ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «МІЖРЕГІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ ПЕРСОНАЛОМ»»

Факультет комп'ютерно-інформаційних технологій

Кафедра комп'ютерних інформаційних систем і технологій

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Тема: «Розробка системи автоматизованого формування розкладу занять для студентів навчальних закладів»

Виконав студент групи ІК-9-22-Б1ІПЗ(4.0д) Снєжков О.С.

Керівник: викладач Яременко Д.М.

Допущено до захисту

Завідувач кафедри _____ д.е.н., професор Кавун С.В.

(підпис)

Київ, 2026

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної роботи: 67 с., 25 рис., 3 додатки, 35 джерел.

РОЗРОБКА СИСТЕМИ АВТОМАТИЗОВАНОГО ФОРМУВАННЯ РОЗКЛАДУ ЗАНЯТЬ ДЛЯ СТУДЕНТІВ ТА ВИКЛАДАЧІВ, PYTHON, FLASK, GIT, POSTMAN, API, MYSQL.

Обєктом дослідження є системи автоматичного генерування розпорядків дня, зустрічей і розкладів для порівняння алгоритмів, логіки та реалізації на різних мовах.

Метою роботи є створення програми для генерування розкладу враховуючи низку факторів таких як: бажаний час проведення пар, залежність предмету від аудиторії, складність предмету і.т.д.

Методи розробки базуються на використанні мови програмування Python з використанням фреймворку для веб застосунків Flask і бази данних MySQL.

Результатом розробки є функціональна частина застосунку яка може враховувати низку факторів. Дана система дозволяє генерувати комплексні розклади підлаштовуючись під людьский фактор.

DEVELOPMENT OF AN AUTOMATED CLASS SCHEDULING SYSTEM FOR STUDENTS AND TEACHERS, PYTHON, FLASK, GIT, POSTMAN, API, MYSQL.

The object of the study is automated systems for generating daily schedules, meetings, and timetables, with the aim of comparing algorithms, logic, and implementations across different programming languages.

The purpose of the work is to create a program for generating schedules while considering a number of factors such as: preferred class times, dependency of subjects on classrooms, subject complexity, etc.

The development methods are based on the use of the Python programming language, the Flask web application framework, and the MySQL database.

The result of the development is a functional part of an application that can take into account a range of factors. This system enables the generation of complex schedules while adapting to human-related constraints.

ЗМІСТ

ВСТУП

Сьогодні цифровізація освіти змінює підходи до організації роботи навчальних закладів, і особливо важливо знайти спосіб оптимізувати складні та часозатратні процеси. Формування студентських розкладів — ще той головний біль. Треба врахувати навчальні плани, коли і які аудиторії вільні, чи не перевантажені викладачі, а ще, по можливості, врахувати побажання кожного учасника освітнього процесу. Все це — справжній пазл. Звичайно, ручне складання розкладів або використання примітивних інструментів не витримують конкуренції: часто помиляються, займають багато часу, а результат залишає бажати кращого. Тут автоматизація стає незамінною. Завдяки сучасним інформаційним системам можна запросто вирішити питання швидкості, точності та врахувати всі обмеження та критерії без стресу. Автоматизований процес розробки розкладу — це не лише про зручність. Це й про те, що навчальний процес стає більш організованим, ресурси використовуються ефективніше, а студенти й викладачі отримують графік, який реально працює. Головна мета цієї роботи — створити систему автоматизованого формування розкладу занять для студентів, яка допоможе оптимізувати планування, зменшити кількість конфліктів у розкладах і значно підвищити ефективність організації навчання.

РОЗДІЛ 1: АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Що таке розклад?

Розклад — це календарний план, що координує учнів і викладачів у межах аудиторій та часових інтервалів шкільного дня. До інших чинників належать навчальні дисципліни та тип наявних приміщень (наприклад, лабораторії)[1]. Така організація допомагає студентам і викладачам не плутатися, тримати темп протягом семестру або тижня, ефективніше планувати свій час та сили. Зазвичай вища школа працює за “парами” — кожна триває 80 хвилин (іноді з перервою, іноді — без). Але й тут можливі відхилення: кожен заклад має свої правила. На практиці розклади бувають різні — семестровий (на довгий період), тижневий (з урахуванням “чисельника” та “знаменника”) й спеціальний, екзаменаційний.[1]

З чого взагалі складається типовий розклад? Тут вказують день і час занять, дисципліну, викладача (часто з ініціалами), номер аудиторії, тип заняття (лекція, практика, лабораторна, семінар), підгрупу, якщо є поділ, і для багатокорпусних вишів — корпус. Іноді додатково використовують кольори або символи, щоб одразу можна було “читати” інформацію для окремих спеціальностей.[1]

Скласти розклад — це не просто “позаписувати у клітинки предмети”. Це складна логістична задача, і ось чому. По-перше, треба уникати “вікон” — довгих пауз між парами для студентів. В ідеалі їх не має бути, але через нестачу аудиторій іноді без них не обійтись. По-друге, важливо рівномірно розподіляти навантаження на викладачів — жодних шести пар поспіль у різних корпусах, бо так і до перевтоми недалеко. Далі — технічні деталі: для лекцій потрібна велика зала; хімічна лабораторія потребує витяжної шафи і реактивів; для інформатики — комп’ютерний клас із релевантним програмним забезпеченням. Четвертий момент — циклічність: деякі дисципліни йдуть лише у “чисельний” тиждень або чергуються, тому важливо чітко позначити типи тижнів у розкладі.[1]][2]

Не можна ігнорувати санітарно-гігієнічні вимоги: дві важкі теоретичні лекції підряд — погане рішення, так само, як і дві фізкультури поспіль. Треба

перемішувати навантаження. Далі — узгодження з іншими групами і потоками: якщо лекцію читають одразу для кількох груп, у цей час вони не можуть мати нічого іншого, а викладач не має бути залучений в іншому потоці. Дуже важливий аспект — бронювання аудиторій: невелика помилка може знищити всі старання, якщо одну лабораторію “прописали” двом викладачам одночасно. Диспетчер чи адміністратор має тримати актуальну базу по кожній аудиторії — інакше хаос.[27]

Ще одна складність — побажання студентів щодо вибіркових дисциплін. Чим освітня система гнучкіша, тим важче все підлаштувати, адже до розкладу треба закласти результати опитування про зручний час занять. Не можна забувати й про форс-мажори: завжди має бути запасний сценарій на випадок хвороби викладача, аварій в корпусі чи університетських подій, які неочікувано змінять плани.[1][30]

Розклад — це справжній “живий” механізм, і щоб він працював, враховують безліч таких нюансів.

1.2 Автоматизація розкладу занять, як можливість оптимізувати функціонал закладу освіти

Автоматизація складання розкладу занять є потужним інструментом оптимізації функціоналу закладу освіти, оскільки вона переводить процес, який традиційно потребує величезних часових та інтелектуальних витрат від заступника директора або навчальної частини, у площину швидких та математично вивірених алгоритмічних рішень. У класичному ручному режимі створення розкладу - це постійний пошук компромісу між вимогами санітарних норм, побажаннями викладачів, наявністю спеціалізованих аудиторій, розподілом навантаження між групами та необхідністю уникнути "вікон" або перевантаження в окремі дні. Цей процес часто займає кілька тижнів на початку семестру, а будь-яка зміна (хвороба викладача, перенесення пари) вимагає фактично ручного перезапуску логістики, що призводить до збоїв, подвійного бронювання аудиторій та, як наслідок, до хаосу в освітньому процесі. Автоматизація вирішує ці

проблеми шляхом створення єдиного цифрового простору, де всі обмеження та ресурси вносяться як вхідні параметри. Сучасні системи управління навчальним процесом здатні генерувати оптимальний розклад за лічені хвилини, враховуючи сотні умов одночасно: рівномірний розподіл навантаження на студентів та викладачів, пріоритетність пар залежно від складності дисципліни, мінімізацію переміщень між корпусами, чергування теоретичних і практичних занять, дотримання норм міжпарами. Оптимізація функціоналу закладу в цьому контексті проявляється одразу на кількох рівнях. По-перше, адміністративний ресурс звільняється від рутинної паперової роботи: замість постійних узгоджень та виправлення помилок, співробітники можуть займатися аналітикою, моніторингом якості освіти або розвитком матеріально-технічної бази. По-друге, підвищується ефективність використання інфраструктури - автоматизована система не допускає простою дорогих лабораторій або спортзалів, оскільки враховує їхню завантаженість у реальному часі, що дозволяє проводити більше занять без додаткових витрат на будівництво нових приміщень. По-третє, значно зростає прозорість для всіх учасників процесу: викладачі та студенти отримують доступ до актуального розкладу через мобільні додатки або веб-інтерфейс, де будь-які зміни автоматично публікуються та синхронізуються з календарями, що знижує кількість пропущених занять через нерозуміння графіку. Крім того, автоматизація дозволяє проводити глибокий аналіз накопичених даних: система може виявити дисбаланс у навантаженні між різними факультетами, спрогнозувати виникнення конфліктів у розкладі ще до початку семестру, а також згенерувати звіти для акредитації або перевірок контролюючих органів без додаткового ручного введення. У результаті заклад освіти отримує не просто розклад як документ, а гнучку адаптивну систему управління часом та ресурсами, що здатна швидко реагувати на зовнішні та внутрішні зміни - від карантинних обмежень до індивідуальних графіків викладачів. Економічний ефект також очевидний: зменшення кількості понаднормової роботи адміністраторів, оптимізація комунальних витрат за рахунок ущільнення занять у першу зміну, подовження

терміну експлуатації обладнання через рівномірне навантаження - усі ці фактори роблять автоматизацію розкладу стратегічною інвестицією, а не просто технічним нововведенням. Таким чином, перехід від ручного планування до автоматизованого генерування розкладу є ключовим чинником підвищення загальної ефективності закладу освіти, оскільки він оптимізує головний дефіцитний ресурс - час - і перенаправляє людські зусилля з рутинного узгодження на творчий та аналітичний розвиток навчального процесу.

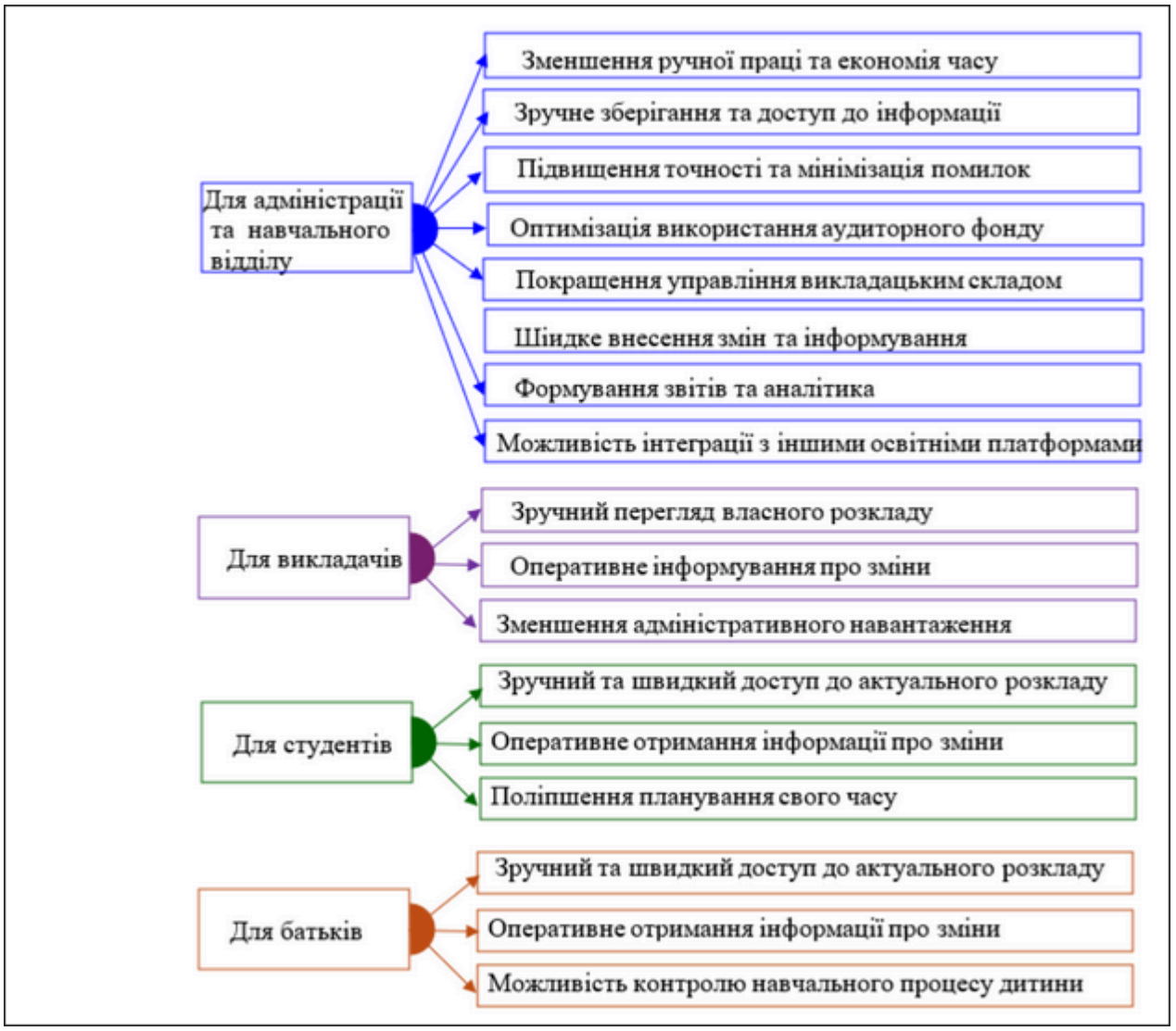


Рис. 1 - переваги автоматизованого розпорядку занять[2]

1.3 Аналіз готових/подібних рішень

Подібних застосунків є чимало, для першого прикладу розглянемо веб застосунок “schedulebuilder”

Рис. 2 - логотип schedulebuilder [3]

Даний застосунок дозволяє створювати розклади під індивідуальні потреби, має зручний і гнучкий інтерфейс. Але даний застосунок не може сам генерувати розклад, і є лише приладом для планування.

