та групи. Всі розклади мають нульове значення фітнес. Вибірка завантажується з директорії DatasetSchedulesCSV та ділиться на вхідну та вихідну. Для вхідної вибірки використовується кодування категоріальних ознак в числовий формат за допомогою об'єкта LabelEncoder з бібліотеки scikit-learn. Вихідна вибірка кодується за допомогою визначеною функції encodeData, яка здійснює кодування даних у форматі "one-hot encoding" для вхідного набору даних.

У циклі для кожного рядка даних виконується наступне:

- створюється тимчасовий масив розміром ($\text{day_week} * \text{number_pair}$), що складається з нулів;
- за допомогою формули $(\text{day}-1)*\text{number_pair} + \text{number}-1$ визначається індекс, в який буде встановлене значення 1;
- значення 1 присвоюється елементу за визначеним індексом.

Після обробки всіх рядків даних, функція повертає список, що містить закодовані дані у форматі "one-hot encoding". Кожен рядок у списку представляє вектор, в якому лише один елемент має значення 1, що відповідає конкретному дню тижня та номеру пари. Решта елементів у векторі мають значення 0.

За допомогою функції `TrainAndTestModel` виконується тренування та тестування моделі машинного навчання. У функції спочатку виконується розділення вхідних на тренувальний та тестовий набори за допомогою функції `train_test_split`. Розмір тестового набору задається параметром `test_size`, а `random_state` визначає генерацію випадкових чисел для розділення даних.

Далі модель `model` тренується на тренувальних даних та за заданою кількістю епох `epochs`, розміром пакета `batch_size` та рівнем деталей `verbose`.

Після тренування моделі виконується оцінка втрати `loss` та точності `acc` на тестових даних `dataTestX` та `dataTestY` за допомогою методу `evaluate` моделі.

`model.save` є методом в бібліотеці Keras, який використовується для збереження моделі машинного навчання у файл. Вказується шлях до файлу, де буде збережена модель. `load_model` є функцією в бібліотеці Keras, яка використовується для завантаження збереженої моделі з файлу.

Основні кроки роботи алгоритма.

Крок 1.

Ініціалізація параметрів алгоритму: встановлюються шляхи до директорій з файлами параметрів, вхідної вибірки, файлу з розкладом.

Крок 2.

Перевіряється файл з моделлю, якщо він визначений перехід на крок 7.

Крок 3.

Якщо файл не визначений, то визначається архітектура нейронної мережі.

Крок 4.

Зчитується та обробляється вхідні дані.

Крок 5.

Тренування та тестування мережі.

Крок 6.

Збереження моделі в файл.

Крок 7.

Завантаження моделі з файлу.

Крок 8.

Обробка даних для прогнозування розкладу.

Крок 9.

Запис розкладу у файл з результатами.

3.4 Висновок

Було розроблено схему даних для БД на її основі побудовані моделі для вебсерверу ExpressJS. Для роботи з моделями та БД було описано використання Sequelize. Були розроблені моделі для схеми GraphQL та запити які поділяються на queries та mutations та наведено приклади їх роботи та опис кожного з них. Було описано налаштування вебсерверу та його запуск.

Було описано основні кроки роботи алгоритмів.

4 ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Загальні дані для тестування алгоритмів

Вхідні дані для проведення тестування генерації розкладу наведені в додатку А.1. Характеристики обчислювальної машини для проведення тестів (табл. 4.1).

Таблиця 4.1 – Характеристики обчислювальної машини для тестів

Назва характеристики	Значення
1	2
Процесор	AMD Ryzen 5 2600
Частота процесора	3.9 ГГц
Кількість ядер	6
Кількість потоків	6
Обсяг оперативної пам'яті	8,00 ГБ
Швидкість оперативної пам'яті	3200 МГц
Тип жорсткого диску	SSD
Операційна система	Virtual Machine Windows Server 2022

Для порівняння роботи алгоритму використовувався параметр Швидкість збіжності. Швидкість збіжності показує наскільки швидко або ефективно алгоритм наближається до свого оптимального розв'язку. Цей параметр можна розрахувати за наступною формулою (4.1):

$$S = \frac{|F_s - F_f|}{|t_f - t_s|}, \quad (4.1)$$

де S – швидкість збіжності;

F_s – початкове значення фітнес;

F_f – кінцеве значення фітнес;

t_s – початковий час роботи алгоритму;

t_f – кінцевий час.

Також слід позначити наступні скорочення:

а) P_e – значення елітизму (elitism), тобто відсоток від загального числа популяції;

б) T_c – тип схрещування (crossing), може приймати такі значення:

1) 1 – «custom_one_gene»;

2) 2 – «k_point»;

в) N_k – кількість k-точок схрещування;

г) P_c – ймовірність схрещування;

е) T_m – тип мутації (mutation), може приймати такі значення:

1) 1 – «custom_one_gene»;

2) 2 – «all_genes»;

є) P_{mg} – ймовірність мутації гена;

ж) P_m – ймовірність мутації;

з) T_s – тип вибірки (selection), може приймати такі значення:

1) 1 – «roulette»;

2) 2 – «ranging»;

3) 3 – «tournament»;

и) T_i – тип ініціалізації (initialization), може приймати такі значення:

1) 1 – «random»;

2) 2 – «simple_algorithm»;

і) N_g – максимальна кількість ітерацій роботи алгоритму;

ї) N_p – розмір популяції.

Також слід зазначити наступні параметри з якими проводилися тести (табл. 4.2).

Таблиця 4.2 – Загальні параметри для проведення тестів

Параметр	Значення
1	2
Кількість днів для складання розкладу	6
Кількість пар для складання розкладу	6
Штраф за вікна у груп	1
Штраф за вікна у викладачів	1
Штраф за накладання занять	5
Штраф за невідповідність рекомендованого часу проведення заняття	5

4.2 Результати тестів генетичного алгоритму

В табл. 4.3 наведено результати роботи генетичного алгоритму та жовтим кольором виділені параметри які тестуються. А в табл. 4.4 вже наведені результати тестів, при чому слід зазначити, що кожний запис – середнє значення з трьох проведених експериментів з однаковими параметрами. В усіх тестах розмір популяції – 500 та максимальне число ітерацій – 2000.

Таблиця 4.3 – Параметри запуску тестів генетичного алгоритму

#	P_e	T_c	N_k	P_c	T_m	P_{mg}	P_m	T_s	T_i
1	2	3	4	5	6	7	8	9	10
1	0,1	1		0,7	1		0,2	1	1
2	0,1	1		0,7	1		0,4	1	1
3	0,1	1		0,7	1		0,6	1	1
4	0,1	1		0,7	2	0,05	0,2	1	1
5	0,1	1		0,7	2	0,1	0,2	1	1
6	0,1	1		0,7	2	0,2	0,2	1	1

Кінець таблиці 4.3

1	2	3	4	5	6	7	8	9	10
7	0,1	1		0,4	1		0,2	1	1
8	0,1	1		0,6	1		0,2	1	1
9	0,1	1		0,8	1		0,2	1	1
10	0,1	1		0,6	1		0,2	1	1
11	0,2	1		0,6	1		0,2	1	1
12	0,3	1		0,6	1		0,2	1	1
13	0,1	1		0,6	1		0,2	1	1
14	0,1	1		0,6	1		0,2	2	1
15	0,1	1		0,6	1		0,2	3	1
16	0,1	1		0,6	1		0,2	3	1
17	0,1	2	2	0,6	1		0,2	3	1
18	0,1	2	4	0,6	1		0,2	3	1
19	0,1	2	6	0,6	1		0,2	3	1
20	0,1	1		0,6	1		0,2	3	1
21	0,1	1		0,6	1		0,2	3	2

Таблиця 4.4 – Результати генетичного алгоритму

#	t_s , c	t_f , c	F_s	F_f	S
1	2	3	4	5	6
1	0.98	1937.364667	1128.666667	22	0.577405312
2	0.97	1852.033333	1157.333333	24.66666667	0.612675785
3	0.980666667	1884.862	1148	119	0.546162918
4	0.990666667	1840.709	1178.666667	75	0.600268487
5	0.964333333	1865.483667	1164.333333	80.33333333	0.581390666
6	1.008333333	1881.805333	1156.666667	54.33333333	0.586229139
7	0.953333333	1851.584667	1183.666667	22	0.62777253
8	0.952	1796.640333	1165	17.66666667	0.638994163

Кінець таблиці 4.4

1	2	3	4	5	6
9	0.965666667	1808.653667	1164	15	0.635672784
10	0.967333333	1813.074	1158.333333	15	0.630908556
11	0.96	1794.840667	1158.333333	14	0.637921701
12	0.979	1840.432333	1148.333333	18	0.614525478
13	0.975	1832.626	1163.666667	15.33333333	0.626933055
14	0.982666667	1878.421333	1165.333333	10	0.615395673
15	1.002666667	1839.763333	1155.333333	14.66666667	0.620395521
16	1.005666667	1830.968	1158.666667	16.33333333	0.624708421
17	1.022666667	1980.765333	1159	15.66666667	0.577514042
18	1.134666667	2196.617	1156.666667	10	0.522431889
19	1.022666667	1980.765333	1159	15.66666667	0.577514042
20	1.077666667	2043.33	1157.333333	13.66666667	0.564996907
21	1.116333333	947.481	8.333333333	1.333333333	0.013857401

Проаналізувавши таблицю 4.3 та 4.4 можна зробити наступні висновки щодо найбільшої швидкості збіжності у параметрів:

- ймовірність мутації – 0.4;
- тип мутації – мутація одного гену;
- ймовірність схрещування – 0.8 (значення дуже близькі між собою);
- значення елітизму – 0.2;
- тип вибірки – рулетка (значення дуже близькі між собою);
- тип схрещування – схрещування одного гену;
- тип ініціалізації – за допомогою алгоритму простого перебору.

4.3 Результати тестів модифікації генетичного алгоритму – острівної моделі

Слід ввести наступні позначення:

- N_i – кількість островів;
- N_{st} – крок збільшення значень у островах;
- N_{it} – кількість ітерацій через яку проводити міграцію між островами;
- P_{mig} – яка кількість індивідів буде приймати участь в міграції.

В табл. 4.5 наведені параметри тестів модифікації генетичного алгоритму – острівної моделі. А в табл. 4.6 наведені вже результати роботи. Кожний тест проводився по 3 рази та вираховувалося середнє значення. Розмір популяції – 500 та максимальне число ітерацій – 2000 для всіх тестів.