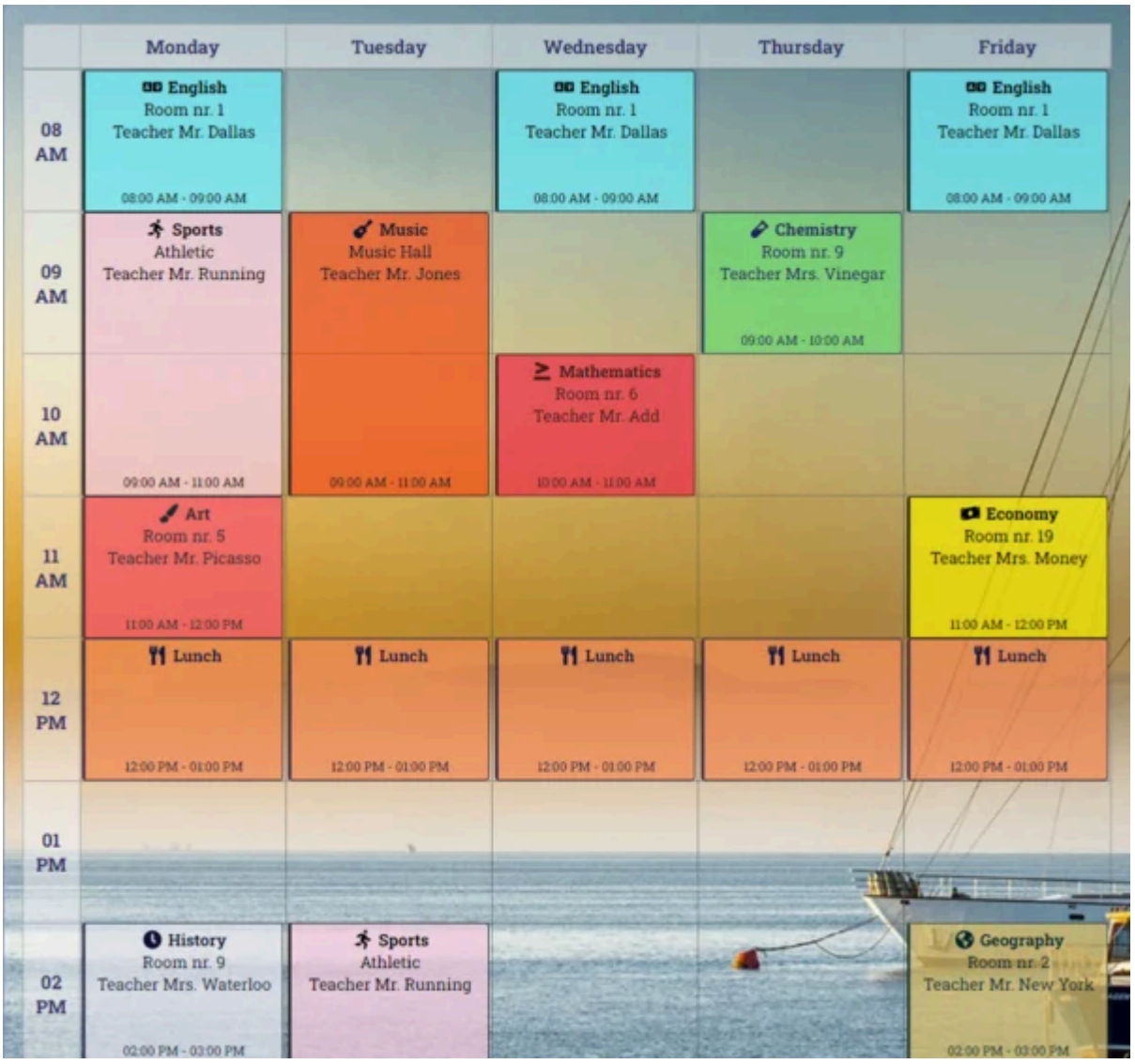


Рис. 3 - інтерфейс schedulebuilder[3]

Для наступного прикладу розглянемо застосунок “Canva”. Canva - платформа графічного дизайну, що дозволяє користувачам створювати графіку, презентації, афіші та інший візуальний контент для соціальних мереж. Доступна як вебверсія, так і мобільна. Сервіс пропонує великий банк зображень, шрифтів, шаблонів та ілюстрацій.[1]

Рис. 4 - логотип Canva[4]

За допомогою цього застосунку також передбачена можливість створення розкладів, оскільки він пропонує користувачеві набір достатньо гнучких інструментів для повноцінної роботи з таблицями, виконуючи, по суті, роль цифрової канви. Це означає, що програма не пропонує жодних шаблонних рішень чи поведінкових алгоритмів, натомість вона надає у ваше розпорядження заздалегідь підготовлену візуальну структуру - сітку з комірок, рядків і стовпців, які користувач заповнює самотужки, спираючись виключно на власне бачення майбутнього графіку. Інакше кажучи, ви отримуєте не готовий результат, а лише середовище, своєрідний "чистий аркуш", у межах якого можна вручну креслити межі занять, вносити назви предметів, позначати часові проміжки та виконувати форматування на власний розсуд.[4]

Водночас, розглядаючи функціональні можливості даного програмного продукту, необхідно чітко розмежовувати поняття "інструменту для ручного введення" та "автоматизованої системи генерації". Попри те, що зовні програма може нагадувати планувальник, вона не здатна працювати автономно й самостійно аналізувати чи обробляти вхідні дані, які в неї завантажуються. Цей застосунок не є інтелектуальним генератором розкладів, адже він не виконує жодних логічних операцій з розподілу навантаження, не враховує перетин змінних, не вирішує

часових конфліктів між подіями та не здійснює автоматичної побудови послідовностей. Простіше кажучи, будь-яка аналітична робота, пов'язана з плануванням, цілковито покладається на людину, тоді як застосунок залишається лише візуальною оболонкою для фіксації вже готових, заздалегідь обміркованих користувачем розкладів. Саме тому важливо усвідомлювати, що без безпосередньої участі оператора, який власноруч наповнює таблицю змістом, система не здатна видати жодного структурованого результату.[4]

Розглянемо наступний застосунок, “timetablemaster”.

Рис. 5 - логотип timetablemaster[5]

TimetableMaster - це сучасне хмарне рішення, побудоване із застосуванням технологій штучного інтелекту, яке дійсно здатне допомагати школам і коледжам створювати розклади відчутно швидше, ніж це можливо за умов виключно ручного опрацювання. Його беззаперечною перевагою є те, що воно бере на себе рутинну частину процесу, зменшуючи вплив людського фактору на етапі первинного компонування сітки занять, і в цьому сенсі його практична цінність для невеликих закладів освіти не викликає сумнівів. Однак, коли мова заходить про автоматичну комплексну генерацію розкладу, яка б оперувала повноцінною, вже існуючою базою даних, цей інструмент стикається з цілою низкою фундаментальних обмежень. Саме ці обмеження, закладені в саму архітектуру продукту, не дозволяють розглядати його як по-справжньому універсальне рішення, придатне для масштабних закладів освіти з багаторівневою та розгалуженою структурою навчального процесу.[5]

Перш за все, необхідно чітко окреслити позиціонування самого продукту: TimetableMaster є насамперед ШІ-асистентом, інтелектуальним помічником, призначеним для полегшення праці людини-упорядника, а не системою повної автоматизації, яка передбачає глибоку, наскрізну інтеграцію із зовнішніми

джерелами даних. Логіка його роботи вибудована за принципом "внесіть необхідне - отримайте результат", і цей підхід реалізовано наступним чином: користувач власноруч вводить до системи перелік викладачів, назви предметів, позначення класів чи груп, а також перелік доступних аудиторій, після чого вручну встановлює низку обмежень, що стосуються, наприклад, зайнятості педагогів або місткості приміщень. Лише після цього вбудований алгоритм розпочинає спроби згенерувати так званий "безконфліктний" розклад, у якому, наскільки це можливо, усунуто перетини між зазначеними елементами. Така модель поведінки цілком успішно спрацьовує для відносно простих, типових шкільних сценаріїв, де структура даних є лінійною, а кількість змінних - обмеженою. Проте вона принципово не враховує тієї складної, багаторівневої та багатовимірної структури даних, яка роками формується і функціонує в університетах, академіях або великих освітніх комплексах, що об'єднують різні ступені навчання.[5]

Реальна ж автоматична генерація розкладу в масштабах великого закладу потребує не просто набору ізольованих таблиць, куди інформацію щоразу заносять окремо, а опори на нормалізовану реляційну базу даних. У такій базі навчальні плани, розбиті на семестри дисципліни, потоки, що об'єднують кілька академічних груп, поділ на підгрупи для лабораторних занять, види навчальних занять (лекції, практичні, семінари), семестрові та календарні графіки, атестаційні тижні, модульні контролі, а також індивідуальні освітні траєкторії студентів - усі ці сутності пов'язані між собою надзвичайно складною, розгалуженою мережею взаємних обмежень. Кожен елемент у цій системі впливає на інші, і вилучення або зміна хоча б одного з них може призвести до каскадного ефекту по всьому розкладі. Справжня інтелектуальна система повинна вміти не просто приймати нові дані, а й підхоплювати та використовувати ту базу даних, яка вже існує в закладі, є актуальною та наповненою відомостями про викладачів, їхнє навантаження, ставки, побажання щодо часу, сумісництво тощо. Натомість TimetableMaster функціонує на рівні "введіть дані з нуля в нашому інтерфейсі →

отримайте розклад", і це є його концептуальною межею. Інструмент не розрахований на те, щоб під'єднуватися до зовнішньої, вже розгорнутої та наповненої реляційної бази даних, яка містить картки викладачів і всі пов'язані з ними атрибути, а отже, не може вважатися рішенням, що забезпечує наскрізну автоматизацію в умовах великої освітньої екосистеми.

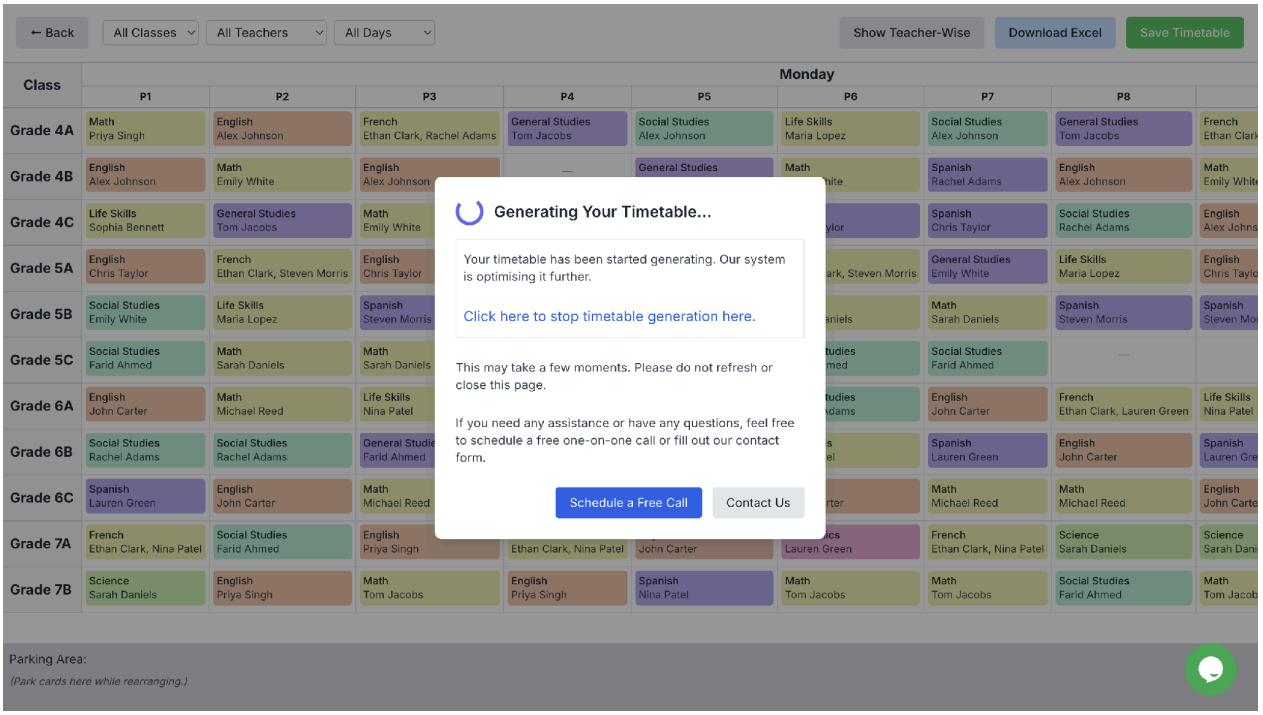


Рис. 6 - генерація розкладу[5]

Подібним додатком за принципом роботи є “aSc Timetables”.

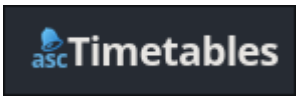


Рис. 7 - логотип aSc Timetables

aSc Timetables - це повністю автоматизована програма для генерування розкладу за допомогою складного алгоритму. Однією з важливих переваг є можливість гнучкого налаштування. Користувач може задавати індивідуальні правила для кожного викладача чи класу, наприклад, обмеження на кількість уроків підряд або побажання щодо вільних днів. Це дозволяє створювати розклад,

який максимально відповідає реальним умовам навчального процесу. Інтерфейс програми є інтуїтивно зрозумілим, що полегшує роботу навіть для користувачів без глибоких технічних знань. Візуальне представлення розкладу допомагає швидко оцінити результат і внести необхідні корективи. Крім того, aSc Timetables підтримує експорт розкладу у різні формати, що дозволяє легко публікувати його на сайтах або друкувати. Також даний застосунок дає змогу розробникам інтегрувати функціонал у свої застосунки за допомогою API[6].

1.4 Вимоги до сучасної системи управління розкладом

Сучасна система управління розкладом для вищого навчального закладу повинна відповідати комплексу функціональних, нефункціональних та організаційних вимог, спрямованих на подолання існуючих проблем та забезпечення ефективної організації навчального процесу. Функціональні вимоги включають повний життєвий цикл управління розкладом: від введення вихідних даних (наявність викладачів, студентських груп, аудиторій, дисциплін з їх атрибутами) та задання складних обмежень (специфікація аудиторій, індивідуальна наявність викладачів, послідовність занять, мінімальні/максимальні інтервали) до автоматичного генерування розкладу за допомогою оптимізаційних алгоритмів, що мінімізують конфлікти та "вікна". Система має надавати зручний веб-інтерфейс для ручного коригування згенерованого варіанту з миттєвою перевіркою на порушення правил, а також підтримувати рольову модель з різними правами доступу для адміністраторів, розкладників, завідувачів кафедр, викладачів та студентів. Критично важливим є забезпечення оперативної комунікації: автоматичне сповіщення про зміни через email або месенджери, публічний календар для кожного користувача з фільтрацією та можливістю експорту в Google Calendar/Outlook, а також механізм запитів на заміну або перенесення занять. Система повинна генерувати звіти для аналітики: завантаженість аудиторій, індивідуальне навантаження викладачів, наявність "вікон", що дозволяє приймати обґрунтовані управлінські рішення.