Таблиця 4.5 – Параметри запуску тестів модифікації генетичного алгоритму – острівної моделі

#	P_e	T_c	N_k	P_c	T_m	P_{mg}	P_m	T_s	T_i	N_i	N_{st}	N_{it}	P_{mig}
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0,1	1		0,4	1		0,2	1	1	6	2	10	0,1
2	0,1	1		0,4	1		0,2	1	1	12	2	10	0,1
3	0,1	1		0,4	1		0,2	1	1	18	2	10	0,1
4	0,1	1		0,4	1		0,2	1	1	6	3	10	0,1
5	0,1	1		0,4	1		0,2	1	1	6	5	10	0,1
6	0,1	1		0,4	1		0,2	1	1	6	7	10	0,1
7	0,1	1		0,4	1		0,2	1	1	6	7	5	0,1
8	0,1	1		0,4	1		0,2	1	1	6	7	8	0,1
9	0,1	1		0,4	1		0,2	1	1	6	7	13	0,1
10	0,1	1		0,4	1		0,2	1	1	6	7	5	0,05
11	0,1	1		0,4	1		0,2	1	1	6	7	5	0,15
12	0,1	1		0,4	1		0,2	1	1	6	7	5	0,25

Таблиця 4.6 – Результати роботи тестів модифікації генетичного алгоритму – острівної моделі

#	t_s , с	t_f , с	F_s	F_f	S
1	2	3	4	5	6
1	1.14	1924.1	1123.67	14.6667	0.57678
2	2.21	3663.41	1101	6.33333	0.2991
3	3.07	5399.51	1118.67	4	0.20657
4	1.12	1835.98	1119.33	9.33333	0.60493
5	1.12	1892.26	1112.67	6	0.58544
6	1.10	1893.74	1094.33	5.33333	0.57553
7	1.16	1898.34	1131	5	0.59348
8	1.06	1929.85	1124	8	0.57871
9	1.12	1880.43	1152.33	10.6667	0.60774
10	1.07	1899.74	1143	5.66667	0.59923
11	1.08	1913.29	1109	7	0.57629
12	1.16	1906.08	1146.67	11	0.59627

Проаналізувавши таблицю 4.5 та 4.6 можна зробити наступні висновки щодо найбільшої швидкості збіжності у параметрів:

- кількість островів – 6 (кількість логічних ядер процесору);
- крок розбіжності параметрів – 3;
- через скільки ітерацій проводити міграцію – 5;
- кількість індивидів для участі в міграції – 0.05 (значення дуже близькі між собою).

4.4 Результати тестів алгоритму імітації відпалу

Слід ввести наступні позначення:

- Temp – початкова температура для імітації відпалу;

– Alpha – коефіцієнт множення температури на саму себе.

В табл. 4.7 наведено результати роботи алгоритму імітації відпалу. Для кожного тесту максимальна кількість ітерацій була однакою – 200000 та кожний тест проводився по 3 рази та обиралося середнє значення.

Таблиця 4.7 – Результати роботи тестів алгоритму імітації відпалу.

#	T_m	P_{mg}	T_i	Temp	Alpha	t_s, c	t_f, c	F_s	F_f	S
1	2	3	4	5	6	7	8	9	10	11
1	1		1	100000	0.99	0.51	358.51	1316	2.6667	6.9349
2	2	0.05	1	100000	0.99	0.45	639.56	1250	372.67	1.3733
3	2	0.15	1	100000	0.99	0.47	670.35	1405	699.67	1.0527
4	1		1	100000000	0.99	0.48	192.75	1330.33	2	7.7573
5	1		1	100000	0.8	0.42	131.42	1134.67	2	8.8305
6	1		2	100	0.99	0.48	156.94	639.333	1.3333	4.175

Проаналізувавши таблицю 4.7 можна зробити наступні висновки щодо найбільшої швидкості збіжності у параметрів:

- тип мутації – однозначно мутація одного гену;
- температура – виявилася краще та, яка має менше значення 10^5 ;
- коефіцієнт альфа – 0.8;
- тип ініціалізації – за допомогою алгоритму простого перебору.

4.5 Результати тестів табу пошуку

Слід ввести наступні позначення:

- N_{tt} – кількість ітерацій дії табу;
- N_n – кількість сусідніх розв’язків;
- N_d – довжина списку табу.

В табл. 4.8 наведено результати роботи алгоритму табу пошуку. Для кожного тесту максимальна кількість ітерацій була однаковою – 5000 та кожний тест проводився по 3 рази та обиралися середнє значення.

Таблиця 4.8 – Результати тестування алгоритму табу пошуку

#	T _m	P _{mg}	T _i	N _{tt}	N _n	N _{tl}	t _s	t _f	F _s	F _f	S
1	2	3	4	5	6	7	8	9	10	11	12
1	1		1	10	10	50	0.04	150.157	1323	7.33333	12.8967
2	2	0.05	1	10	10	50	0.03	192.472	1274.33	43.6667	6.39401
3	1		1	20	10	50	0.03	114.101	1287	1.33333	14.4287
4	1		1	30	10	50	0.03	188.309	1268	2	6.79257
5	1		1	30	20	50	0.05	257.322	1329.33	21	5.09907
6	1		1	30	30	50	0.07	317.511	1263	5.33333	3.96421
7	1		1	30	30	100	0.07	276.071	1323.67	406.667	16.9409
8	1		2	30	30	100	0.07	292.891	16.6667	5.33333	0.10436

Проаналізувавши таблицю 4.8 можна зробити наступні висновки щодо найбільшої швидкості збіжності у параметрів:

- тип мутації – мутація одного гену;
- кількість ітерацій дії табу – 30;
- кількість сусідніх розв’язків – 30;
- довжина списку табу – 100;
- тип ініціалізації – алгоритм простого перебору.

4.6 Результати тестів нейронної мережі

В ході проведених експериментів, було встановлено такі параметри нейронної мережі: функція активації softmax, функція втрати categorical_crossentropy, алгоритм оптимізації adam, метрика "accuracy", що

вимірює точність класифікації моделі, розмір пакета 1, кількість епох 500, частка даних, які будуть використовуватися для тестування моделі 0.2. Тестування моделі з цими параметрами наведено в табл. 4.9.

Таблиця 4.9 – Тестування нейронної мережі

Номер тесту	Фітнес-значення
1	2
1	3351
2	2765
3	2803
4	2843
5	2843

Отже, з результатів таблиці можна зробити висновки, що складена модель погано складає розклад занять. Існує кілька можливих причин, чому модель LSTM може погано складати розклад занять:

- недостатні дані. Модель LSTM потребує достатньої кількості якісних і репрезентативних даних для ефективного навчання;
- неправильна обробка даних;
- неправильна архітектура моделі;
- недостатня тривалість навчання. Модель LSTM може потребувати більш тривалого навчання для досягнення найкращих результатів;
- недостатня розмірність моделі;
- неправильний підбір гіперпараметрів.

Загалом, використання моделі LSTM для складання розкладу занять може бути вимагаючим завданням, і потребує уважного підходу до підготовки даних, налаштування моделі та експериментування з параметрами.

4.7 Висновок

В табл. 4.10 занесено результати роботи алгоритмів з найкращими параметрами.

Таблиця 4.10 – Результати тестів алгоритмів з найкращими параметрами

Алгоритм	№	t_s , с	t_f , с	F_s	F_f	S
1	2	3	4	5	6	7
Імітація відпалу	1	0,023	68,1	40	0	0.59
	2	0,037	175	15	0	0.09
	3	0,05	47	26	0	0.55
	Сер.	0,37	96,7	27	0	0.41
Генетичний алгоритм	4	1	1290	14	0	0.011
	5	1	734	10	0	0.013
	6	1	913	6	0	0.007
	Сер.	1	979	10	0	0.01
Модифікація ГА – Острівна модель	7	1	113	6	0	0.054
	8	1	256	12	0	0.047
	9	1	439	8	0	0.018
	Сер.	1	269,3	8,6	0	0.04
Табу пошук	10	0,069	395	2	2	0
	11	0,1	424	12	4	0,019
	12	0,082	406	8	4	0,01
	Сер.	0,084	408,3	7,3	3,3	0,01

Проаналізувавши отримані результати (табл. 4.10) тестування алгоритмів з найкращими параметрами для цієї задачі можна зробити наступні висновки:

– функція швидкості збіжності найбільше значення має в алгоритмі імітації відпалу – 0.41, далі ОМ – 0.04, далі ГА – 0.01 та ТБ – 0.01. Це говорить про те що швидкість збіжності краще має ІВ саме в цій задачі;

– всі алгоритми змогли знайти кращий розв’язок у відведений час виконання крім табу пошуку;

– наступний параметр порівняння сер. час роботи алгоритму, найкраще значення має ІВ – 96.7, далі ОМ – 269.3, далі ТБ – 408.3, далі ГА – 979. Це говорить про відверту перевагу роботи ІВ над іншими алгоритмами, а також про те що модифікація ГА значно покращує його роботу.

ВИСНОВКИ

В процесі виконання кваліфікованої роботи бакалавра було розроблено вебсервер системи автоматизованого складання та керування розкладу для занять в навчальних закладах. А також розроблені та протестовані алгоритми генерації розкладу.

Було проведено аналіз предметної області, виявлені основні поняття та залежності, поставлено завдання до виконання. Було досліджено які функції слід оптимізувати в ході роботи алгоритмів, а саме: функція підрахунку вікон між парами, функція підрахунку накладок пар та функція перевірки рекомендованого часу проведення до пар. Було проведено порівняльну характеристику наявних засобів складання та керування розкладу з якої було складено вимоги до нашої системи.

Було проведено порівняння мов програмування та їх фреймворків для розробки вебсерверу. Як результат було обрано розроблювати вебсервер на основі фреймворку ExpressJS. Було проведено аналіз архітектур та шаблонів проєктування вебзастосунків. Як результат обрано поєднання таких архітектур: 3-рівнева, SPA, GraphQL та обрано шаблон проєктування Model-View-Presenter. Було проведено порівняння та аналіз баз даних в результаті якого прийнято рішення користуватися БД MySQL.

Було надано структуру БД та описані таблиці, їх параметри та зв'язки. Було надано структуру вебсерверу: моделі sequelize, моделі graphql, запити graphql та налаштування домену за допомоги nginx. Було проведено детальний опис кожного запиту до вебсерверу.

Було обрано такі мови програмування для розробки алгоритмів: C++ для розробки нескладних алгоритмів які потребують швидкого виконання та Python для розробки нейронної мережі.

Було розроблено власно створений алгоритм ініціалізації початкового розкладу суть якого полягає у переборі пар для занять та підстановки їх до найкращого часу проведення шляхом пошуку перетинів вільних пар у груп,

викладачів та аудиторій. Це дозволило зменшити початкове значення фітнес-функції з приблизно 1200 до 0-50.

Далі було проведено тестування розроблених алгоритмів за допомоги функції швидкості збіжності для відбору найкращих параметрів для кожного з них. Після чого було проведено тестування між алгоритмами з найкращими параметрами. В результаті якого для конкретного розкладу (додаток А.1) найкращим алгоритмом виявився – алгоритм імітації відпалу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Solving Timetable Problem by GeneticAlgorithm and Heuristic Search Case Study:Universitas Pelita Harapan Timetable [Electronic resource]. – Access mode:

https://www.researchgate.net/publication/221927228_Solving_Timetable_Problem_by_Genetic_Algorithm_and_Heuristic_Search_Case_Study_Universitas_Pelita_Harapan_Timetable.

2. An Evolutionary Algorithm for Solving Academic Courses Timetable Scheduling Problem [Electronic resource]. – Access mode:

https://www.researchgate.net/publication/356044560_An_Evolutionary_Algorithm_for_Solving_Academic_Courses_Timetable_Scheduling_Problem.

3. A Review of Optimization Algorithms for University Timetable Scheduling [Electronic resource]. – Access mode:

https://www.researchgate.net/publication/347802207_A_Review_of_Optimization_Algorithms_for_University_Timetable_Scheduling

4. Some Methods of Solving the NP-difficult Problem of Optimal Schedule for the University [Electronic resource]. – Access mode:

<https://www.sciencedirect.com/science/article/pii/S187705091930417X#:~:text=However%2C%20there%20are%20a%20number%20of%20heuristic%20and,method%2C%20graph%20coloring%20method%2C%20intelligent%20method%2C%20genetic%20algorithm>.