Нефункціональні вимоги стосуються продуктивності, надійності та зручності: система має бути швидкою та забезпечувати стабільну роботу під час пікового навантаження (початок семестру), мати інтуїтивно зрозумілий інтерфейс, адаптований під мобільні пристрої, та бути захищеною від несанкціонованого доступу з використанням авторизації та шифрування чутливих даних. Технічні вимоги передбачають гнучку архітектуру, що дозволяє інтегруватися з існуючими системами ВНЗ через API або імпорт/експорт даних у поширених форматах (Excel, CSV, JSON), а також легкість впровадження та супроводу, що особливо важливо для закладів з обмеженими IT-ресурсами. Крім того, система має бути адаптованою до локальних особливостей, таких як україномовний інтерфейс, підтримка національних освітніх стандартів, облік специфіки денної та заочної форми навчання, а також мати можливість кастомізації правил генерування під конкретний заклад. Таким чином, ідеальна система управління розкладом – це не просто інструмент автозаповнення комірок у таблиці, а комплексна платформа для організації, аналізу та оперативного управління навчальним процесом, орієнтована на користувача та здатна адаптуватися до постійних змін.[26][29][32]

1.5 Формування мети та завдань роботи

Мета кваліфікаційної роботи — створити, спроєктувати й реалізувати програму, яка сама складає розклад занять для студентів та викладачів. Вона має розподіляти пари з урахуванням груп, викладачів, аудиторій і часових рамок. Система зменшує кількість конфліктів у розкладі й полегшує керування навчальним процесом.[35]

Для досягнення поставленої мети необхідно виконати такі завдання:

Проаналізувати як саме організовують навчальний процес у навчальних закладах та які вимоги існують до побудови розкладу занять.

Дослідити наявні методи й програмні засоби автоматизованого складання розкладів, виявити їхні сильні та слабкі сторони.

Сформулювати функціональні й нефункціональні вимоги до системи автоматизованого формування розкладу.

Спроєктувати архітектуру системи та структуру бази даних.

Обґрунтувати вибір алгоритмів автоматизованого формування розкладу з урахуванням заданих обмежень.

Реалізувати програмне забезпечення системи автоматизованого формування розкладу занять.

Провести тестування розробленої системи і проаналізувати результати.

1.6 Висновки до першого розділу

У результаті всебічного аналізу предметної області формування навчальних розкладів, а також ґрунтовного вивчення сучасних підходів до їх автоматизації, було прийнято виважене та обґрунтоване рішення щодо розробки спеціалізованої програмної системи. Ця система покликана забезпечити можливість автоматизованої генерації розкладу занять із обов'язковим урахуванням цілої сукупності обмежень, які накладаються навчальним процесом, а також індивідуальних побажань викладачів, що стосуються зручного для них часу проведення занять, рівномірності тижневого навантаження, послідовності пар та можливості уникнення так званих «вікон» у власному графіку.[27][35]

Створення системи подібного роду має на меті сприяти суттєвій оптимізації всього процесу планування навчального навантаження, який традиційно вважається одним із найбільш трудомістких, відповідальних і чутливих до помилок етапів організації освітнього процесу. Йдеться про те, що ручне складання розкладу, особливо в умовах значної кількості груп, викладачів і аудиторного фонду, неминуче призводить до виникнення конфліктних ситуацій, подвійних призначень, нераціонального використання приміщень або до систематичного нехтування суб'єктивними побажаннями педагогічного складу. Запропонована система, на противагу ручному підходу, забезпечуватиме ефективний, логічно вивірений розподіл занять протягом робочого тижня, беручи

до уваги такі ключові чинники, як фактична доступність викладачів у конкретні дні та години, перелік закріплених за ними предметів із відповідним обсягом годин, а також наявність і граничну місткість власних аудиторій, що можуть бути використані для проведення занять різного типу. Крім того, система дозволяє уникнути ситуацій, коли приміщення одночасно призначається для двох різних груп, і водночас мінімізує непродуктивні перерви в роботі викладачів, що позитивно впливає на загальну якість організації навчального процесу.[27][28]

Паралельно з цим було проведено окремий, детальний аналіз сучасних програмних засобів, представлених на ринку та призначених для складання розкладів. За підсумками цього аналізу можна зробити однозначний висновок про те, що переважна більшість існуючих рішень позиціонується як складні, багатофункціональні та багатомодульні системи. Вони передусім орієнтовані на потреби великих навчальних закладів — університетів, академій або освітніх комплексів із розгалуженою інфраструктурою, департаментами, кафедрами та тисячами студентів. Відповідно, такі системи мають або високу вартість ліцензій та впровадження, що є фінансово обтяжливим для невеликих коледжів чи окремих факультетів, або ж містять надлишкову функціональність, яка суттєво ускладнює їх освоєння та щоденне використання. Користувач у подібних середовищах часто стикається з перевантаженим інтерфейсом, необхідністю опрацьовувати десятки параметрів, більшість із яких не є актуальними для завдань його рівня, та потребою у спеціальному навчанні для роботи з програмою. Таким чином, розрив між реальними потребами відносно невеликих освітніх структур і тим, що пропонує ринок, стає цілком очевидним.[27][29]

Виходячи з вищезазначеного, можна дійти логічного та цілком обґрунтованого висновку про безсумнівну доцільність розробки простого, інтуїтивно зрозумілого, але водночас функціонально достатнього програмного засобу для автоматизованого формування розкладу занять. Цей засіб повинен бути сфокусований саме на тих завданнях, які становлять щоденну практичну цінність: врахування індивідуальних побажань викладачів щодо часу, коректне

опрацювання їхнього сумарного навчального навантаження, автоматичне накладання ресурсних обмежень, пов'язаних з аудиторним фондом, а також запобігання будь-яким часовим колізіям. При цьому ключовими вимогами до розроблюваного продукту залишаються загальна зручність використання, що не потребує тривалої підготовки персоналу, та фінансова й технічна доступність, яка робить його придатним для впровадження у закладах з помірним бюджетом та обмеженими ІТ-ресурсами без втрати якості кінцевого результату.[27]

РОЗДІЛ 2: ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ ТА АРХІТЕКТУРИ СИСТЕМИ

1.1 Аналіз фреймворку Flask для розробки back-end

Flask як мікрофреймворк для розробки веб-додатків на Python[15][24] пропонує унікальний баланс між мінімалізмом та потужністю, що робить його ідеальним вибором для розробки спеціалізованих систем, таких як автоматизоване формування розкладу, особливо в контексті дипломного проекту або стартапу. Його головна філософія полягає в наданні лише базового каркасу, навколо якого розробник може створювати структуру, обираючи лише ті бібліотеки та інструменти, які необхідні саме для конкретного завдання[1]. Це суттєво відрізняє Flask від більш монолітних фреймворків, як-от Django, які нав'язують певну архітектуру та комплект інструментів "з коробки"(див. Рис. 7). Для завдання реалізації складної бізнес-логіки (оптимізаційних алгоритмів генерування розкладу) ця гнучкість є критичною, оскільки дозволяє інтегрувати спеціалізовані математичні бібліотеки (PuLP, OR-Tools, SciPy) без необхідності "боротьби" з надмірними абстракціями монолітного фреймворку. Архітектура Flask прозора та передбачувана: система маршрутизації через декоратори @app.route дозволяє чітко структурувати API та точки входу, механізм шаблонізації Jinja2[14] (інтегрований за замовчуванням) ефективно відокремлює логіку від представлення, а підтримка "застосунків" (Blueprints) дає змогу організувати великий проект на модулі (наприклад, окремо модуль аутентифікації, модуль розкладу, модуль адміністрування), що підвищує читабельність та підтримуваність коду[8].

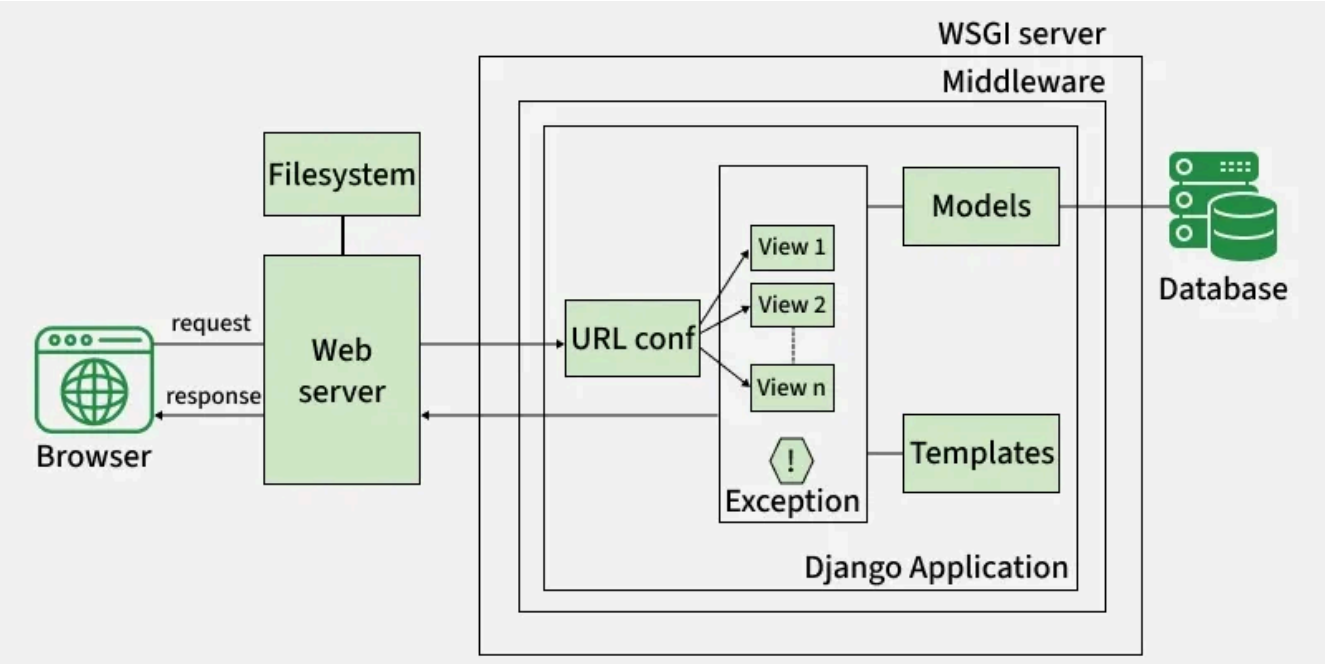


Рис. 7 - Робоча архітектура фреймворка Django[7]

Екосистема розширень Flask-SQLAlchemy, Flask-Login, Flask-WTF, Flask-Migrate та інших надає готові, добре інтегровані рішення для типових завдань (робота з БД, автентифікація, обробка форм, міграції), причому розробник обирає лише ті компоненти, які потрібні. Для системи управління розкладом це означає можливість швидко побудувати безпечну адміністративну панель для управління даними (викладачами, групами, аудиторіями), реалізувати рольовий доступ та налагодити процес введення та перевірки вхідних обмежень. З точки зору продуктивності, Flask демонструє хороші показники для середньонавантажених систем типу "запис-чит-онлі", де основне навантаження припадає на логіку формування розкладу, яка може бути винесена в окремі фонові завдання або оброблятися асинхронно. Важливою перевагою для навчального та дипломного проекту є низький поріг входження, зрозуміла документація та велика спільнота, що дозволяє швидко знаходити відповіді на питання. Однак мінімалізм Flask одночасно накладає і відповідальність на розробника: він має самостійно обирати та інтегрувати компоненти для безпеки (CSRF-захист, санітизація вводу), кешування, управління сесіями та інші важливі аспекти, що вимагає глибшого

розуміння принципів веб-розробки. Для системи розкладу це також означає необхідність ретельної побудови власної архітектури обробки даних та станів, оскільки сама модель даних (складні взаємозв'язки між сутностями) та алгоритми є ядром програми. Підсумовуючи, Flask з його "конструкторським" підходом є оптимальним інструментом для створення ефективного back-end системи автоматизації розкладу, де розробник потребує повного контролю над логікою та структурою, не відволікаючись на нав'язані надбудови, і може сконцентруватися на вирішенні головної науково-прикладної задачі – розробці та впровадженню оптимальних алгоритмів генерування навчального розкладу.[7]

1.2 Вибір бази даних (MySQL)

Обираючи систему керування базами даних (СКБД) для системи автоматизованого формування розкладу, необхідно враховувати такі ключові фактори, як надійність зберігання даних, ефективність виконання складних зв'язних запитів, зрілість технології, спрощене адміністрування та сумісність із вибраним стеком розробки (Flask, Python). MySQL відповідає цим вимогам та є одним з найпоширеніших рішень для веб-додатків середньої складності. На противагу легкій SQLite, яка ідеальна для прототипування [9], MySQL розроблена для роботи у виробничому середовищі, підтримує одночасну роботу великої кількості користувачів, надає потужні механізми транзакційної цілісності та розвинену систему прав доступу, що є критично важливим для багатокористувацької системи з різними ролями (адміністратор, викладач, студент). Архітектура даних системи розкладу передбачає наявність багатьох сутностей (факультети, групи, студенти, викладачі, дисципліни, аудиторії, заняття) зі складними зв'язками "один-до-багатьох" та "багато-до-багатьох" (наприклад, один викладач може вести кілька дисциплін у різних групах, а одна аудиторія може використовуватися для різних занять у різний час). MySQL, будучи реляційною СКБД, ідеально підходить для моделювання таких структурованих зв'язків, а завдяки підтримці зовнішніх ключів (foreign keys) та механізму ACID

(Atomicity, Consistency, Isolation, Durability) гарантує цілісність та узгодженість даних при паралельних операціях оновлення, що є ключовим при ручному коригуванні розкладу кількома адміністраторами. Продуктивність MySQL для операцій вибірки (SELECT) з складними JOIN-запитами, необхідними для формування зведеного розкладу або перевірки на конфлікти, є високою, особливо при правильній індексації полів, що беруть участь у пошуку та фільтрації (наприклад, поля дати, ідентифікаторів викладача та аудиторії)[10].

Важливою практичною перевагою є широка поширеність та зрілість MySQL: існує велика кількість документації, прикладів, інструментів для адміністрування (наприклад, phpMyAdmin, MySQL Workbench) та готових рішень для резервного копіювання та масштабування. Для інтеграції з Flask через ORM SQLAlchemy MySQL є одним з найкраще підтримуваних "діалектів". Драйвер PyMySQL або mysqlclient дозволяють легко підключити базу до додатку. Незважаючи на те, що сучасні тенденції часто вказують на PostgreSQL як на потужнішу альтернативу з розширеною підтримкою складних типів даних та більш суворим дотриманням стандартів SQL, для завдань системи розкладу MySQL пропонує достатній функціонал, а її простота у налаштуванні та менші вимоги до ресурсів у початковій фазі розробки є вагомим аргументом. Крім того, сумісність з великою кількістю хостинг-провайдерів та платформ (включно з низькобюджетними віртуальними серверами VPS[11]) спрощує майбутнє розгортання готової системи у навчальному закладі. Таким чином, вибір MySQL як СКБД забезпечує оптимальний баланс між продуктивністю, надійністю, легкістю розробки та експлуатації для веб-додатку зі структурованою реляційною моделлю даних середнього масштабу, яким є система автоматизованого формування розкладу занять.[25]

1.3 Висновки до другого розділу

Проаналізувавши сучасні програмні засоби та технології, що застосовуються для розробки веб-орієнтованих систем автоматизованого

формування розкладу, можна зробити висновок, що ефективність таких систем значною мірою залежить від гнучкості обраного інструментарію, а також від правильності побудови архітектури та моделі даних. Використання мікрофреймворку Flask забезпечує необхідний баланс між простотою та розширюваністю, дозволяючи зосередитись на реалізації складної бізнес-логіки, зокрема алгоритмів оптимізації розкладу.