5. University Timetable Generator Using Tabu Search [Electronic resource]. – Access mode:

https://www.researchgate.net/publication/322298839_University_Timetable_Generator_Using_Tabu_Search.

6. Simulated Annealing Algorithm for Deep Learning [Electronic resource]. – Access mode:

<https://www.sciencedirect.com/science/article/pii/S1877050915035759>.

7. A review on genetic algorithm: past, present, and future [Electronic resource]. – Access mode: <https://link.springer.com/article/10.1007/s11042-020-10139-6>.

8. Субботін, С. О. Нейронні мережі : навчальний посібник / С. О. Субботін, А. О. Олійник ; під заг. ред. проф. С. О. Субботіна. – Запоріжжя : ЗНТУ, 2014. – 135 с.

9. Добровська Л. М. Теорія та практика нейронних мереж : навч. посіб. / Л. М. Добровська, І. А. Добровська. – К. : НТУУ «КПІ» Вид-во «Політехніка», 2015. – 396 с. – Бібліогр. : С. 385–387. – 55 пр.

10. Новотарський М. А. Штучні нейронні мережі: обчислення / М. А. Новотарський, Б. Б. Нестеренко // Праці Інституту математики НАН України. – Т50. – Київ: Ін-т математики НАН України, 2004. – 408 с.

11. Long short-term memory [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Long_short-term_memory.

12. Untis Timetabe scheduling [Electronic resource]. – Access mode: <https://www.untis.at/en/products/untis-timetabe-scheduling>.

13. TimeTabler Timetabe scheduling [Electronic resource]. – Access mode: <https://www.timetabler.com/timetabler>.

14. TimeEdit Timetabe scheduling [Electronic resource]. – Access mode: <https://www.timeedit.com/products/te-core/overview>

15. What Is JavaScript? How It Works & What JavaScript Is Used For [Electronic resource]. – Access mode: <https://learntocodewith.me/learn/javascript>.

16. ExpressJS Tutorial: The Ultimate Guide (2023) [Electronic resource]. – Access mode: <https://masteringbackend.com/posts/expressjs-5-tutorial-the-ultimate-guide>.

17. NestJS Typescript: The Ultimate Guide (2022) [Electronic resource]. – Access mode: <https://masteringbackend.com/posts/nestjs-typescript-ultimate-guide>.

18. Why Learning Python is Perfect for Beginners, Career-Changers, and Anyone Else [Electronic resource]. – Access mode: <https://learntocodewith.me/resources/why-learn-python>.

19. Django Tutorial: The Ultimate Guide (2022) [Electronic resource]. – Access mode: <https://masteringbackend.com/posts/django-tutorial-the-ultimate-guide>.
20. What is Flask Python [Electronic resource]. – Access mode: <https://pythonbasics.org/what-is-flask-python>.
21. What is C# [Electronic resource]. – Access mode: <https://riptutorial.com/csharp/learn/100000/overview>.
22. Features of C++ [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/features-of-cpp>.
23. Web Application Architecture: The Ultimate Guide [Electronic resource]. – Access mode: <https://www.intellectsoft.net/blog/web-application-architecture>.
24. MVC, MVP, MVVM: Which One to Choose [Electronic resource]. – Access mode: <https://www.makeuseof.com/mvc-mvp-mvvm-which-choose>.
25. The Ultimate Guide to Open Source Databases [Electronic resource]. – Access mode: <https://www.percona.com/blog/ultimate-guide-open-source-databases>.

ДОДАТОК А
ДАНІ РОБОТИ АЛГОРИТМІВ

А.1 Таблиця вхідних даних з відомості заручень.

Таблиця А.1 – Вхідні дані для алгоритмів з відомості заручень

№	Назва дисципліни	№ груп	Вид	Кількість	Прізвище Ім'я По батькові викладачів	Пропозиції
1	2	3	4	5	6	7
1	Основи програмування	КНТ-110+120тр;120+120та;130	лк	2	Льовкін Валерій Миколайович	45.55
		КНТ-110+120тр	пр	1	Петрова Ольга Анатоліївна	57
		КНТ-120+120та	пр	1	Петрова Ольга Анатоліївна	57
		КНТ-130	пр	1	Петрова Ольга Анатоліївна	57
		КНТ-110+120тр	лб	3	Дейнега Лариса Юріївна	57
		КНТ-120+120та	лб	3	Дейнега Лариса Юріївна	57
		КНТ-130	лб	3	Дейнега Лариса Юріївна	57
2	Алгоритмізація та прогармування	КНТ-210;220+210тб	лк	2	Гладкова Ольга Миколаївна	44.55
		КНТ-210	лб	2	Качан Олександр Іванович	58
		КНТ-220+210тб	лб	2	Качан Олександр Іванович	58
3	Алгоритми та структури даних	КНТ-119;129;139;110те+120та+120тр+120тг	лк	2	Миронова Наталя Олексіївна	45.55
		КНТ-119	лб	2	Короткий Олександр Володимирович	58
		КНТ-129	лб	2	Короткий Олександр Володимирович	58
		КНТ-139	лб	2	Кривцун Олена Володимирівна	58
		КНТ-110те	лб	2	Дейнега Лариса Юріївна	58
		КНТ-120те+120та+120тр+120тг	лб	2	Терещенко Еліна Валентинівна	58

Продовження таблиці А.1

1	2	3	4	5	6
4	Людино-машинна взаємодія	КНТ-119;129;139;110те+120та+120тр+120тг	лк	2	Сердюк Сергій Микитович 45.55
		КНТ-119	лб	2	Сердюк Сергій Микитович 49
		КНТ-129	лб	2	Сердюк Сергій Микитович 49
		КНТ-139	лб	2	Камінська Жанна Константинівна 49
		КНТ-110те	лб	2	Камінська Жанна Константинівна 49
		КНТ-120те+120та+120тр+120тг	лб	2	Камінська Жанна Константинівна 49
5	Основи програмної інженерії	КНТ-119;129+120та+120тр+120тг;139	лк	1	Каплієнко Тетяна Ігорівна 45.55
		КНТ-119	лб	2	Качан Олександр Іванович 49
		КНТ-129+120та+120тр+120тг	лб	2	Качан Олександр Іванович 49
		КНТ-139	лб	2	Качан Олександр Іванович 49
		КНТ-119	лб	1	Качан Олександр Іванович 56
		КНТ-129+120та+120тр+120тг	лб	1	Качан Олександр Іванович 56
		КНТ-139	лб	1	Качан Олександр Іванович 56
6	Теорія алгоритмів	КНТ-219+210та;229+210тб	лк	2	Миронова Наталя Олексіївна 45.55
		КНТ-219+210та	лб	2	Дейнега Лариса Юріївна 49
		КНТ-229+210тб	лб	2	Дейнега Лариса Юріївна 49
7	Комп'ютерна схемотехніка та архітектура комп'ютерів	КНТ-219+210та;229+210тб	лк	2	Корнієнко Сергій Константинович 45.55
		КНТ-219	лк	2	Корнієнко Сергій Константинович 56
		КНТ-229	лк	2	Корнієнко Сергій Константинович 56
		КНТ-210те+210тб	лк	2	Корнієнко Сергій Константинович 56

Продовження таблиці А.1

1	2	3	4	5	6
8	Чисельні методи	КНТ-219+210те;229+210тб	лк	2	Миронова Наталя Олексіївна 45.55
		КНТ-219+210те	лб	1	Кривцун Олена Володимирівна 58
		КНТ-229+210тб	лб	1	Кривцун Олена Володимирівна 58
9	Аналіз вимог до програмного забезпечення	КНТ-118;128+119т;138+129т	лк	1	Каплієнко Тетяна Ігорівна 45.55
		КНТ-118	лб	2	Камінська Жанна Констянтинівна 49
		КНТ-128+119т	лб	2	Камінська Жанна Констянтинівна 49
		КНТ-138+129т	лб	2	Камінська Жанна Констянтинівна 49
10	Професійна практика програмної інженерії	КНТ-118;128+119т;138+129т	лк	1	Льовкін Валерій Миколайович 45.55
		КНТ-118	лб	2	Зайко Тетяна Анатоліївна 58
		КНТ-128+119т	лб	2	Петрова Ольга Анатоліївна 58
		КНТ-138+129т	лб	2	Петрова Ольга Анатоліївна 58
11	Архітектура та проєктування програмного забезпечення	КНТ-118;128+119т;138+129т	лк	1	Олійник Андрій Олександрович 45.55
		КНТ-118	лб	3	Колпакова Тетяна Олексіївна 49
		КНТ-128+119т	лб	3	Леоценко Сергій Дмитрович 49
		КНТ-138+129т	лб	3	Леоценко Сергій Дмитрович 49
12	Інженерія інтелектуальних систем	КНТ-118;128+119т;138+129т	лк	1	Субботін Сергій Олександрович 45.55
		КНТ-118	лб	1	Каврін Дмитро Анатолійович 56
		КНТ-128+119т	лб	1	Каврін Дмитро Анатолійович 56
		КНТ-138+129т	лб	1	Каврін Дмитро Анатолійович 56
13	Емпіричні методи в інформаційних технологіях	КНТ-118;128+119т;138+129т	лк	1	Олійник Андрій Олександрович 45.55
		КНТ-118	лб	1	Андрєєв Максим Олександрович 57

Продовження таблиці А.1

1	2	3	4	5	6	7
13	Емпіричні методи в інформаційній технологіях	КНТ-128+119т	лб	1	Андрєєв Максим Олександрович	57
		КНТ-138+129т	лб	1	Андрєєв Максим Олександрович	57
14	Емпіричні методи в інформаційній технологіях	КНТ-218;228+219т	лк	1	Олійник Андрій Олександрович	45.55
		КНТ-218	лб	2	Леощенко Сергій Дмитрович	49
		КНТ-228+219т	лб	2	Федорченко Євген Миколайович	49
15	Програмування Інтернет-застосувань	КНТ-118;128+119т;138+129т	лк	1	Степаненко Олександр Олексійович	45.55
		КНТ-118	лб	1	Федорченко Євген Миколайович	56
		КНТ-128+119т	лб	1	Федорченко Євген Миколайович	56
		КНТ-138+129т	лб	1	Федорченко Євген Миколайович	56
16	Операційні системи	КНТ-118;128+119т;138+129т	лк	1	Сердюк Сергій Микитович	45.55
		КНТ-118	лб	3	Сердюк Сергій Микитович	56
		КНТ-128+119т	лб	3	Качан Олександр Іванович	56
		КНТ-138+129т	лб	3	Сердюк Сергій Микитович	56
17	Верифікація цифрових систем	КНТ-218;228+219т	лк	1	Табунщик Галина Володимирівна	45.55
		КНТ-218	лб	2	Каплієнко Тетяна Ігорівна	53а
		КНТ-228+219т	лб	2	Каплієнко Тетяна Ігорівна	53а
18	Вебпрограмування	КНТ-218;228+219т	лк	1	Скрупський Степан Юрійович	45.55
		КНТ-218	лб	2	Скрупський Степан Юрійович	58
		КНТ-228+219т	лб	2	Скрупський Степан Юрійович	58
19	Технології комп'ютерного програмування	КНТ-218;228+219т	лк	1	Пархоменко Анжеліка Володимирівна	45.55
		КНТ-218	лб	2	Андрєєв Максим Олександрович	57
		КНТ-228+219т	лб	2	Андрєєв Максим Олександрович	57