Головними недоліками подібних систем можна вважати:

1. складність реалізації та підтримки власної архітектури через мінімалізм фреймворку;
2. необхідність самостійної інтеграції компонентів безпеки та додаткових сервісів;
3. залежність продуктивності від якості реалізації алгоритмів та структури бази даних.

Такі недоліки можуть впливати на швидкість розробки, стабільність роботи системи та вимагати від розробника глибшого розуміння принципів веб-розробки і проєктування програмного забезпечення.[1]

Використання сучасних технологій дозволяє ефективно поєднувати різні інструменти для створення гнучких і масштабованих інформаційних систем. У якості основної технології для розробки серверної частини обрано мову програмування Python[25] та мікрофреймворк Flask, що забезпечує високу адаптивність і контроль над логікою застосунку. Для зберігання даних використовується реляційна система керування базами даних MySQL, яка гарантує надійність, цілісність та ефективну обробку складних запитів у багатокористувацькому середовищі.[31]

Обраний стек технологій дозволяє створити ефективну, гнучку та масштабовану систему автоматизованого формування розкладу, здатну працювати в умовах реального навчального процесу та забезпечувати якісну обробку складних взаємозв’язків між даними.[32]

РОЗДІЛ 3: ПРОЕКТУВАННЯ СИСТЕМИ АВТОМАТИЗОВАНОГО ФОРМУВАННЯ РОЗКЛАДУ

2.1 Функціональні вимоги до системи

Система автоматизованого формування розкладу має забезпечувати комплексне управління повним життєвим циклом навчального розкладу від початкового внесення даних до публікації та оперативного коригування. Головною вимогою є можливість автоматичного генерування розкладу на основі заданих обмежень та критеріїв оптимальності. Для цього система повинна надавати інтерфейс для повного та гнучкого конфігурування обмежень, які поділяються на обов'язкові (жорсткі) та бажані (м'які). До жорстких обмежень належать: унікальність викладача, групи та аудиторії в один часний слот; відповідність аудиторії типу заняття (лекційна, лабораторна, спортивна зала) та її місткість кількості студентів у групі; врахування неробочих годин та днів для конкретних викладачів; заборона розміщення занять поза робочим часом навчального закладу. М'які обмеження, порушення яких система намагається мінімізувати, включають: мінімізацію "вікон" у розкладах викладачів та студентських груп; рівномірний розподіл навантаження по днях тижня для груп; забезпечення певної послідовності типів занять (наприклад, лекція перед семінаром); розміщення складних дисциплін у першій половині дня; забезпечення мінімального часу на переходи між корпусами.[26]

Система має підтримувати повний цикл управління вхідними даними (CRUD-операції) для всіх сутностей: факультетів, кафедр, спеціальностей, навчальних груп (з можливістю імпорту списку студентів), викладачів (з прив'язкою до кафедри, вказівкою ставки, наукового ступеня та індивідуальних часових обмежень), дисциплін навчального плану (з типом, годинами, призначеними викладачами), аудиторного фонду (з атрибутами: корпус, номер, місткість, тип, обладнання). Критично важливим є модуль навчального плану, де

адміністратор може встановити, які дисципліни, у якому семестрі та для яких груп є обов'язковими, а також призначити викладачів.[28]

Для користувачів системи передбачається рольова модель з чітким розмежуванням прав: Адміністратор має повний доступ до всіх даних та функцій, включаючи конфігурацію системи, запуск генерації розкладу та його затвердження. Методист/Розкладник (може бути співробітник деканату) має права на управління даними своїх факультетів/кафедр, запуск генерації, ручне коригування та перегляд звітів. Викладач може переглядати свій персональний розклад, подавати запити на перенесення або заміну занять (з вказівкою причини), переглядати своє навантаження, а також вести облік відвідування для своїх груп (опціонально). Студент отримує доступ до розкладу своєї групи в зручному вигляді (денний, тижневий перегляд), може переглядати інформацію про викладача та аудиторію, а також отримувати сповіщення про зміни.[26][34]

Система повинна надавати зручні інструменти візуалізації та роботи з розкладом: календарне відображення (тижневий, денний вигляд) з можливістю фільтрації за групою, викладачем або аудиторією; drag-and-drop інтерфейс для ручного переміщення занять з автоматичною перевіркою на нові конфлікти; виділення кольором порушень правил або типу занять. Експорт та публікація є обов'язковою функцією: система повинна генерувати розклад у форматі PDF для друку, експортувати в Excel, а також надавати публічне посилання або можливість інтеграції розкладу у вигляді iCal (Google Calendar, Outlook) для кожного викладача та групи.[27][33]

Для підтримки процесу затвердження та інформування необхідний механізм сповіщень та історії змін: автоматична розсилка email-повідомлень про затвердження нового розкладу, про перенесення пар, а також ведення логу всіх змін (хто, коли і що змінив). Система повинна генерувати аналітичні звіти: про завантаженість аудиторій, про виконання індивідуального плану викладача, про наявність "вікон", про суперечності в навчальних планах, що дозволить приймати обґрунтовані управлінські рішення.

Функціонал також повинен включати резервне копіювання та відновлення даних, багатокористувацький доступ без конфліктів (блокуються одночасні редагування одного об'єкта), та підтримку багатомовності (принаймні української та англійської мови інтерфейсу). Усі дії користувачів, особливо адміністративні, повинні підлягати логіюванню для забезпечення відстеженості.[26][27]

2.2 Алгоритм автоматичного генерування розкладу з обмеженнями

Алгоритм автоматичного генерування розкладу являє собою ядро системи та реалізується як задача задоволення обмежень (Constraint Satisfaction Problem, CSP) з елементами оптимізації, що може бути доповнена евристичними методами для покращення продуктивності на великих наборах даних. На вхід алгоритм отримує: множину навчальних груп G, множину викладачів T, множину дисциплін D з призначеними викладачами та типами занять, множину аудиторій R з атрибутами, множину часових слотів S (день тижня, номер пари) та множину обмежень C. Алгоритм працює у кілька етапів. На етапі препроцесингу дані валідуються та трансформуються: перевіряється непротиріччість обмежень, складається список конкретних завдань L – кожне завдання є трійкою (дисципліна, група, викладач) з кількістю пар на тиждень, типом заняття та пріоритетом. Завдання, що потребують поділу на підгрупи, розбиваються. На етапі формульного опису задачі кожна змінна X_ij відповідає призначенню часового слота S_i та аудиторії R_j для конкретного завдання L_k[22]. Домени змінних формуються з усіх можливих комбінацій слотів та аудиторій, що відповідають базовим обмеженням (робочий час, тип аудиторії). Далі накладаються жорсткі обмеження у вигляді предикатів, які фільтрують домени: унікальність викладача в одному слоті; унікальність групи в одному слоті; унікальність аудиторії в одному слоті; відповідність місткості аудиторії кількості студентів; заборона занять поза індивідуально доступними для викладача годинами; дотримання послідовності типів занять (якщо задано); обов'язковість певних дисциплін у конкретні дні (за розкладом кафедри). Після цього визначаються м'які обмеження, порушення яких мінімізується цільовою

функцією: мінімізація кількості "вікон" у розкладах груп та викладачів (штраф за кожне вікно); рівномірний розподіл навантаження по днях для кожної групи (штраф за перенавантаження окремих днів); пріоритет розміщення "складних" дисциплін у першій половині дня; мінімізація переміщень між корпусами для груп (штраф за перерви між заняттями в різних корпусах).[29]

Для пошуку рішення може застосовуватись комбінований підхід. Спочатку використовується алгоритм пошуку з поверненням (backtracking) з евристиками упорядкування змінних (Variable Ordering Heuristic), наприклад, MRV (Minimum Remaining Values) – спочатку обираються завдання з найменшою кількістю вільних слотів, та евристикою упорядкування значень (Value Ordering Heuristic), що спочатку пробує значення (час-аудиторія), які залишають більшу свободу іншим змінним. Для прискорення застосовується пропагація обмежень (constraint propagation), зокрема техніка просування вперед (forward checking), яка миттєво прибирає з доменів суміжних змінних значення, що стають недопустимими після чергового призначення. Оскільки чистий CSP може не знайти допустимого розкладу в обмежений час через велику розмірність, на практиці алгоритм часто реалізують за допомогою методу локального пошуку, наприклад, алгоритму відпалу (simulated annealing) або табу-пошуку (tabu search), які працюють з повним, але можливо недопустимим початковим розкладом, а потім ітеративно покращують його, мінімізуючи сумарну вартість порушених жорстких та м'яких обмежень. Початковий розклад може генеруватись випадково або жадібним алгоритмом. Альтернативний шлях – формалізація як задачі цілочисельного лінійного програмування (ILP) з використанням бібліотек (PuLP, OR-Tools), де змінні бінарні, обмеження – лінійні нерівності, а цільова функція – сума штрафів. Це дає точне оптимальне рішення, але може бути обчислювально важким для великих екземплярів.[1][33]

Після отримання розкладу проводиться постобробка: перевірка на невраховані ручні обмеження, форматування виводу. Алгоритм також повинен надавати зворотний зв'язок: якщо розклад неможливо побудувати, система вказує

на "вузькі місця" – список конфліктних обмежень або завдань, що їх не вдалось розмістити (наприклад, через брак аудиторій певного типу або перевантаженість викладача), що дозволяє адміністратору скоригувати вхідні дані. Для інтеграції з Flask алгоритм інкапсулюється в окремий сервісний модуль, який може запускатись синхронно для невеликих даних або асинхронно (через Celery або фоновий потік) для великих обсягів, з можливістю відстеження прогресу та повернення результатів у веб-інтерфейс. Цей підхід забезпечує баланс між швидкістю, оптимальністю та гнучкістю у врахуванні специфічних вимог конкретного навчального закладу.[29][31]

2.3 Інтерфейс користувача: рольова модель (адмін, користувач)

Враховуючи специфіку проекту, де викладач та студент мають ідентичний набір прав та функціональних можливостей в системі, рольова модель спрощується до двох основних ролей: Адміністратор (Admin) та Користувач (User), де під користувачем розуміється одночасно і студент, і викладач. Такий підхід зберігає логіку розмежування доступу, але зменшує складність розробки та підтримки, фокусуючись на ключових відмінностях між тим, хто керує системою, і тим, хто її використовує для отримання інформації.

1. Роль: Адміністратор (Admin). Ця роль має повний, необмежений доступ до всіх функцій та даних системи. Інтерфейс адміністратора є комплексною панеллю управління (admin dashboard) і включає наступні ключові модулі:

Генерація та управління розкладом: Центральна сторінка з кнопкою запуску автоматичного генерування розкладу. Візуальне відображення згенерованого розкладу у вигляді інтерактивної сітки (тижневий/денний вигляд) з можливістю фільтрації за групою, аудиторією. Можливість затвердження фінальної версії розкладу, після чого він стає видимим для всіх користувачів.

Аналітика та звіти: Розділ для перегляду звітів: завантаженість аудиторій, навантаження користувачів, статистика "вікон", журнал змін у системі (хто і коли вносив зміни).[26][28][29]

Системні функції: Управління обліковими записами, скидання паролів, резервне копіювання бази даних.

Конфігурація системи: Сторінка для налаштування глобальних параметрів: робочі дні та години закладу, тривалість пар, перерви, корпуси. Налаштування та управління обмеженнями для генератора розкладу (жорсткі та м'які правила).

Управління даними: CRUD-інтерфейси для всіх сутностей: факультети/кафедри, навчальні групи, користувачі (як студенти, так і викладачі), дисципліни, аудиторії, навчальні плани. Можливість масового імпорту даних з CSV/Excel файлів.

2. Роль: Користувач (User) — об'єднана роль студента/викладача

Інтерфейс користувача є публічним, простим та інтуїтивно зрозумілим, орієнтованим виключно на перегляд інформації. Після автентифікації користувач потрапляє на персональну інформаційну панель, яка включає:

Пошук та фільтрація: Можливість швидко знайти конкретне заняття, переключитися на перегляд розкладу іншої групи або аудиторії (у режимі "публічного перегляду").

Експорт та синхронізація: Чіткі кнопки для експорту персонального розкладу у формати, зручні для користувача: PDF (для друку). Одноразове завантаження файлу або отримання посилання для синхронізації. [26][28][29]

2.4Схеми взаємодії компонентів системи

Взаємодія компонентів системи автоматизованого формування розкладу будується на архітектурі клієнт-сервер з чітко визначеними рівнями та протоколами обміну даними. На клієнтському рівні користувач взаємодіє з системою через веб-браузер, де завантажується статичний контент (HTML, CSS, JavaScript) та відбувається динамічне формування інтерфейсу на основі шаблонів Jinja2, які генеруються серверною частиною. На серверному рівні ядром є Flask-додаток, який структуровано за патерном MVC[12][20]. Всі вхідні

HTTP-запити спочатку обробляються маршрутизатором (Router), який на основі URL та методу (GET, POST) визначає, яка view-функція (Controller) має обробити запит. Контролер виконує наступну логіку: перевіряє автентифікацію користувача та його права за допомогою декораторів (@login_required, @admin_required); отримує дані з запиту (форми, JSON, параметри URL); звертається до сервісних шарів (Service Layer) або безпосередньо до моделей (Model) для виконання бізнес-логіки. Для критично важливих операцій, таких як генерація розкладу, створений окремий Сервіс Генератора (Timetable Generator Service). Цей сервіс інкапсулює алгоритм CSP/оптимізації, отримує від контролера всі необхідні дані (списки груп, викладачів, аудиторій, обмеження), виконує обчислення та повертає готовий набір розміщених занять або повідомлення про помилку. Після обробки контролер завантажує необхідні дані з бази даних через ORM[13] MySQL, який трансформує запити в SQL та конвертує результати в Python-об'єкти. Отримані дані передаються у шаблонізатор Jinja2 (View), який генерує фінальну HTML-сторінку, або, у випадку API-запиту, контролер повертає відповідь у форматі JSON. База даних є централізованим сховищем всіх станів системи. Взаємодія з нею відбувається виключно через моделі SQLAlchemy, що забезпечує абстракцію та безпеку. Схема БД включає таблиці для користувачів (з полем is_admin), груп, дисциплін, аудиторій, занять (lessons) тощо. Для операцій, що вимагають високої узгодженості (наприклад, фіксація розкладу), використовуються транзакції. Уся комунікація між компонентами внутрішньо сервера відбувається через виклики методів Python, що забезпечує високу швидкість та контроль над потоком виконання. Для зберігання тимчасових даних сесій та кешування може використовуватися Redis або подібне in-memory сховище, що прискорює роботу з часто запитуваними даними (наприклад, публічний розклад). Таким чином, система будується як набір слабко зв'язаних модулів з чіткими інтерфейсами взаємодії, що дозволяє легко тестувати, модифікувати та масштабувати окремі її частини.[15][17][18]

2.5 Висновки до третього розділу

У ході виконання даної роботи було проведено комплексний аналіз предметної області та здійснено повноцінне проєктування системи автоматизованого формування навчального розкладу, яка охоплює всі ключові етапи — від збору первинних даних до кінцевого відображення готового графіку занять. У результаті виконаних досліджень, узагальнення вимог до подібного роду програмних продуктів та послідовного застосування методів структурного й об'єктно-орієнтованого проєктування було отримано детальний, технічно пророблений опис архітектури системи. Цей опис включає вичерпний перелік функціональних вимог, які регламентують поведінку кожного модуля, а також специфікацію ключових механізмів, що відповідають за такі критично важливі операції, як перевірка відповідності розкладу системі жорстких і м'яких обмежень, розподіл аудиторного фонду, врахування індивідуальних побажань викладачів і автоматичне виявлення колізій. Загалом спроєктована система здатна забезпечити повний, безперервний життєвий цикл управління розкладом, починаючи з етапу введення первинних даних — таких як списки груп, картки викладачів, переліки дисциплін і аудиторій — і завершуючи публікацією фінальної версії розкладу з подальшою можливістю його оперативного коригування у відповідь на зміни в навчальному процесі, заміни, перенесення занять або інші непередбачувані обставини.

Розроблена система, розглянута в межах даного розділу, являє собою не лише теоретично обґрунтоване, але й практично реалізоване рішення, яке має значний потенціал для впровадження в умовах реального освітнього середовища. Її архітектура спроєктована з урахуванням сучасних принципів модульності, що робить систему масштабованою та придатною для поступового розширення від обслуговування невеликого коледжу до застосування в межах багатопрофільного університету. Система враховує реальні обмеження, що є невід'ємною частиною будь-якого навчального закладу, чітко розмежовуючи їх на жорсткі, порушення яких є категорично неприпустимим (наприклад, присутність одного викладача в

двох різних аудиторіях одночасно), та м'які, дотримання яких є бажаним, але може бути скориговане залежно від поточної ситуації (скажімо, побажання викладача уникати занять у вечірні години). Важливою складовою рішення є зручний, інтуїтивно зрозумілий інтерфейс, адаптований для різних категорій користувачів: методистів, які безпосередньо керують розкладом, викладачів, що переглядають власне навантаження та вказують побажання, а також студентів, зацікавлених у доступі до актуального графіку занять. Система також підтримує необхідні для реальної експлуатації формати експорту даних для інтеграції із зовнішніми сервісами, веб-порталами та системами електронного документообігу, а завдяки вбудованим механізмам логування всіх суттєвих дій та розсилання сповіщень забезпечується повна прозорість змін, що вносяться до розкладу.

Таким чином, спроєктована система дозволяє суттєво, іноді в рази, зменшити трудовитрати методистів, які традиційно витрачають значну кількість робочого часу на рутинне узгодження, перевірку та ручне виправлення конфліктів у розкладі. Автоматизація цих процесів мінімізує ймовірність виникнення конфліктів, пов'язаних із подвійним призначенням аудиторій або накладанням занять у викладачів, що, своєю чергою, позитивно позначається на загальній організованості навчального процесу. Крім суто технічних переваг, впровадження подібної системи має й відчутний соціальний ефект, оскільки врахування побажань педагогічного складу та студентів, рівномірний розподіл навантаження протягом тижня і відсутність незручних «вікон» сприяють підвищенню задоволеності всіх учасників освітнього процесу якістю організації їхньої спільної роботи. Це робить розроблену систему не просто технічним інструментом, а повноцінним засобом покращення внутрішнього клімату та ефективності функціонування навчального закладу в цілому.

РОЗДІЛ 4: РЕАЛІЗАЦІЯ СИСТЕМИ НА ОСНОВІ FLASK

3.1 Налаштування проекту та структура каталогів

За основу структури каталогів взятий приклад із документації роботи з фреймворком Flask. Також на майбутнє відведено місце збереження докер файлів для запуску проєкта за допомогою кросплатформеного застосунку Docker який в свою чергу дозволяє легко керувати ресурсами які витрачаються системою[16][23].

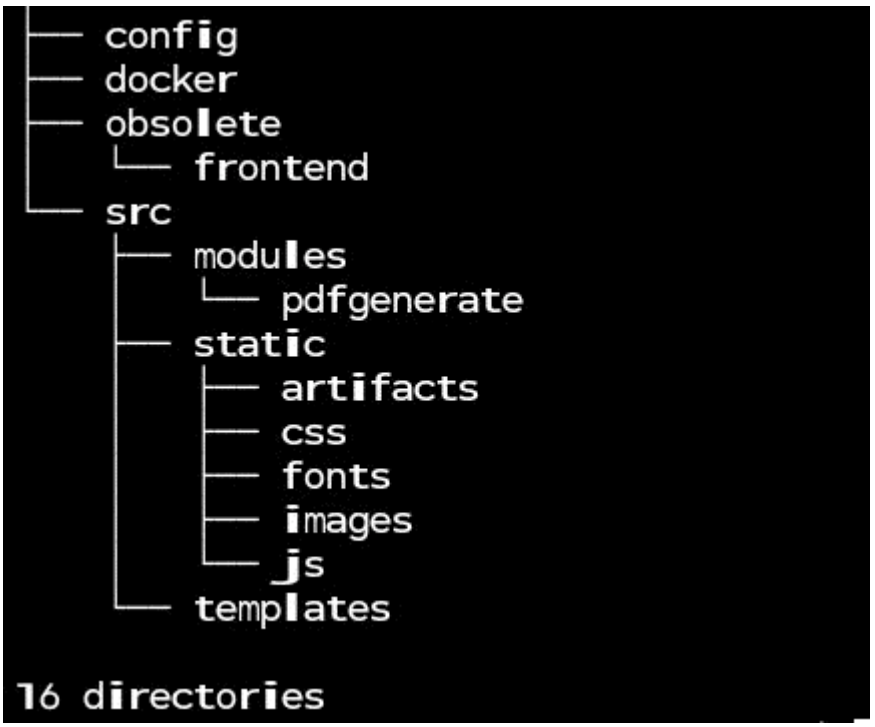


Рис. 8 — структура директорій проєкту

У корені проєкту розташовано файл README.md, який містить загальний опис системи. Поряд знаходиться директорія obsolete, призначена для застарілих компонентів фронтенду, зокрема файлів create.css та create.html, які не використовуються в актуальній версії програми.

Основний код системи зосереджено в директорії src. Точкою входу є файл app.py, який є центральним файлом і відповідає за маршрутизацію вебзастосунку. Директорія modules містить набір Python-модулів: dbConfig.py відповідає за

конфігурацію підключення до бази даних, groupedTeachers.py за логіку групування викладачів, schedClass.py визначає клас розкладу, schedGen.py містить алгоритм генерації розкладу, а teacherClass.py описує клас викладача. Окрема піддиректорія pdfgenerate всередині modules включає три модулі для роботи з PDF-документами: pdf_generator.py забезпечує створення PDF-файлів, pdf_handler.py обробляє операції з PDF, rimConverter.py, конвертує дані у внутрішній формат системи.

Директорія static містить статичні ресурси. У піддиректорії artifacts зберігаються згенеровані PDF-файли розкладів за різними інститутами та викладачами. Піддиректорія css включає файли стилів для всіх сторінок: base.css – базові стилі, dashboard.css – стилі панелі керування, generator.css – для сторінки генерації розкладу, index.css – головної сторінки, а також стилі для сторінок редагування студентів та викладачів. Піддиректорія fonts вміщує чотири шрифтові файли Times New Roman різних накреслень. У піддиректорії images знаходяться логотипи проєкту.

Директорія templates містить HTML-шаблони вебінтерфейсу: base.html є базовим шаблоном, який розширюється іншими сторінками, dashboard.html відповідає за панель керування, generator.html – за інтерфейс генерації розкладу, index.html – головну сторінку, pdf.html – сторінку перегляду PDF-файлів, students_edit.html та teachers_edit.html – сторінки редагування даних студентів і викладачів відповідно. Файл requirements.txt у корені директорії src визначає необхідні залежності Python для роботи системи.

3.2 Навіщо розбиття на файли

Розбиття проєкту на директорії та модулі є фундаментальним принципом інженерії програмного забезпечення, який застосовується для організації кодової бази незалежно від її розміру чи призначення. Коли всі файли - серверна логіка, конфігурація бази даних, стилі, скрипти, шаблони та статичні ресурси -

зберігаються в одному кореневому каталозі, така структура швидко стає непридатною для підтримки, масштабування та колективної розробки[17].

По-перше, модульна організація забезпечує розділення відповідальності, що є ключовим принципом проєктування. Кожна директорія або модуль виконує чітко визначену функцію в системі: модулі збираються в окрему папку, статичні ресурси - у свою, шаблони - у свою. Такий підхід дозволяє розробнику швидко локалізувати місце внесення змін, не переглядаючи десятки сторонніх файлів. Наприклад, якщо потрібно змінити логіку генерації PDF, розробник одразу звертається до модуля pdfgenerate, а не шукає відповідний фрагмент коду серед файлів обробки маршрутів чи взаємодії з базою даних.

По-друге, модульність сприяє повторному використанню коду. Модуль конфігурації бази даних dbConfig.py може імпортуватися в будь-яку частину системи без дублювання логіки підключення. Клас викладача, описаний у teacherClass.py, використовується як при генерації розкладу, так і при редагуванні даних викладачів. Якби всі ці визначення знаходилися в одному файлі, будь-яке їх використання вимагало б або копіювання коду, або створення складної спагеті-подібної структури з циклічними залежностями.

По-третє, розділення на директорії значно спрощує навігацію та розуміння проєкту для нового розробника, що приєднується до команди. Замість вивчення сотень рядків коду в одному каталозі, розробник отримує інтуїтивно зрозумілу структуру, де назви директорій вказують на їхнє призначення - templates для HTML, static для ресурсів, modules для бізнес-логіки. Це особливо важливо в академічному або кваліфікаційному контексті, де проєкт може передаватися від одного виконавця до іншого.

Крім того, окреме зберігання статичних ресурсів у директорії static дозволяє вебсерверу ефективно їх обслуговувати, застосовувати кешування та стиснення, що неможливо, коли всі файли змішані в одному місці. Це впливає на продуктивність застосунку. А розміщення застарілих компонентів у директорії

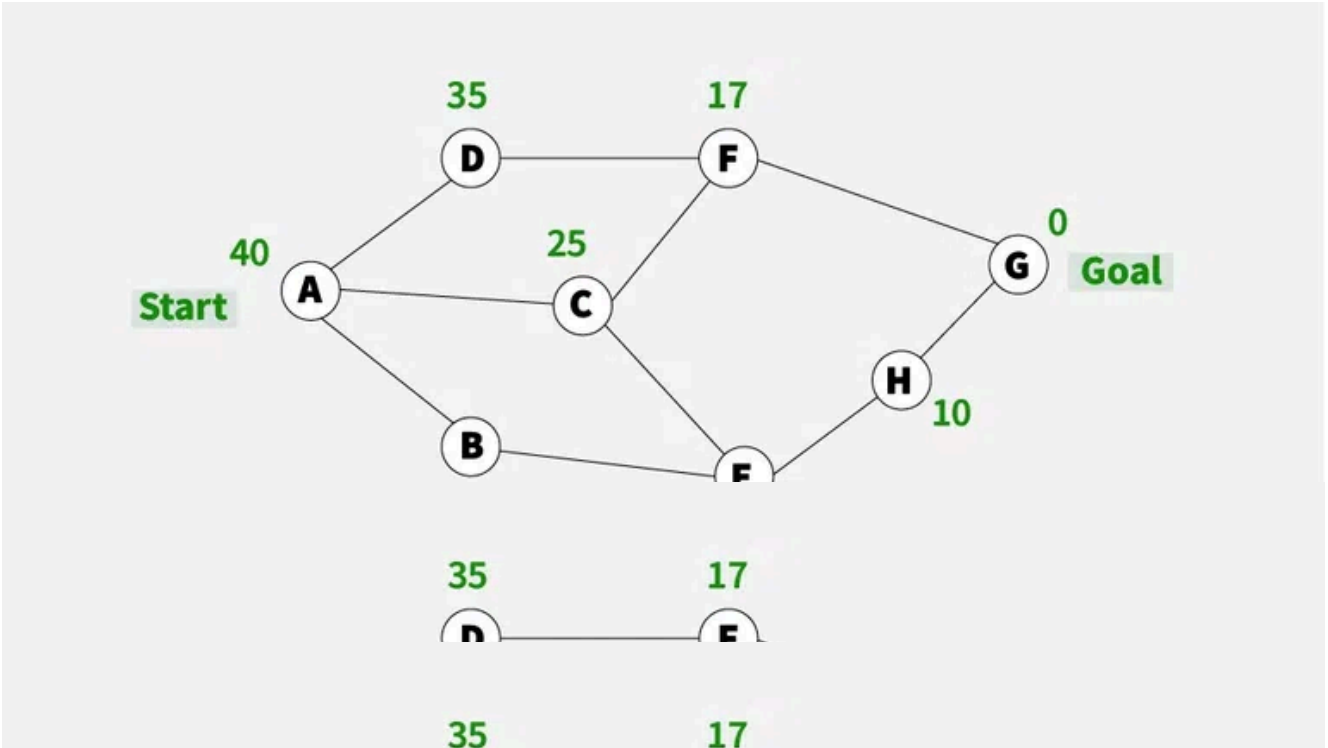
obsolete сигналізує про те, що ці файли не є частиною поточної системи, але можуть зберігатися для історичних довідок або майбутньої рефакторингу.

Нарешті, така організація полегшує автоматизацію процесів розробки. Системи контролю версій можуть ігнорувати певні директорії, інструменти збирання можуть окремо обробляти різні типи файлів, а контейнеризація (Docker) дозволяє монтувати окремі каталоги для розробки без перезбирання всього образу. Без чіткої структурної організації такі операції стають або неможливими, або надзвичайно громіздкими.

Таким чином, розбиття проєкту на директорії та модулі є не просто питанням естетики чи звички, а необхідною умовою створення підтримуваного, масштабованого та зрозумілого програмного забезпечення, особливо в контексті кваліфікаційної роботи, де проєкт має демонструвати інженерну зрілість та дотримання професійних стандартів.

3.3 Створення центральної логіки розподілення

Для початку потрібно обрати принцип по якому буде йти розподілення. Є декілька математичних алгоритмів: жадібний алгоритм, алгоритм пошуку з поверненням генетичний алгоритм, імітація відпалу, задоволення обмежень, цілочисельне лінійне. На мою думку для даної задачі найкраще підійде жадібний алгоритм.