Продовження таблиці А.1

1	2	3	4	5	6	7
20	Операційні системи	КНТ-218;228+219т	лк	2	Сердюк Сергій Микитович	45.55
		КНТ-218	лб	2	Сердюк Сергій Микитович	56
		КНТ-228+219т	лб	2	Качан Олександр Іванович	56
21	Моделювання систем	КНТ-218;228+219т	лк	1	Сердюк Сергій Микитович	45.55
		КНТ-218	лб	2	Камінська Жанна Константинівна	49
		КНТ-228+219т	лб	2	Камінська Жанна Константинівна	49
22	Системи автоматизованого проектування	КНТ-218;228+219т	лк	1	Пархоменко Анжеліка Володимирівна	45.55
		КНТ-218	лб	1	Андрєєв Максим Олександрович	57
		КНТ-228+219т	лб	1	Андрєєв Максим Олександрович	57
23	Методи та алгоритми комп'ютерної графіки	КНТ-117;127+128т;137+118т	лк	2	Пархоменко Анжеліка Володимирівна	45.55
		КНТ-117	лб	2	Камінська Жанна Константинівна	49
		КНТ-127+128т	лб	2	Калініна Марина Володимирівна	57
		КНТ-137+118т	лб	2	Калініна Марина Володимирівна	57
24	Комп'ютерна графіка та візуалізація даних	КНТ-217;227+218т	лк	2	Пархоменко Анжеліка Володимирівна	45.55
		КНТ-217	лб	2	Камінська Жанна Константинівна	49
		КНТ-227	лб	2	Камінська Жанна Константинівна	49
25	Крос-платформне програмування	КНТ-117;127;137;217;227	лк	1	Олійник Андрій Олександрович	45.55
		КНТ-117	лб	2	Федорченко Євген Миколайович	58
		КНТ-127	лб	2	Федорченко Євген Миколайович	58
		КНТ-137	лб	2	Леоценко Сергій Дмитрович	58
		КНТ-217	лб	2	Федорченко Євген Миколайович	58
		КНТ-227	лб	2	Федорченко Євген Миколайович	58

Продовження таблиці А.1

1	2	3	4	5	6	7
26	Інтелектуальний аналіз даних	КНТ-118т;128т;218т	лк	1	Олійник Андрій Олександрович	45.55
		КНТ-118т	лб	2	Андрєєв Максим Олександрович	56
		КНТ-218т	лб	2	Андрєєв Максим Олександрович	56
		КНТ-128т	лб	2	Федорченко Євген Миколайович	56
27	Мобільні операційні системи та GPS	КНТ-117;127+128т;137+118т	лк	2	Степаненко Олександр Олексійович	45.55
		КНТ-117	лб	2	Федорченко Євген Миколайович	58
		КНТ-127+128т	лб	2	Федорченко Євген Миколайович	58
		КНТ-137+118т	лб	2	Федорченко Євген Миколайович	58
28	Мобільні операційні системи	КНТ-217;227+218т	лк	1	Степаненко Олександр Олексійович	45.55
		КНТ-217	лб	1	Камінська Жанна Констянтинівна	56
		КНТ-227+218т	лб	1	Камінська Жанна Констянтинівна	56
29	Системи автоматизованого проєктування	КНТ-117;127+128т;137+118т	лк	1	Пархоменко Анжеліка Володимирівна	45.55
		КНТ-117	лб	2	Андрєєв Максим Олександрович	57
		КНТ-127+128т	лб	2	Андрєєв Максим Олександрович	57
		КНТ-137+118т	лб	2	Андрєєв Максим Олександрович	57
30	Теорія прийняття рішень	КНТ-117;127+128т;137+118т	лк	2	Дубровін Валерій Іванович	45.55
		КНТ-117	лб	2	Андрєєв Максим Олександрович	56
		КНТ-127+128т	лб	2	Андрєєв Максим Олександрович	56
		КНТ-137+118т	лб	2	Андрєєв Максим Олександрович	56
31	Теорія прийняття рішень	КНТ-217;227+218т	лк	2	Дубровін Валерій Іванович	45.55
		КНТ-217	лб	1	Дейнега Лариса Юріївна	56
		КНТ-227+218т	лб	1	Дейнега Лариса Юріївна	56

Продовження таблиці А.1

1	2	3	4	5	6	7
32	Якість програмного забезпечення та тестування	КНТ- 117;127+128т;13 7+118т	лк	2	Табунщик Галина Володимирівна	45.55
		КНТ-117	лб	2	Каплієнко Тетяна Ігорівна	57
		КНТ-127+128т	лб	2	Каплієнко Тетяна Ігорівна	57
		КНТ-137+118т	лб	2	Каплієнко Тетяна Ігорівна	57
33	Проектуванн я інформаційн их систем	КНТ- 217;227+218т	лк	1	Табунщик Галина Володимирівна	45.55
		КНТ-217	лб	2	Табунщик Галина Володимирівна	53а
		КНТ-227+218т	лб	2	Табунщик Галина Володимирівна	53а
34	Технологія створення програмних продуктів	КНТ- 217;227+218т	лк	1	Олійник Андрій Олександрович	45.55
		КНТ-217	лб	2	Колпакова Тетяна Олексіївна	49
		КНТ-227+218т	лб	2	Федорченко Євген Миколайович	58
35	Технологія захисту інформації	КНТ- 217;227+218т	лк	1	Зайко Тетяна Анатоліївна	45.55
		КНТ-217	лб	2	Качан Олександр Іванович	58
		КНТ-227+218т	лб	2	Качан Олександр Іванович	58
36	Еволюційне програмуван ня	КНТ- 110м;120м;210м А+210мБ	лк	1	Олійник Андрій Олександрович	44.55
		КНТ-110м	лб	2	Федорченко Євген Миколайович	57
		КНТ-120м	лб	2	Леощенко Сергій Дмитрович	57
		КНТ- 210мА+210мБ	лб	2	Леощенко Сергій Дмитрович	57
37	Методологія наукових досліджень та академічне письмо	КНТ- 110м;120м;210м А+210мБ	лк	1	Дубровін Валерій Іванович	44.55
		КНТ-110м	лб	1	Дубровін Валерій Іванович	53а
		КНТ-120м	лб	1	Дубровін Валерій Іванович	53а
		КНТ- 210мА+210мБ	лб	1	Дубровін Валерій Іванович	53а

Продовження таблиці А.1

1	2	3	4	5	6	7
38	Мережні застосунки	КНТ-110м	лк	1	Степаненко Олександр Олексійович	45.55
		КНТ-110м	лб	1	Степаненко Олександр Олексійович	57
39	Scada-системи	КНТ-120м	лк	1	Зайко Тетяна Анатоліївна	45.55
		КНТ-120м	лб	1	Зайко Тетяна Анатоліївна	53а
40	Сертифікація програмного забезпечення	КНТ-110м;120м	лк	1	Льовкін Валерій Миколайович	44.55
		КНТ-110м	лб	1	Льовкін Валерій Миколайович	57
		КНТ-120м	лб	1	Льовкін Валерій Миколайович	57
41	CAD/CAM/CAE-системи	КНТ-120м;210мА+210мБ	лк	1	Пархоменко Анжеліка Володимирівна	45.55
		КНТ-120м	пр	1	Камінська Жанна Констянтинівна	53а
		КНТ-210мА+210мБ	пр	1	Камінська Жанна Констянтинівна	53а
		КНТ-120м	лб	2	Гладкова Ольга Миколаївна	57
		КНТ-210мА+210мБ	лб	2	Гладкова Ольга Миколаївна	57
42	Проектування та моделювання програмного забезпечення вбудованих систем	КНТ-110м	лк	1	Табунщик Галина Володимирівна	53а
		КНТ-110м	пр	1	Табунщик Галина Володимирівна	53а
		КНТ-110м	лб	2	Табунщик Галина Володимирівна	53а
43	Комп'ютерні методи обробки даних	КНТ-110м;120м;210мА+210мБ	лк	1	Дубровін Валерій Іванович	44.55
		КНТ-110м	лб	2	Дейнега Лариса Юріївна	58
		КНТ-120м	лб	2	Дейнега Лариса Юріївна	58
		КНТ-210мА+210мБ	лб	2	Дейнега Лариса Юріївна	58
44	Проектування та моделювання IoT систем	КНТ-210мА	лк	1	Табунщик Галина Володимирівна	25а,27
		КНТ-210мА	лб	2	Табунщик Галина Володимирівна	53а

Кінець таблиці А.1

1	2	3	4	5	6	7
45	Теорія обчислюваль ного навчання	КНТ-210мБ	лк	1	Миронова Наталя Олексіївна	45.55
		КНТ-210мБ	лб	2	Миронова Наталя Олексіївна	53а
46	Мережні застосунки	КНТ-210мА	лк	1	Степаненко Олександр Олексійович	45.55
		КНТ-210мА	лб	1	Степаненко Олександр Олексійович	57
47	Кіберфізичні системи	КНТ-210мБ	лк	1	Пархоменко Анжеліка Володимирівна	45.55
		КНТ-210мБ	лб	1	Туленков Артем Вячеславович	53а
48	Індуктивні методи автоматизов аного навчання та пізнання	КНТ-А9	лк	1	Субботін Сергій Олександрович	25а,27
		КНТ-А9	пр	1	Субботін Сергій Олександрович	25а,27
49	Інформаційн і технології у науовій діяльності	КНТ-А9	лк	1	Субботін Сергій Олександрович	25а,27
		КНТ-А9	пр	1	Субботін Сергій Олександрович	25а,27
50	Еволюційні та гібридні інформаційні системи	КНТ-А8	лк	1	Олійник Андрій Олександрович	25а,27
		КНТ-А8	пр	2	Олійник Андрій Олександрович	25а,27
51	Геоінформац ійні системи	КНТ- 117;127+128т;13 7+118т;217;227+ 218т	лк	2	Степаненко Олександр Олексійович	45.55
		КНТ-117	лб	2	Федорченко Євген Миколайович	58
		КНТ-127+128т	лб	2	Федорченко Євген Миколайович	58
		КНТ-137+118т	лб	2	Федорченко Євген Миколайович	58
		КНТ-217	лб	1	Камінська Жанна Констянтинівна	45.55
		КНТ-227+218т	лб	1	Камінська Жанна Констянтинівна	56

ДОДАТОК Б
ТЕКСТ ПРОГРАМИ

Б.1 Фрагмент файла app.js

```
const main = async () => {
  const app = express();
  let corsOptions = {
    origin: "*",
    credentials: true,
  };
  app.use(cors(corsOptions));

  await db.Connection.sync({})
    .then(async (result) => {
      await InitRecords(db);
      console.log("Connected to DB");
    })
    .catch((err) => console.log(err));

  app.use(bodyParser.urlencoded({ extended: true }));
  app.use(bodyParser.json());
  app.use(
    "/graphql",
    graphqlHTTP({
      schema: Schema,
      graphiql: true,
    })
  );

  http.createServer(app).listen(config.PORT, () => {
    console.log(`Server HTTP is running on ${config.PORT}`);
  });
};

main().catch((err) => {
  console.log(err);
});
```

Б.2 Фрагмент файла database.js

```
const db = {};

const models = [
  Teacher,
  Cathedra,
  Audience,
  Assigned_audience,
  Specialty,
  Type_class,
  Group,
  Discipline,
  Assigned_discipline,
  Class,
  Recommended_audience,
  Assigned_teacher,
  Schedule,
  Assigned_group,
  User,
  Info,
  Recommended_schedule,
  Algorithm,
```

```

    Results_algorithm,
  ];
  let Connection = new Sequelize(config.DB_NAME, config.DB_USER,
config.DB_PASS, {
    dialect: "mysql",
    host: config.HOST,
    define: { timestamps: false },
  });
  await Connection.sync({});

models.forEach((model) => {
  const seqModel = model(Connection, Sequelize);
  console.log(seqModel.name);
  db[seqModel.name] = seqModel;
});

Object.keys(db).forEach((key) => {
  if ("associate" in db[key]) {
    db[key].associate(db);
  }
});

db.Connection = Connection;
db.Sequelize = Sequelize;

export default db;

```

Б.3 Фрагмент файла Algorithms/SimpleAlgorithmCPP/main.cpp

```

int main(int argc, char* argv[])
{
  try
  {
    auto StartTime = chrono::high_resolution_clock::now();
    string path;
    if (argc == 2) {
      path = argv[1];
    }
    else {
      path = filesystem::current_path().string();
    }
    srand(time(NULL));
    json data = json();
    ifstream fileData(path + "\\data.json");
    data = json::parse(fileData);
    auto result = vector<pair<int, double>>();

    auto SA = SimpleAlgorithm(data);

    SA.Fitness(0);

    auto best_individ = SA.GetBestIndivid();

    auto EndTime = chrono::high_resolution_clock::now();
    chrono::duration<float, std::milli> duration = EndTime -
StartTime;
    result.push_back(make_pair(duration.count(),
best_individ.fitnessValue.fitnessValue));

```

```

        cout << "Simple Algorithm: " << " fitness: " <<
best_individ.fitnessValue.fitnessValue << endl;

        Service::SaveResults(best_individ, path, result);
    }
    catch (exception &ex)
    {
        cout << ex.what() << endl;
    }
    catch (...)
    {
        cout << "any mistake" << endl;
    }

    return 0;
}

```

Б.4**Фрагмент****файлу****Algorithms/SimulatedAnnealingAlgorithmCPP/main.cpp**

```

int main(int argc, char* argv[])
{
    try
    {
        string path;
        string pathToSA;
        if (argc >= 2){
            path = argv[1];
            if (argc == 3){
                pathToSA = argv[2];
            }
        }
        else {
            path = filesystem::current_path().string();
            pathToSA =
            "..\\SimpleAlgorithmCPP\\SimpleAlgorithmCPP.exe";
        }
        srand(time(NULL));
        json data = json();
        ifstream fileData(path + "\\data.json");
        data = json::parse(fileData);

        json data_SA = json();

        if (data["params"]["type_initialization"] ==
"simple_algorithm"){
            auto code = system(string(pathToSA + " " + path).c_str());
            if (code == 0){
                ifstream fileData(path + "\\result.json");
                data_SA = json::parse(fileData);
            }
            else{
                throw "Error run SimpleAlgorithm.exe";
            }
        }

        int countIter = 0;
        auto best_individ = bestIndivid();
    }
}

```

```

auto result = vector<pair<int, double>>();
auto StartTime = chrono::high_resolution_clock::now();

SimulatedAnnealing mainAlgorithm(data);
mainAlgorithm.InitBaseSchedule(data);
mainAlgorithm.InitClasses(0, data_SA);
mainAlgorithm.InitPopulations(0);

const int max_generations = data["params"]["max_generations"];
while (countIter < max_generations &&
mainAlgorithm.populations[0].fitnessValue.fitnessValue != 0) {
    mainAlgorithm.ClearIndivid(1);
    mainAlgorithm.SetIndivid(1, 0);
    mainAlgorithm.InitPopulations(1);
    mainAlgorithm.Mutation(1);
    mainAlgorithm.Fitness(0);
    auto fitness =
mainAlgorithm.populations[0].fitnessValue.fitnessValue;

    mainAlgorithm.Fitness(1);
    auto new_fitness =
mainAlgorithm.populations[1].fitnessValue.fitnessValue;

    auto difference = new_fitness - fitness;
    if (difference < 0){
        mainAlgorithm.ClearIndivid(0);
        mainAlgorithm.SetIndivid(0, 1);
        mainAlgorithm.InitPopulations(0);
    } else{
        if (GetRndDouble() < exp(-difference /
mainAlgorithm.temperature)) {
            mainAlgorithm.ClearIndivid(0);
            mainAlgorithm.SetIndivid(0, 1);
            mainAlgorithm.InitPopulations(0);
        }
    }

    mainAlgorithm.UpdateTemperature();
    ++countIter;
    if (countIter % 100 == 0){
        cout << "Iter: " << countIter << " , fitness: " <<
mainAlgorithm.populations[0].fitnessValue.fitnessValue << " , temp: " <<
mainAlgorithm.temperature << endl;
        auto EndTime = chrono::high_resolution_clock::now();
        chrono::duration<float, std::milli> duration = EndTime
- StartTime;
        result.push_back(make_pair(duration.count(),
mainAlgorithm.populations[0].fitnessValue.fitnessValue));
    }
    best_individ = mainAlgorithm.GetBestIndivid();
    Service::SaveResults(best_individ, path, result);
}
catch (exception &ex)
{
    cout << ex.what() << endl;
}
catch (...)
{
    cout << "any mistake" << endl;
}

return 0;

```

```
}
```

Б.5 Фрагмент файла Algorithms/EvolutionAlgorithmCPP/main.cpp

```
int main(int argc, char* argv[])
{
    try
    {
        string path;
        string pathToSA;
        if (argc >= 2) {
            path = argv[1];
            if (argc == 3) {
                pathToSA = argv[2];
            }
        }
        else {
            path = filesystem::current_path().string();
            pathToSA =
            "..\\SimpleAlgorithmCPP\\SimpleAlgorithmCPP.exe";
        }
        srand(time(NULL));
        json data = json();
        ifstream fileData(path + "\\data.json");
        data = json::parse(fileData);

        const int max_generations = data["params"]["max_generations"];
        thread_pool worker_pool(thread::hardware_concurrency());
        timer Timer;
        Timer.start();

        json data_SA = NULL;
        if (data["params"]["type_initialization"] ==
        "simple_algorithm") {
            auto code = system(string(pathToSA + " " + path).c_str());
            if (code == 0) {

                ifstream fileData(path + "\\result.json");
                data_SA = json::parse(fileData);
            }
            else {
                throw "Error run SimpleAlgorithm.exe";
            }
        }
        EvolutionAlgorithm mainAlgorithm(data, data_SA);

        Timer.stop();
        cout << "Init " << Timer.ms() << " ms.\n";

        int countIter = 0;
        auto best_individ = bestIndivid();
        auto result = vector<pair<int, double>>();
        auto StartTime = chrono::high_resolution_clock::now();

        while (countIter < max_generations &&
        best_individ.fitnessValue.fitnessValue != 0)
        {
            Timer.start();
```

```

mainAlgorithm.CrossingLoop(worker_pool);

Timer.stop();

cout << "Crossing " << Timer.ms() << "ms" << endl;

Timer.start();

mainAlgorithm.MutationLoop(worker_pool);

Timer.stop();

cout << "Mutation " << Timer.ms() << "ms" << endl;

Timer.start();

mainAlgorithm.FitnessLoop(worker_pool);

Timer.stop();

cout << "Fitness " << Timer.ms() << "ms" << endl;

Timer.start();

mainAlgorithm.SelectionLoop(worker_pool);

Timer.stop();

cout << "Selection " << Timer.ms() << "ms" << endl;

best_individ = mainAlgorithm.GetBestIndivid();

    cout << "Iter: " << ++countIter << ", Min fitness: " <<
best_individ.fitnessValue.fitnessValue << ", Mean fitness: " <<
mainAlgorithm.GetMeanFitnessValue() << endl;
    auto EndTime = chrono::high_resolution_clock::now();
    chrono::duration<float, std::milli> duration = EndTime -
StartTime;
    result.push_back(make_pair(duration.count(),
best_individ.fitnessValue.fitnessValue));
}
    Service::SaveResults(best_individ, path, result);
}
catch (exception &ex)
{
    cout << ex.what() << endl;
}
catch (...)
{
    cout << "any mistake" << endl;
}

return 0;
}

```

Б.6**Фрагмент****файлу****Algorithms/IslandModelEvolutionAlgorithmCPP/main.cpp**

```

int main(int argc, char* argv[])
{
    try
    {
        string path;
        string pathToSA;
        if (argc >= 2){
            path = argv[1];
            if (argc == 3){
                pathToSA = argv[2];
            }
        }
        else {
            path = filesystem::current_path().string();
            pathToSA =
            "..\\SimpleAlgorithmCPP\\SimpleAlgorithmCPP.exe";
        }
        srand(time(NULL));
        json data = json();
        ifstream fileData(path + "\\data.json");
        data = json::parse(fileData);

        int number_island = data["params"]["number_islands"];
        double step = data["params"]["step"];
        const int max_generations = data["params"]["max_generations"];
        const int population_size = data["params"]["population_size"];
        const double n_migration = data["params"]["n_migration"];
        const int len_migration = population_size * n_migration;
        const int iter_migration = data["params"]["iter_migration"];

        thread_pool worker_pool(thread::hardware_concurrency());
        timer Timer;
        Timer.start();

        double base_p_crossover =
        (double) data["params"]["p_crossover"];
        double base_p_mutation = (double) data["params"]["p_mutation"];
        double base_p_elitism = (double) data["params"]["p_elitism"];
        json data_SA = json();
        if (data["params"]["type_initialization"] ==
        "simple_algorithm"){
            auto code = system(string(pathToSA + " " + path).c_str());
            if (code == 0){
                ifstream fileData(path + "\\result.json");
                data_SA = json::parse(fileData);
            }
            else{
                throw "Error run SimpleAlgorithm.exe";
            }
        }
        vector<IslandModelEvolutionAlgorithm> islands;
        for (int i = 0; i < number_island; i++){
            if(i != 0) {
                double value_p_crossover = (i * step *
                base_p_crossover) / 100;
                double value_p_mutation = (i * step * base_p_mutation)
                / 100;
                double value_p_elitism = (i * step * base_p_elitism)
                / 100;
                data["params"]["p_crossover"] = base_p_crossover +
                value_p_crossover;
                data["params"]["p_mutation"] = base_p_mutation +
                value_p_mutation;
            }
        }
    }
}