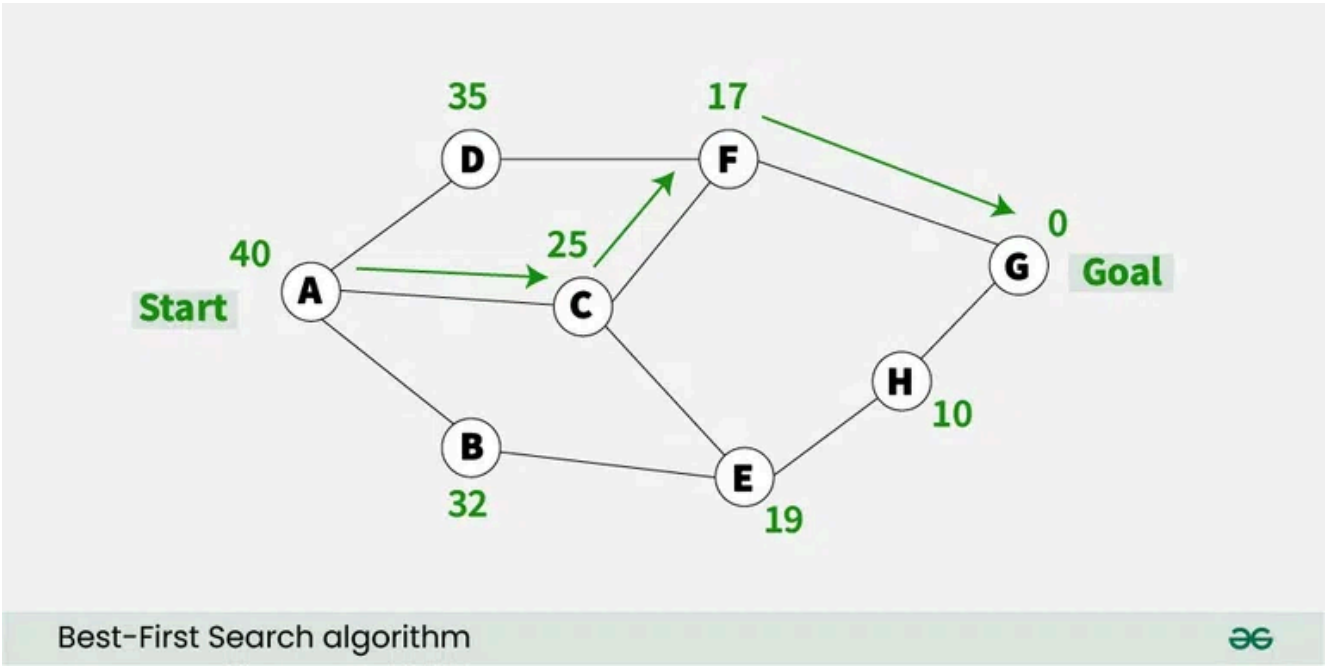


Рис. 9, 10, 11, 12 — принцип роботи жадібного алгоритму[7].

Почнемо процес зі створення об’єкту викладача. За викладачем має бути закріплено предмети які він викладає, бажаний час і чи є в нього вимога до кабінету.

```
class Teacher:
    def __init__(self, name, subjects, preferred_days=None, preferred_times=None, has_own_room=False):
        self.name = name
        self.subjects = subjects                                    # Список предметів, які викладач викладає
        self.preferred_days = preferred_days if preferred_days else []      # Бажані дні
        self.preferred_times = preferred_times if preferred_times else []   # Бажані години
        self.has_own_room = has_own_room                            # Чи є в нього власний кабінет
```

Рис. 13 - реалізація класу викладач

Далі потрібно створити метод зберігання залежності годин від предмета. Тобто зберігати в зручному форматі скільки годин на тиждень має бути певного предмету.

```
class Subject:
    def __init__(self, name, hours_per_week):
        self.name = name
        self.hours_per_week = hours_per_week                        # Скільки пар на тиждень потрібно провести
```

Рис. 14 - реалізація способу зберігання предметів

Маючи “скелет” викладача та способу зберігання данних про предмет, можна створити умовне поле з клітинками яке буде зберігати в собі інформацію про предмет в конкретний час.

```
class Schedule:
    def __init__(self):
        self.schedule = [[None for _ in range(4)] for _ in range(5)]

    def is_time_available(self, day, time):
        return self.schedule[day][time] is None

    def assign_class(self, day, time, teacher, subject):
        self.schedule[day][time] = f"{teacher.name} - {subject.name}"

    def print_schedule(self):
        days = ["Понеділок", "Вівторок", "Середа", "Четвер", "П'ятниця"]
        for i, day in enumerate(self.schedule):
            print(f"{days[i]}: {day}")
```

Рис. 15 - реалізація таблиці для розкладу

Тепер нарешті можна ствоити логіку для розподілення викладачів по комірках.

```
# Функція для генерації розкладу для викладачів з власними кабінетами
def assign_teachers_with_own_rooms(teachers, schedule):
    for teacher in teachers:
        if teacher.has_own_room:
            for subject in teacher.subjects:
                hours_assigned = 0
                for day in teacher.preferred_days:
                    for time in teacher.preferred_times:
                        if schedule.is_time_available(day, time):
                            schedule.assign_class(day, time, teacher, subject)
                            hours_assigned += 1
                        if hours_assigned >= subject.hours_per_week:
                            break
                    if hours_assigned >= subject.hours_per_week:
                        break
```

Рис. 16 - Функція для генерації розкладу для викладачів з власними кабінетами

Функція проходить по всіх викладачах і одразу відсіює тих, у кого немає власного кабінету, адже її логіка призначена саме для тих, хто прив'язаний до

конкретного місця проведення занять. Для кожного такого викладача вона далі перебирає всі предмети, які він веде, і для кожного предмета ініціалізує лічильник призначених годин, щоб відслідковувати, скільки занять уже вдалося розставити в розкладі відповідно до тижневого навантаження. Після цього починається спроба фактичного розміщення занять: функція проходить по бажаних днях викладача, а всередині кожного дня - по бажаних часових слотах. На кожній ітерації перевіряється, чи вільний відповідний слот у розкладі, і якщо так - одразу виконується призначення заняття з конкретним викладачем і предметом, а лічильник годин збільшується. Як тільки кількість призначених годин досягає необхідної норми для цього предмета за тиждень, внутрішні цикли достроково перериваються, щоб не створювати зайвих занять, і алгоритм переходить до наступного предмета або викладача, фактично заповнюючи розклад жадібним способом на основі доступних і бажаних слотів. Після проставлення викладачів із власними кабінетами, потрібно виконати проставлення всіх інших викладачів.

```
# Функція для призначення занять викладачам без власних кабінетів
def assign_other_teachers(teachers, schedule):
    for teacher in teachers:
        if not teacher.has_own_room:
            for subject in teacher.subjects:
                hours_assigned = 0
                for day in range(5):
                    for time in range(4):
                        if schedule.is_time_available(day, time):
                            schedule.assign_class(day, time, teacher, subject)
                            hours_assigned += 1
                        if hours_assigned >= subject.hours_per_week:
                            break
                    if hours_assigned >= subject.hours_per_week:
                        break
```

Рис. 17 - Функція для призначення занять викладачам без власних кабінетів

Дана функція є подібною до попередньої, але призначена лише для викладачів без власного кабінету. Дане рішення є тимчасовим і є частиною початку експериментів. Розбиття викладачів на залежних і не залежних від аудиторій є важливою частиною в розподілі. Дане розподілення дозволить

викладачам залежним від аудиторій провести і звільнити відповідні аудиторії, що в свою чергу дозволить іншим викладачам отримати більший вибір аудиторій.

Наступним кроком спробуємо створити екземпляри класів і в ручному форматі задати змінні.

```
# Викладачі
teacher1 = Teacher(name="Іваненко", subjects=[Subject("Математика", 2)], preferred_days=[1, 3], preferred_times=[1, 2], has_own_room=True)
teacher2 = Teacher(name="Петренко", subjects=[Subject("Фізика", 3)], preferred_days=[0, 2], preferred_times=[1 ,2, 3], has_own_room=True)
teacher3 = Teacher(name="Олег", subjects=[Subject("Хімія", 5)], preferred_days=[0 - 2], preferred_times=[0-3], has_own_room=False)
teacher4 = Teacher(name="Скоропадський", subjects=[Subject("фізра", 1)], preferred_days=[3,4], preferred_times=[0-3], has_own_room=False)
```

Рис. 18 – приклад вхідних данних для викладачів

Також потрібно створити таблицю.

```
schedule = Schedule()
```

Рис. 19 – створення таблиці

І після цього потрібно передати данні функціям які в свою чергу мають правильно розставити викладачів дотримуючись побажань. Для початку створимо просту одну таблицю.

```
Понеділок: ['Олег - Хімія', 'Петренко - Фізика', 'Петренко - Фізика', 'Петренко - Фізика']
Вівторок: ['Олег - Хімія', 'Іваненко - Математика', 'Іваненко - Математика', 'Олег - Хімія']
Середа: ['Олег - Хімія', 'Олег - Хімія', 'Скоропадський - фізра', None]
Четвер: [None, None, None, None]
П'ятниця: [None, None, None, None]
```

Рис. 20 – результат роботи першого тесту

Перший тестовий екземпляр було успішно створено, але ще є багато нюансів. В даному екземплярі створюється лише один розклад і якщо викладач бажає для прикладу в понеділок вийти на 2гу пару а в середу на 3тю, то дана логіка виставить його предмети двічі за день (якщо 3тя пара в понеділок вільна або 2га в середу). Також в даній версії можна об’єднати дві функції в одну.

```
def assign_teachers_with_own_rooms(teachers, schedule):
    for teacher in teachers:
        for subject in teacher.subjects:
                hours_assigned = 0
                for day in teacher.preferred_days:
                    for time in teacher.preferred_times:
                        if schedule.is_time_available(day, time):
                            schedule.assign_class(day, time, teacher, subject)
                            hours_assigned += 1
                        if hours_assigned >= subject.hours_per_week:
                            break
                    if hours_assigned >= subject.hours_per_week:
                        break
```

Рис. 21 – об’єднана функція проставлення занять

Також в оновленій версії можна помітити що викладачі передаються не по одному, а декілька за раз. Таке рішення потрібне для подальшої простішої інтеграції і розширення даної логіки. В даному випадку потрібно передавати попередньо відсортований масив з викладачами де в першу чергу будуть проставлені викладачі із залежністю від аудиторій. Для такого завдання гарно підійде метод “sort()”. Дана функція може приймати як вхідні данні масиви і сортувати за певним ключем (key)[18] який в нашому випадку є факт залежності від аудиторії.

```
Teachers = [teacher1,teacher2,teacher3,teacher4]
sTeachers = sorted(Teachers,key=lambda teacher: teacher.has_own_room,reverse=True )
```

Рис. 22 – розділення викладачів

Даний код став трохи чистішим, але все ще не вирішував посталені задачі і мав проблеми, хоча став більш читабеьлним.

Для вирішення проблеми з некоректним проставленням предметів, потрібно щоб об’єкт викладач вмів зберігати в собі данні про бажаний час. Для цього в

класі “Teachers” просто додано рекурсивний ітератор який буде читати данні викладача і коректно зберігати його побажання по годинах.

```
1  1     class Teacher:
2          -     def __init__(self, name, subjects, preferred_schedule=None, has_own_room=False):
           2     +     def __init__(self, name, subjects, preferred_schedule=None, has_own_room=False,counter = 0):
3  3                     self.name = name
4  4                     self.subjects = subjects                                        # Список предметів, які викладач викладає
5  5                     self.preferred_schedule = preferred_schedule if preferred_schedule else {}  # Словник з бажаними днями і годинами
6  6                     self.has_own_room = has_own_room                                # Чи є в нього власний кабінет
           7     +         for i in preferred_schedule.values():
           8     +             for n in i:
           9     +                 counter +=1
           10    +         self.counter = counter
```

Рис. 23 – зміни в способі зберігання викладача[19]

Також зі зміненням вхідних данних в основний блок розподілення, потрібно змінити відповідно й даний блок. Дана зміна має вирішити ще одне питання.

```
30  35      def assign_teachers_with_own_rooms(sTeachers, schedule):
31          -   teachers = sorted(sTeachers, key=lambda teacher: teacher.has_own_room, reverse=True)
        36  +   rteachers =[]
        37  +   nteachers =[]
        38  +   for i in sTeachers:
        39  +       if i.has_own_room == True:
        40  +           rteachers.append(i)
        41  +       else:
        42  +           nteachers.append(i)
        43  +   snteachers = sorted(nteachers, key=lambda teacher: teacher.counter)
        44  +   srteachers = sorted(rteachers, key=lambda teacher: teacher.counter)
        45  +   teachers = srteachers+snteachers
32      46      for teacher in teachers:
33      47          for subject in teacher.subjects:
34      48              hours_assigned = 0
35      49              for day, times in teacher.preferred_schedule.items():
36      50                  for time in times:
37      51                      if schedule.is_time_available(day, time):
38      52                          schedule.assign_class(day, time, teacher, subject)
39      53                          hours_assigned += 1
40      54                      if hours_assigned >= subject.hours_per_week:
41      55                          break
42      56                  if hours_assigned >= subject.hours_per_week:
43      57                      break
```

Рис. 24 – оновлена функція генерації розкладу[19]

Тепер даний блок коду має можливість проставляти побажання викладачів по годинах, виставляючи в приорітет викладачів із залежністю від кабінета. Також через дану зміну, потрібно передавати більше інформації про викладача.

```
61  teacher1 = Teacher(
62      name="Іваненко",
63      subjects=[Subject("Математика", 3), Subject("Бебра", 3)],
64      preferred_schedule={0: [0, 1, 2, 3], 1: [0, 3], 3: [1, 2]},
65      has_own_room=True
66  )
```

Рис. 25 – приклад передачі данних про викладача[19]

Після даної зміни потрібно перевірити правильність виконання логіки. Тому створимо викладачів які матимуть статичні предмети і бажаний час проведення пар. Скажімо крім викладача “Іваненко” є ще один викладач на ім’я “Петренко”. Нехай “Петренко” бажає проводити пари в понеділок з другої пари по четверту, у вівторок лише другу пару і третю і в середу лише другу та четверту. Також даний викладач не матиме залежності від аудиторії і має провести за тиждень чотири пари. В той час як викладач “Іваненко” (див. рис. 25) має провести шість пар за тиждень, має залежність від аудиторій і бажаний час проведення пар це понеділок з першої пари до четвертої, вівторок лише перша і четверта і в четвер другу пару й четверту. Функція розподілення проставить в першу чергу всі предмети викладача в якого є залежність від аудиторій, після чого виставить всі пари у викладача без залежності від аудиторії, в даному випадку перш за все проставляться пари у викладача “Іваненко”, а потім у викладача “Петренко”.

```
Понеділок: ['Іваненко - Математика', 'Іваненко - Математика', 'Іваненко - Математика', 'Іваненко - Бебра']
Вівторок: ['Іваненко - Бебра', 'Петренко - Фізика', 'Петренко - Фізика', 'Іваненко - Бебра']
Середа: [None, 'Петренко - Фізика', None, 'Петренко - влофіврф']
Четвер: [None, None, None, None]
П'ятниця: [None, None, None, None]
```

Рис. 26 – результат роботи оновленого коду

Тепер спробуємо прибрати бажаний час проведення пари в понеділок у “Іваненко” другу пару, таким чином один предмет даного викладача має перемістится на середу, в той час як інший викладач зможе зайняти даний час.

```
Понеділок: ['Іваненко - Математика', 'Петренко - Фізика', 'Іваненко - Математика', 'Іваненко - Математика']
Вівторок: ['Іваненко - Бебра', 'Петренко - Фізика', 'Петренко - Фізика', 'Іваненко - Бебра']
Середа: [None, 'Петренко - влофіврф', None, None]
Четвер: [None, 'Іваненко - Бебра', None, None]
П'ятниця: [None, None, None, None]
```

Рис. 27 – результат зміни побажань

Тепер є успішно згенерований один розклад який враховує побажання декількох викладачів. На даному етапі можна почати інтеграцію фреймворку flask[8].

Почнемо з самої мінімальної архітектури, просто головний файл і папка з html файлом щоб переконатися що данні можливо передавати динамічно.

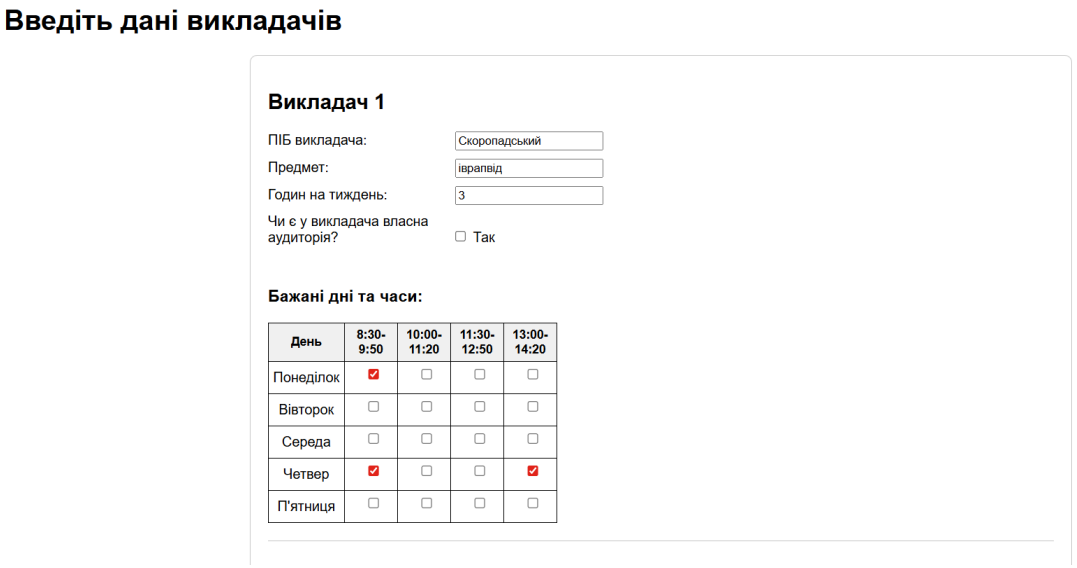


Рис. 28 – перший екземпляр додання викладача

В даному варіанті можна додати лише одного викладача якого зразу буде використано в розкладі.

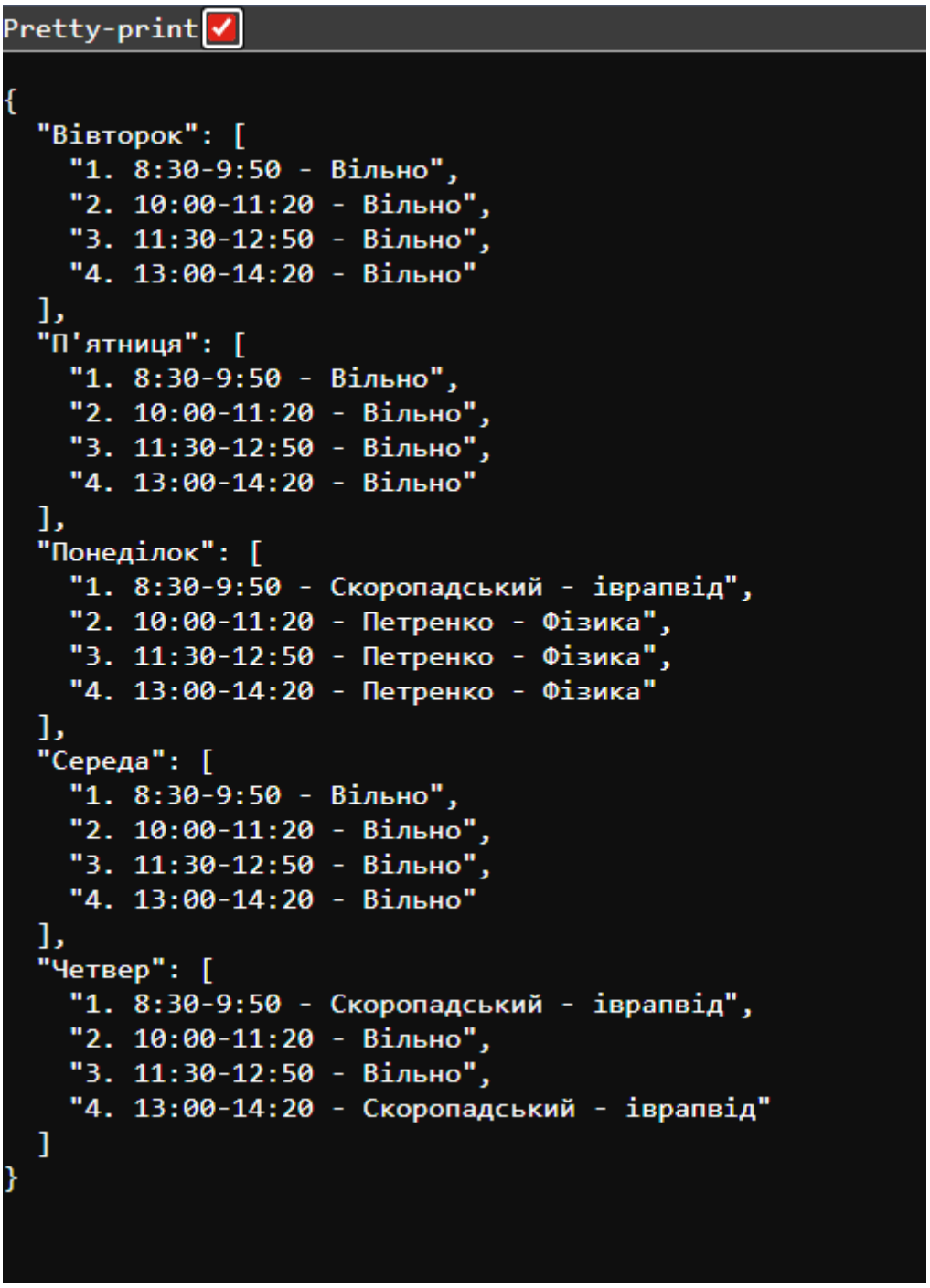


Рис. 29 – приклад розкладу

В даній версії весь код який відповідає за генерацію і переадресацію знаходиться в одному файлі, окремо лише тестовий html яким можна додати одного викладача в розклад. Тому потрібно делегувати код в окремі директорії, а в головному файлі залишити лише переадресації й обмін даними. Для прикладу

для виведення початкової сторінки потрібно лише повертати index.html, в той час як для генерації й виведення розкладу потрібно зробити запит з бази, пропустити через функцію генерації і результат передати далі. Якщо все записати в одному файлі, то з часом стане важче підтримувати продукт через мішанину різних блоків.

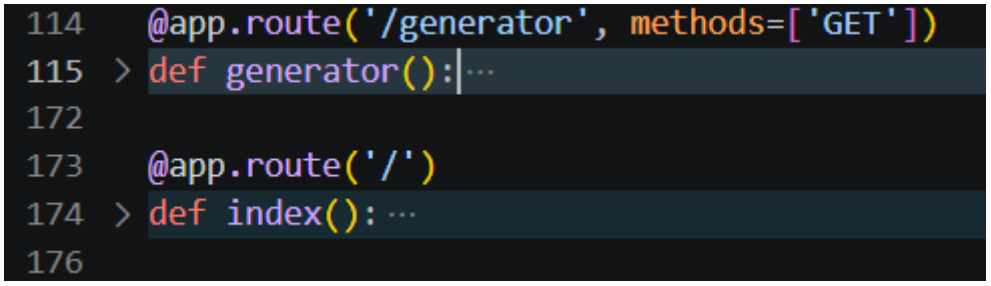


Рис. 30 – приклад запису коду

На рис. 30 видно як одна функція займає майже шість десятків рядків коду, в той час як інша лише два рядки. Тому центральна логіка генерації мігрує у файл “schedGen.py”. Даний файл відповідатиме за створення розкладу і вносити відповідні зміни у базу данних.

Також оскільки нам потрібно генерувати більше одного розклада за раз і викладач може викладати свій предмет в декількох групах, потрібно створити модуль який буде вносити зміни у базу данних відповідно до використаних занять викладача. Для початку потрібно створити тимчасовуу змінну яка буде зберігати використані данні. Через можливість створювати декілька розкладів, змінну потрібно робити за межами функції генерації. Також якщо у декількох груп предмети і викладачі співпадають, потрібно використати цю можливість як проведення однієї пари в декількох групах водночас. Відповідно до цього можна створити ще один приорітет підчас виставлення пар.

```python
def assign_teachers_with_own_rooms(sTeachers, schedule, mysql = None, isValid = True):
    usedTeachers = {}
    rteachers = []
    nteachers = []
    for i in sTeachers:
        if i.has_own_room:
            rteachers.append(i)
        else:
            nteachers.append(i)
    snteachers = sorted(nteachers, key=lambda teacher: teacher.counter)
    srteachers = sorted(rteachers, key=lambda teacher: teacher.counter)
    teachers = srteachers + snteachers
    for teacher in teachers:
        usedTeachers[teacher.name] = {}
        for subject in teacher.subjects:
            hours_assigned = 0
            to_remove = []
            for day, times in teacher.preferred_schedule.items():
                for time in times:
                    if schedule.is_time_available(day, time):
                        schedule.assign_class(day, time, teacher, subject)
                        hours_assigned += 1
                        if day not in usedTeachers[teacher.name]:
                            usedTeachers[teacher.name][day] = []
                        usedTeachers[teacher.name][day].append(time)
                        to_remove.append((day, time))
                    if hours_assigned >= subject.hours_per_week:
                        break
                if hours_assigned >= subject.hours_per_week:
                    break
            for day, time in to_remove:
                teacher.preferred_schedule[day].remove(time)
                if not teacher.preferred_schedule[day]:
                    del teacher.preferred_schedule[day]
        if isValid and mysql!= None:
            update_teacher_schedule(teacher.ID, teacher.preferred_schedule, mysql)
```

Рис. 31 – актуальна версія генерації розкладу

Також потрібен контролер який буде передавати данні для швидкого редагування використаних викладачів, таке рішення є необхідним щоб уникнути накладання пар.

```python
def update_teacher_schedule(teacher_id, updated_schedule, mysql):
    cur = mysql.connection.cursor()
    cur.execute("SELECT teacher_data FROM teachers WHERE ID = %s", (teacher_id,))
    teacher_data = cur.fetchone()[0]
    teacher_data_dict = json.loads(teacher_data)
    teacher_data_dict['schedule'] = updated_schedule
    updated_teacher_data = json.dumps(teacher_data_dict, ensure_ascii=False)
    update_query = "UPDATE teachers SET teacher_data = %s WHERE ID = %s"
    cur.execute(update_query, (updated_teacher_data, teacher_id))
    mysql.connection.commit()
    cur.close()
```

Рис. 32 – контролер оновлення використаних данних

Також відповідно до нових методів роботи з данними потрібно додати нові відповідні способи обробки данних і збереження інформації про викладачів.

```python
class Teacher:
    def __init__(self, ID, name, subjects, preferred_schedule=None, has_own_room=False,counter = 0,room_number = None):
        self.ID = ID
        self.name = name
        self.subjects = subjects                                                    # Список предметів, які викладач викладає
        self.preferred_schedule = preferred_schedule if preferred_schedule else {}  # Словник з бажаними днями і годинами
        self.has_own_room = has_own_room                                            # Чи є в нього власний кабінет
        for i in preferred_schedule.values():
            for n in i:
                counter +=1
        self.counter = counter
        self.room_number = room_number  # Номер аудиторії, якщо є власна
```

Рис. 33 – актуальний об’єкт збереження данних про викладача

Через делегацію робіт, файли з бази потрібно приводити у зручний для подальшої обробки стандарт, тому потрібно створити модуль який буде читати вхідні данні і перетворювати їх у об’єкти класу викладача. Даний модуль має приймати на вхід данні з бази данних і перетворювати їх на об’єкти класу “teacher”. Також потрібен модуль який буде створювати простіший для читання формат таблиці. Ця маленька модернізація допоможе в подальшому зручніше розуміти з яким днем тижня та в конкретно який час маємо справу.

```python
def groupofteacher(var):
    days_mapping = {
    "monday": 0,
    "tuesday": 1,
    "wednesday": 2,
    "thursday": 3,
    "friday": 4,
    "0":0,
    "1":1,
    "2":2,
    "3":3,
    "4":4
    }
    teachers = []
    for i in var:
        teacher_id = i[0]
        name = str(i[1])
        data = json.loads(i[2])
        schedule = data.get("schedule", {})
        subjects_data = data.get("subjects", {})
        room_number = True
        subjects = [Subject(subj_name, hours/1.5) for subj_name, hours in subjects_data.items()]
        numeric_schedule = {
            days_mapping[day]: hours for day, hours in schedule.items() if day in days_mapping
        }
        teacher = Teacher(
            ID=teacher_id,
            name=name,
            subjects=subjects,
            preferred_schedule=numeric_schedule,
            has_own_room=bool(room_number),
            room_number=room_number
        )
        teachers.append(teacher)
    return teachers
```

Рис. 34 – модуль для стандартизації викладачів

```python
class Schedule:
    def __init__(self):
        self.schedule = [[None for _ in range(4)] for _ in range(5)]
        self.time_slots = {
            0: "8:30-9:50",
            1: "10:00-11:20",
            2: "11:30-12:50",
            3: "13:00-14:20",
            4: "14:30-15:50"
        }

    def is_time_available(self, day, time):
        return self.schedule[day][time] is None

    def assign_class(self, day, time, teacher, subject):
        room_info = f" (Аудиторія: {teacher.room_number})" if teacher.room_number else ""
        self.schedule[day][time] = f"{teacher.name} - {subject.name}{room_info}"

    def get_schedule(self):
        days = ["Понеділок", "Вівторок", "Середа", "Четвер", "П'ятниця"]
        formatted_schedule = {}
        for i, day in enumerate(self.schedule):
            day_schedule = []
            for j, entry in enumerate(day):
                time_info = self.time_slots[j]
                if entry:
                    day_schedule.append(f"{j + 1}. {time_info} - {entry}")
                else:
                    day_schedule.append("")
            formatted_schedule[days[i]] = day_schedule
        return formatted_schedule
```