```

```

        data["params"]["p_elitism"]      =      base_p_elitism      +
value_p_elitism;
    }
    islands.push_back(IslandModelEvolutionAlgorithm(data,
data_SA));
}

Timer.stop();
cout << "Init " << Timer.ms() << "ms" << endl;

int countIter = 0;
auto best_individ = bestIndivid();
auto result = vector<pair<int, double>>();
auto StartTime = chrono::high_resolution_clock::now();

while      (countIter      <      max_generations      &&
best_individ.fitnessValue.fitnessValue != 0)
{
    Timer.start();
    for(int i =0;i<number_island;i++){
        islands[i].CrossingLoop(worker_pool);
    }
    Timer.stop();
    cout << "Crossing " << Timer.ms() << "ms" << endl;

    Timer.start();
    for(int i =0;i<number_island;i++){
        islands[i].MutationLoop(worker_pool);
    }
    Timer.stop();
    cout << "Mutation " << Timer.ms() << "ms" << endl;

    Timer.start();
    for(int i =0;i<number_island;i++){
        islands[i].FitnessLoop(worker_pool);
    }
    Timer.stop();
    cout << "Fitness " << Timer.ms() << "ms" << endl;

    Timer.start();
    for(auto &island: islands){
        island.SelectionLoop(worker_pool);
    }

    Timer.stop();
    cout << "Selection " << Timer.ms() << "ms" << endl;

    if((countIter + 1) % iter_migration == 0){
        Timer.start();
        auto      islandsBestIndivids      =
vector<vector<pair<vector<vector<schedule>>,int>>>());

        for(auto &island : islands){

islandsBestIndivids.push_back(island.GetBestIndivids(len_migration));
        }
        int best_index = -1;
        double sum = DBL_MAX;
        for(auto i = 0; i < number_island; i++){
            double tempSum = 0;
            for(auto &ind: islandsBestIndivids[i]){
                tempSum += ind.second;
            }

```

```

        if(sum > tempSum){
            best_index = i;
            sum = tempSum;
        }
    }
    for(auto i = 0; i < number_island; i++){
        if(i != best_index) {
islands[i].ChangeWorstIndivids(islandsBestIndivids[best_index]);
        }
    }
    Timer.stop();
    cout << "Migration " << Timer.ms() << "ms" << endl;
}
for(auto &island: islands){
    auto bestInIsland = island.GetBestIndivid();
    if(best_individ.fitnessValue.fitnessValue >
bestInIsland.fitnessValue.fitnessValue){
        best_individ = bestInIsland;
    }
}

    cout << "Iter: " << ++countIter << ", Min fitness: " <<
best_individ.fitnessValue.fitnessValue << endl;
    auto EndTime = chrono::high_resolution_clock::now();
    chrono::duration<float, std::milli> duration = EndTime -
StartTime;
    result.push_back(make_pair(duration.count(),
best_individ.fitnessValue.fitnessValue));

}
Service::SaveResults(best_individ, path, result);
}
catch (exception &ex)
{
    cout << ex.what() << endl;
}
catch (...)
{
    cout << "any mistake" << endl;
}

return 0;
}

```

Б.7**Фрагмент****файлу****Algorithms/TabuSearchAlgorithmCPP/main.cpp**

```

int main(int argc, char* argv[])
{
    try
    {
        string path;
        string pathToSA;
        if (argc >= 2){
            path = argv[1];
            if (argc == 3){
                pathToSA = argv[2];

```

```

    }
}
else {
    path = filesystem::current_path().string();
    pathToSA
    =
    "..\\SimpleAlgorithmCPP\\SimpleAlgorithmCPP.exe";
    }
    srand(time(NULL));
    json data = json();
    ifstream fileData(path + "\\data.json");
    data = json::parse(fileData);
    json data_SA=NULL;
    TabuSearch mainAlgorithm;
    if
        (data["params"]["type_initialization"]
        ==
        "simple_algorithm"){
        auto code = system(string(pathToSA + " " + path).c_str());
        if (code == 0){
            data_SA = json();
            ifstream fileData(path + "\\result.json");
            data_SA = json::parse(fileData);
        }
        else{
            throw "Error run SimpleAlgorithm.exe";
        }
    }
    timer Timer;
    Timer.start();
    mainAlgorithm = TabuSearch(data,Timer,data_SA);
    auto result = vector<pair<int, double>>();
    auto best_individ = mainAlgorithm.MainLoop(data,Timer,result);
    Service::SaveResults(best_individ, path, result);

}
catch (exception &ex)
{
    cout << ex.what() << endl;
}
catch (...)
{
    cout << "any mistake" << endl;
}

return 0;
}

```

ДОДАТОК В
СЛАЙДИ ПРЕЗЕНТАЦІЇ

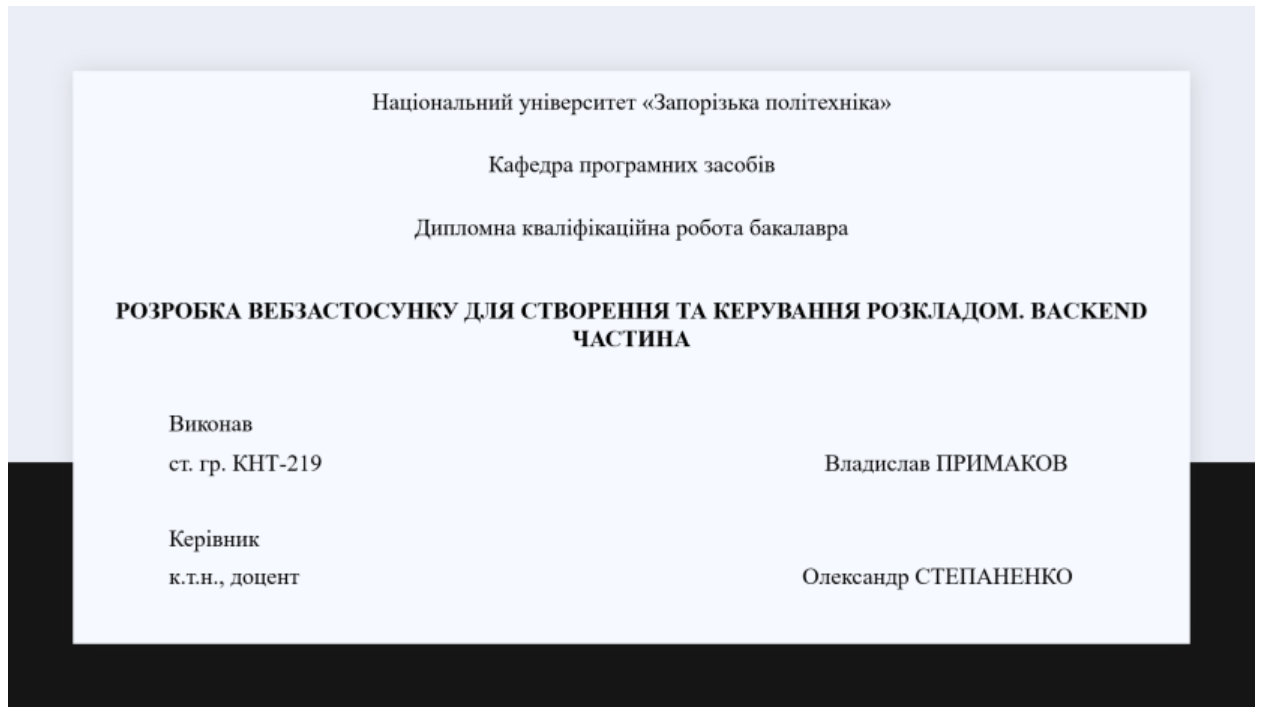


Рисунок В.1 – Слайд № 1

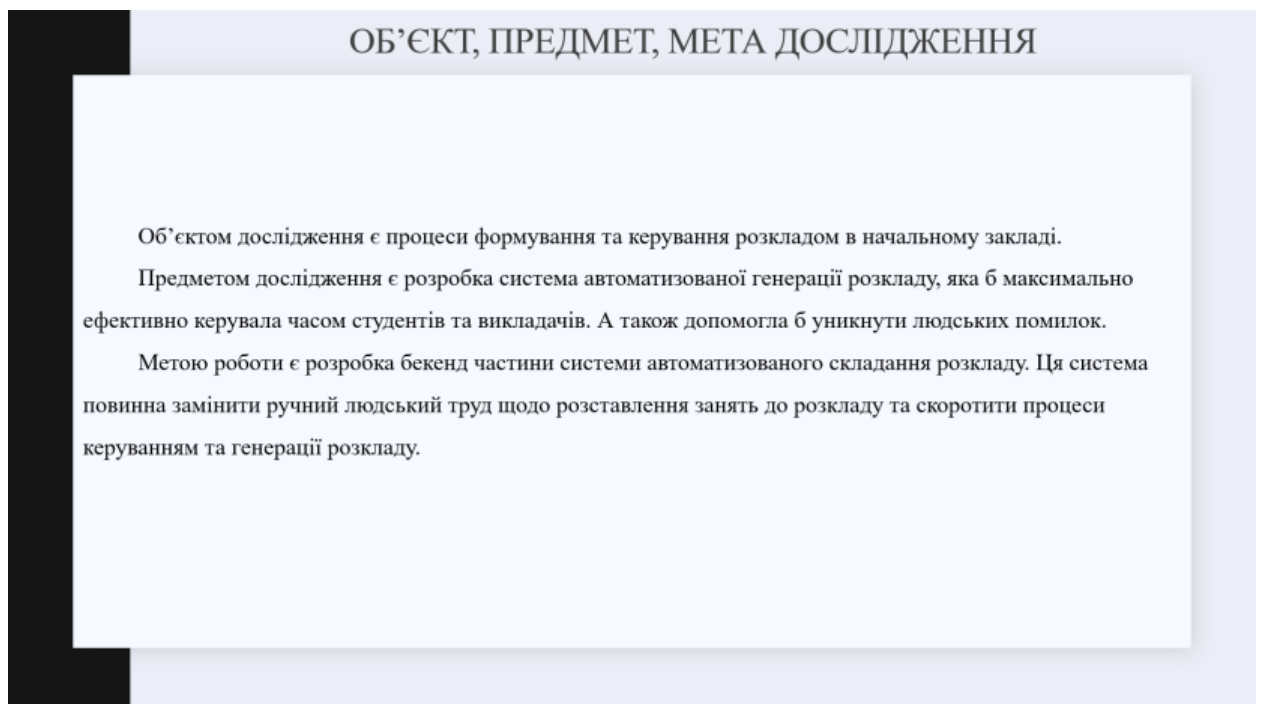


Рисунок В.2 – Слайд № 2

ОГЛЯД НАЯВНИХ СИСТЕМ			
Характеристика	Untis	TimeTabler	TimeEdit
Відкритий код	Ні	Ні	Ні
Операційна система	Кросплатформна	Кросплатформна	Кросплатформна
Підтримка різних мов	Багатомовна	Багатомовна	Багатомовна
Наявність ліцензії	Так	Так	Так
Адаптивність до різних типів установ	Так	Так	Так
Тип програми	Десктопна	Десктопна	Вебзастосунок
Автоматичне створення розкладу	Так	Так	Так
Формат збереження даних	.unt, .xlsx, .xml	.ttf або .ttb, .xlsx, .xml	.xml, .csv, sql
Ручне редагування розкладу	Так	Так	Так
Редагування даних	Так	Так	Так

Рисунок В.3 – Слайд № 3

ПОСТАНОВКА ЗАВДАННЯ ДО РОБОТИ
Метою роботи є дослідження та розробка програмного забезпечення для генерації розкладу за допомогою різноманітних алгоритмів та керування їм. Вхідними даними для системи будуть: дані про заняття, викладачів, групи, аудиторії, дисципліни, спеціальності, кафедри, параметри запуску алгоритмів для генерації розкладу, загальні дані такі як кількість днів, кількість пар та штрафні значення. Вихідними даними буде згенерований розклад певним алгоритмом який мінімізує вікна між парами для груп та викладачів та накладки занять, а також врахує побажання щодо часу проведення занять. До системи висувають такі вимоги: система зберігання даних, вебсервер для обробки запитів для керування даними, система генерації розкладу, система конвертації розкладу з відомостей, система контролю стану розкладу.

Рисунок В.4 – Слайд № 4

ВИБІР МОВИ ПРОГРАМУВАННЯ ДЛЯ ВЕБСЕРВЕРА			
Характеристика	JavaScript	Python	C#
Складність синтаксису	Середня	Проста	Складна
Швидкість виконання	Помірна	Повільна	Висока
Екосистема та бібліотеки	Широкий спектр фреймворків	Широкий спектр фреймворків	Пов'язана з платформою .NET
Система типів	Слаба типізація	Сильна типізація	Сильна типізація
ООП	Прототипи і функції-конструктори	Повноцінне	Повноцінне
Багатопочіність	Асинхронне програмування	Вбудована підтримка	Вбудована підтримка
Підтримка платформ	Більшість вебплатформ	Різні ОС	Windows, інші з .NET Core
Безпека	Відкритий доступ до клієнтського коду	Вбудовані заходи безпеки	Потужні механізми безпеки з .NET

Рисунок В.5 – Слайд № 5

ВИБІР ФРЕЙМВОРКУ ДЛЯ ВЕБСЕРВЕРА						
Характеристика	ExpressJS	NestJS	Jango	Flask	.NET Framework	.NET Core
Мова програмування	JS	JS	Python	Python	C#	C#
Функціональність	Легкість використання	Повна підтримка TS та структурована архітектура	Повна функціональність для розробки вебдодатків	Мінімалістичний фреймворк з простотою використання	Повна платформа для розробки вебдодатків	Повна платформа для розробки вебдодатків
Спільнота та підтримка	Велика спільнота та активне оновлення	Швидко зростаюча спільнота	Велика та досвідчена спільнота та регулярні оновлення	Зростаюча спільнота та регулярні оновлення	Велика спільнота в середовищі Microsoft	Швидко зростаюча спільнота
Швидкодія та продуктивність	Висока продуктивність особливо при роботі зі статичними файлами	Залежить від ефективного використання асинхронного програмування	Швидкодія повільніше за інші фреймворки	Швидкодія повільніше за інші фреймворки	Добре оптимізований JIT-компілятор	Використовує той самий JIT-компілятор
Безпека	Не має вбудованих механізмів безпеки	Має вбудовану підтримку	Вбудовані інструменти	Разні пакети Flask-WTF Flask-Security	Широкий спектр інструментів	Покращена безпека

Рисунок В.6 – Слайд № 6

ВИБІР АРХІТЕКТУРИ ВЕБЗАСТОСУНКУ

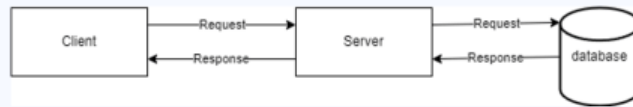


Схема 3-рівневої архітектури

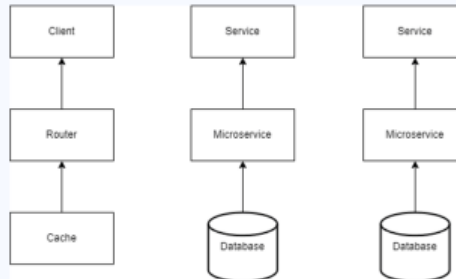


Схема мікросервісної архітектури

Рисунок В.7 – Слайд № 7

ВИБІР АРХІТЕКТУРИ ВЕБЗАСТОСУНКУ

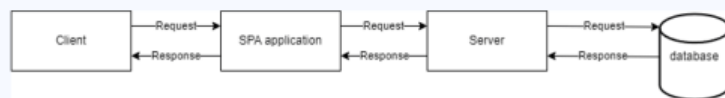


Схема SPA архітектури



Схема PWA архітектури



Схема GraphQL архітектури



Загальна схема архітектур вебзастосунку

Рисунок В.8 – Слайд № 8

ВИБІР ШАБЛОНУ ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ

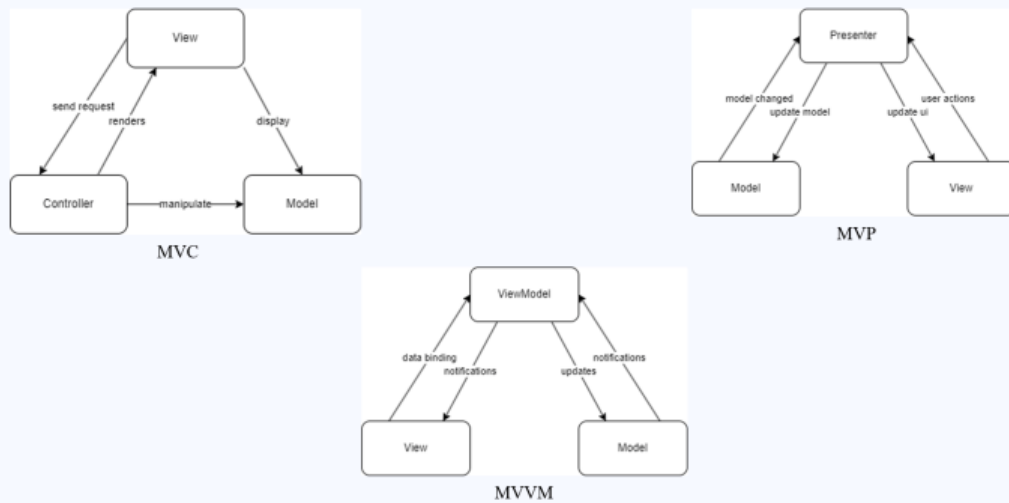


Рисунок В.9 – Слайд № 9

ВИБІР БАЗИ ДАНИХ ДЛЯ ВЕБЗАСТОСУНКУ

Характеристика	MySQL	PostgreSQL	MongoDB
Тип	Реляційна	Об'єктно-реляційна	Документоорієнтовна
Мова запитів	SQL	SQL, PL/pgSQL	JSON-подібна
Масштабованість	Вертикальна і горизонтальна	Вертикальна і горизонтальна	Горизонтальна
Підтримка операцій	Широкий набір	Надає розширенні можливості включаючи rollback та savepoints	Широкий набір
Підтримка інструментів	Розширена підтримка для адміністрування, розробки, моніторингу та налагодження	Розширена підтримка для адміністрування, розробки, моніторингу та налагодження	Власний набір інструментів для роботи який надає графічний інтерфейс

Рисунок В.10 – Слайд № 10

СХЕМА БАЗИ ДАНИХ

The diagram illustrates a database schema for a recommendation system. It consists of 15 tables, each with its own set of fields and primary/foreign keys. The tables are interconnected by dashed lines representing relationships.

- groups**: id INT (PK), name VARCHAR(255), num_bot_students INT, semester INT, id_specialty INT (FK).
- assigned_groups**: id INT (PK), id_class INT (FK), id_group INT (FK), teachers (FK).
- specifications**: id INT (PK), name VARCHAR(255), code INT, id_audience INT (FK).
- assigned_descriptions**: id INT (PK), id_specialty INT (FK), id_discipline INT (FK), semester INT (FK).
- disciplines**: id INT (PK), name VARCHAR(255), teachers (FK).
- cathedras**: id INT (PK), name VARCHAR(255), chair_name VARCHAR(255), teachers (FK).
- teachers**: id INT (PK), name VARCHAR(255), patronymic VARCHAR(255), id_cathedra INT (FK).
- assigned_teachers**: id INT (PK), id_class INT (FK), id_teacher INT (FK), teachers (FK).
- classes**: id INT (PK), id_name_class INT (FK), id_assigned_discipline INT (FK), id_time_perk DOUBLE, id_day_perk INT, id_class INT (FK).
- recommended_schedules**: id INT (PK), num_bot_perk INT, id_day_perk INT, id_class INT (FK).
- assigned_audiences**: id INT (PK), id_audience INT (FK), id_class INT (FK).
- audiences**: id INT (PK), name VARCHAR(255), id_type_class INT (FK), capacity SMALLINT, teachers (FK).
- type_classes**: id INT (PK), name VARCHAR(10), teachers (FK).
- info**: id INT (PK), num_day INT, num_perk INT, num_semester INT, fitness_value VARCHAR(255), general_value VARCHAR(255), teachers (FK).
- users**: id INT (PK), email VARCHAR(255), password VARCHAR(255), accessToken VARCHAR(255), teachers (FK).
- recommended_audiences**: id INT (PK), id_audience INT (FK), id_class INT (FK).
- schedules**: id INT (PK), num_bot_perk INT, id_day_perk INT, id_perk INT, id_class INT (FK), id_audience INT (FK).
- algorithms**: id INT (PK), name VARCHAR(255), label VARCHAR(255), id_params VARCHAR(2000), teachers (FK).
- results_algorithms**: id INT (PK), param_value VARCHAR(255), id_name_algorithm VARCHAR(255), results LONGTEXT, teachers (FK).

Рисунок В.12 – Слайд № 12

ПОБУДОВА МОДЕЛЕЙ SEQUELIZE ТА ПІДКЛЮЧЕННЯ ДО БД

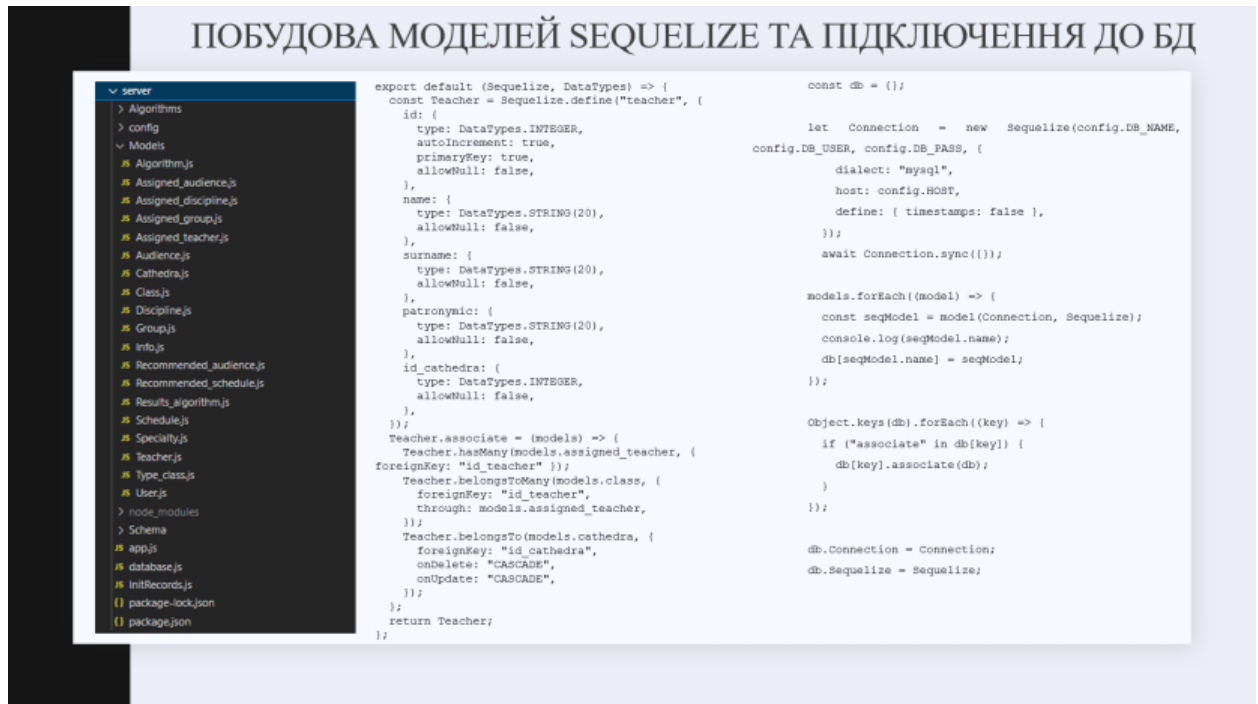


Рисунок В.13 – Слайд № 13

ПОБУДОВА МОДЕЛЕЙ GRAPHQL

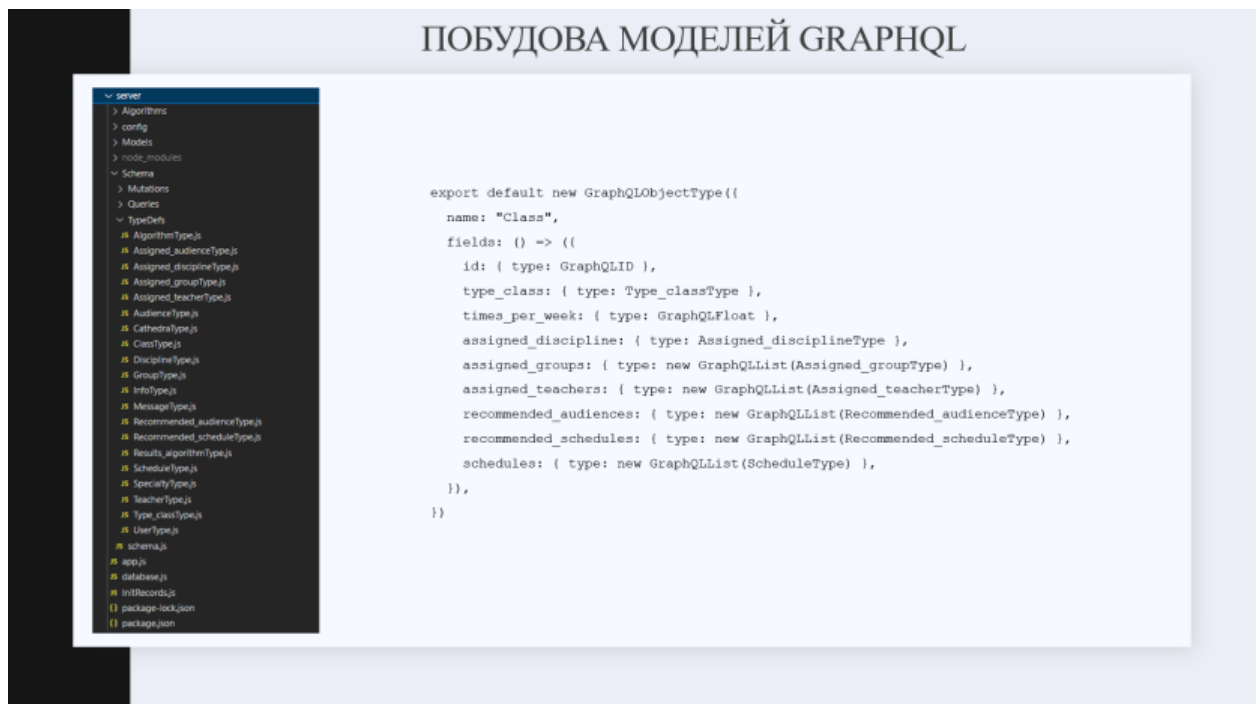


Рисунок В.14 – Слайд № 14

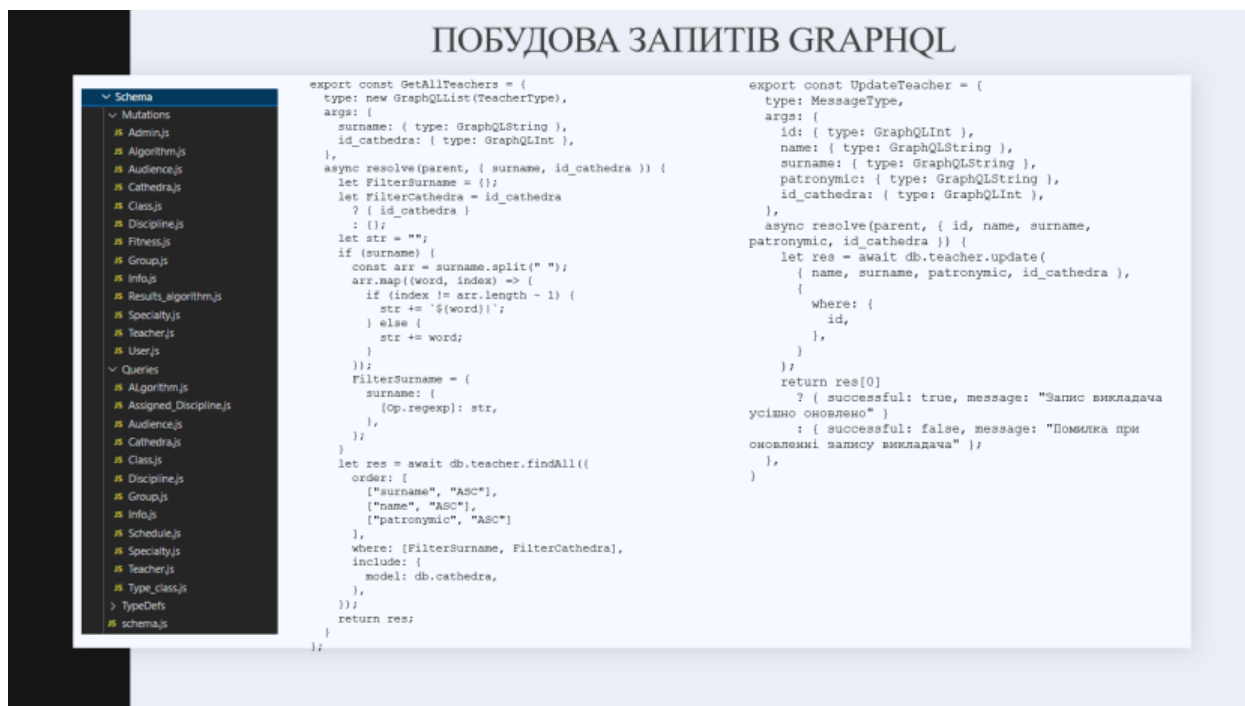


Рисунок В.15 – Слайд № 15



Рисунок В.16 – Слайд № 16

ТЕСТУВАННЯ АЛГОРИТМІВ ЗАГАЛЬНІ ДАНІ	
Характеристика	Значення
Процесор	Ryzen 5 2600
Частота процесора	3.9 ГГц
Кількість ядер	6
Кількість потоків	6
Обсяг оперативної пам'яті	8 ГБ
Швидкість оперативної пам'яті	3200 МГц
Тип жорсткого диску	SSD
Операційна система	Virtual Machine Windows Server 2022
Кількість днів	6
Кількість пар	6
Штраф за вікно у груп	1
Штраф за вікно у викладача	1
Штраф за накладання заняття	5
Штраф за невідповідність рекомендованого часу проведення заняття	5

$$S = \frac{|F_s - F_f|}{|t_f - t_s|},$$

де S – швидкість збіжності;

F_s – початкове значення фітнес;

F_f – кінцеве значення фітнес;

t_s – початковий час роботи алгоритму;

t_f – кінцевий час.

Рисунок В.17 – Слайд № 17

РЕЗУЛЬТАТИ ТЕСТУВАННЯ ПАРАМЕТРІВ АЛГОРИТМІВ		
Назва алгоритма	Параметр	Значення
Генетичний алгоритм	ймовірність мутації	0.4
	тип мутації	мутація одного гена
	ймовірність схрещування	0.8 (значення дуже близькі між собою)
	значення елітизму	0.2
	тип вибірки	рулетка (значення дуже близькі між собою)
	тип схрещування	схрещування одного гена
Модифікація генетичного алгоритму – острівна модель	тип ініціалізації	за допомогою алгоритму простого перебору
	кількість островів	6 (кількість логічних ядер процесору)
	крок розбіжності параметрів	3
	через скільки ітерацій проводити міграцію	5
Табу пошук	кількість індивідів для участі в міграції	0.05 (значення дуже близькі між собою)
	тип мутації	мутація одного гену
	кількість ітерацій дії табу	30
	кількість сусідніх розв'язків	30
	довжина списку табу	100
Імітація відпаду	тип ініціалізації	алгоритм простого перебору
	тип мутації	мутація одного гена
	температура	краще та, яка має менше значення 10^5
	коефіцієнт альфа	0.8
	тип ініціалізації	алгоритм простого перебору

Рисунок В.18 – Слайд № 18

ТЕСТУВАННЯ АЛГОРИТМІВ З ПІДБРАНИМИ НАЙКРАЩИМИ ПАРАМЕТРАМИ						
Алгоритм	№	T _с , с	T _р , с	F _с	F _і	S
Імітація відпалу	1	0,023	68,1	40	0	0,59
	2	0,037	175	15	0	0,09
	3	0,05	47	26	0	0,55
	Сер.	0,37	96,7	27	0	0,41
Генетичний алгоритм	4	1	1290	14	0	0,011
	5	1	734	10	0	0,013
	6	1	913	6	0	0,007
	Сер.	1	979	10	0	0,01
Острівна модель	7	1	113	6	0	0,054
	8	1	256	12	0	0,047
	9	1	439	8	0	0,018
	Сер.	1	269,3	8,6	0	0,04
Табу пошук	10	0,069	395	2	2	0
	11	0,1	424	12	4	0,019
	12	0,082	406	8	4	0,01
	Сер.	0,084	408,3	7,3	3,3	0,01

Рисунок В.19 – Слайд № 19

ВИСНОВОК

В процесі виконання кваліфікованої роботи бакалавра було розроблено вебсервер для системи автоматизованого складання та керування розкладу для занять в навчальних закладах. А також розроблені та протестовані алгоритми для генерації розкладу: алгоритм простого перебору, імітації відпалу, табу пошук, генетичний алгоритм, модифікація генетичного алгоритму – острівна модель.

Вебзастосунок призначений для ефективного керування розкладом в навчальному закладі. Він призначений замінити ручний труд розставлення пар у навчальному закладі на точний машинний розрахунок та униканню помилок.

Було розроблено власно створений алгоритм ініціалізації початкового розкладу – алгоритм простого перебору. Для конкретної задачі генерації розкладу (Додаток А.1) було виявлено, що найкраще справляється з генерацією розкладу – алгоритм імітації відпалу як по середньому часу роботи так і по швидкості збіжності рішення

Рисунок В.20 – Слайд № 20