Рис. 36 – модернізована версія створення пустого розкладу

3.4 Тестування

Для початку створимо декількох викладачів

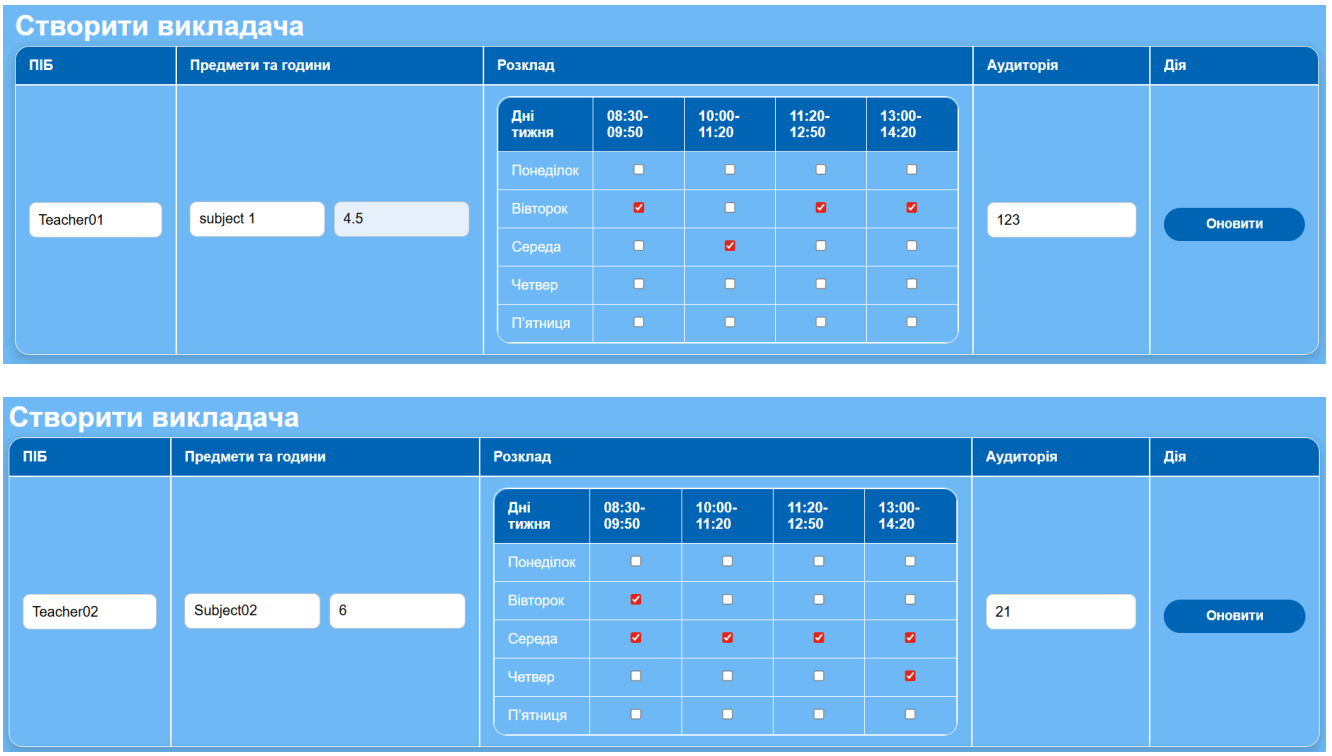


Рис. 37, 38 – тестові викладачі

Зверніть увагу, що на даному етапі потрібно вводити скільки годин потрібно виділити часу, а не скільки пар провести. Тому для проведення трьох пар потрібно 4.5 годи і для проведення чотирьох пар потрібно 6 годин.

Для початку спробуємо створити розклад для однієї групи. Нехай в першому експерименті ми бажаємо бачити двух викладачів які ведуть в одній групі.

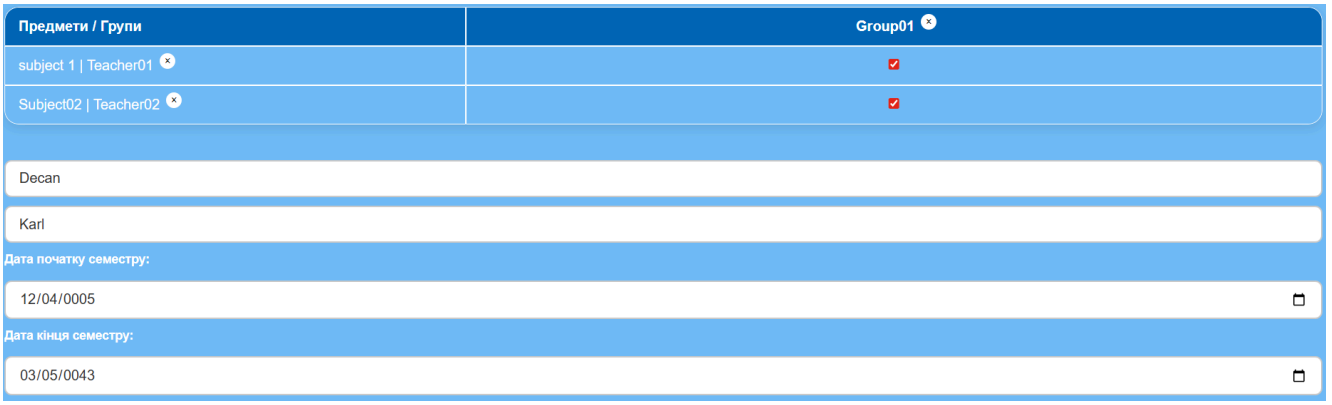


Рис. 39 – вхідні данні

Дати та імена наведені на Рис. 39 є прикладом, всі збіги випадкові і не несуть ніякої прихованої мети.

(назва інституту)

День тижня	Пара	Group01
Вівторок 07.04.0364-29.04.0064	8:30-9:50	1. 8:30-9:50 - Teacher01 - subject 1 (Аудиторія: True)
	11:30-12:50	3. 11:30-12:50 - Teacher01 - subject 1 (Аудиторія: True)
	13:00-14:20	4. 13:00-14:20 - Teacher01 - subject 1 (Аудиторія: True)
Середа 08.04.0364-30.04.0064	8:30-9:50	1. 8:30-9:50 - Teacher02 - Subject02 (Аудиторія: True)
	10:00-11:20	2. 10:00-11:20 - Teacher02 - Subject02 (Аудиторія: True)
	11:30-12:50	3. 11:30-12:50 - Teacher02 - Subject02 (Аудиторія: True)
Четвер 09.04.0364-01.05.0064	13:00-14:20	4. 13:00-14:20 - Teacher02 - Subject02 (Аудиторія: True)

Рис. 40 - результат першого тесту

Для другого тесту додамо ще одну групу. В другій групі для прикладу один викладач має провести такі самі пари.

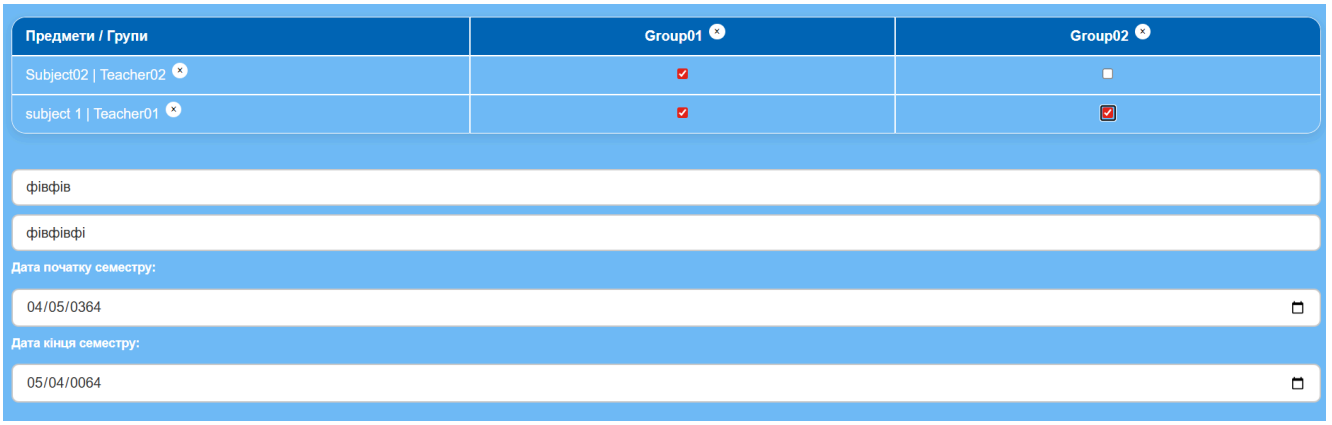


Рис. 41 – вхідні данні другого тесту

В даному випадку викладач 1 має можливість провести по три пари в обох групах водночас.

День тижня	Пара	Group01	Group02
Вівторок 07.04.0364-29.04.0064	8:30-9:50	1. 8:30-9:50 - Teacher01 - subject 1 (Аудиторія: True)	1. 8:30-9:50 - Teacher01 - subject 1 (Аудиторія: True)
	11:30-12:50	3. 11:30-12:50 - Teacher01 - subject 1 (Аудиторія: True)	3. 11:30-12:50 - Teacher01 - subject 1 (Аудиторія: True)
	13:00-14:20	4. 13:00-14:20 - Teacher01 - subject 1 (Аудиторія: True)	4. 13:00-14:20 - Teacher01 - subject 1 (Аудиторія: True)
Середа 08.04.0364-30.04.0064	8:30-9:50	1. 8:30-9:50 - Teacher02 - Subject02 (Аудиторія: True)	
	10:00-11:20	2. 10:00-11:20 - Teacher02 - Subject02 (Аудиторія: True)	
	11:30-12:50	3. 11:30-12:50 - Teacher02 - Subject02 (Аудиторія: True)	
Четвер 09.04.0364-01.05.0064	13:00-14:20	4. 13:00-14:20 - Teacher02 - Subject02 (Аудиторія: True)	

Рис. 42 – результат другого тесту

В наступному тесті пропоную створити ще одну групу і викладача.

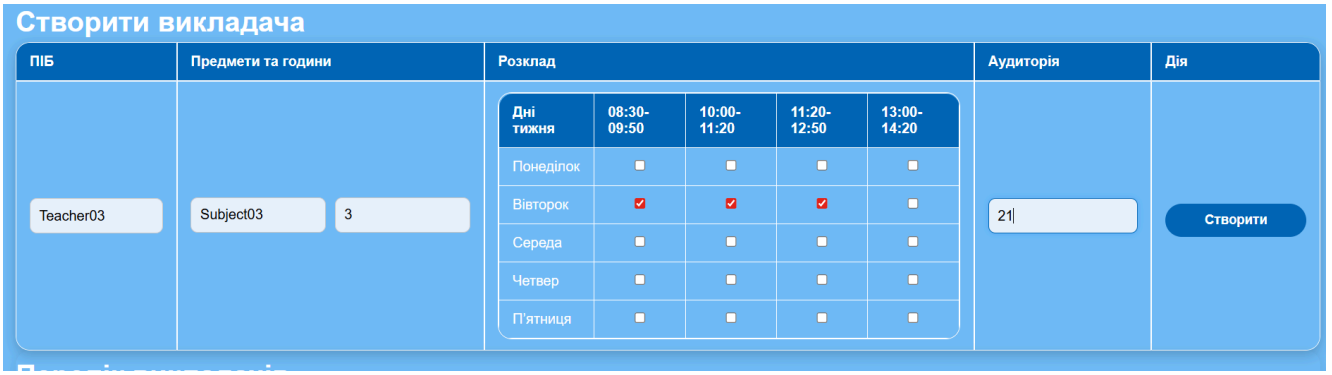


Рис. 43 – тестовий викладач

В даному експерименті нехай викладач під нгомером один проведе так само пари в перших двух групах, викладач під номером два проведе пари у всіх трьох групах і викладач під номером три проведе пари у другій і третій групі.

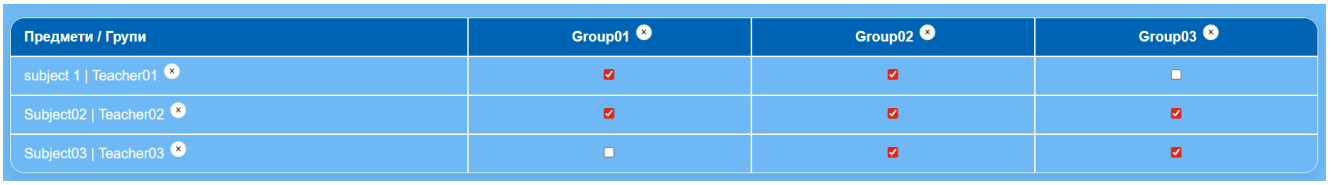


Рис. 44 – вхідні данні для третього експерименту

В даному експерименті через малу кількість доступних днів у викладача під номером три, має отримати вищій приорітет. Якщо проставити в першу чергу викладача під номером три, то він потисне викладача під номером один і змістить

його графік на інший день, що в свою чергу змістить графік викладачеві під номером два.

(назва інституту)

День тижня	Пара	Group01	Group02	Group03
Вівторок 08.04.0003-04.03.0053	8:30-9:50	1. 8:30-9:50 - Teacher01 - subject 1 (Аудиторія: True)	1. 8:30-9:50 - Teacher03 - Subject03 (Аудиторія: True)	1. 8:30-9:50 - Teacher03 - Subject03 (Аудиторія: True)
	10:00-11:20		2. 10:00-11:20 - Teacher03 - Subject03 (Аудиторія: True)	2. 10:00-11:20 - Teacher03 - Subject03 (Аудиторія: True)
	11:30-12:50	3. 11:30-12:50 - Teacher01 - subject 1 (Аудиторія: True)	3. 11:30-12:50 - Teacher01 - subject 1 (Аудиторія: True)	
	13:00-14:20	4. 13:00-14:20 - Teacher01 - subject 1 (Аудиторія: True)	4. 13:00-14:20 - Teacher01 - subject 1 (Аудиторія: True)	
Середа 09.04.0003-26.02.0053	8:30-9:50	1. 8:30-9:50 - Teacher02 - Subject02 (Аудиторія: True)	1. 8:30-9:50 - Teacher02 - Subject02 (Аудиторія: True)	1. 8:30-9:50 - Teacher02 - Subject02 (Аудиторія: True)
	10:00-11:20	2. 10:00-11:20 - Teacher02 - Subject02 (Аудиторія: True)	2. 10:00-11:20 - Teacher01 - subject 1 (Аудиторія: True)	2. 10:00-11:20 - Teacher02 - Subject02 (Аудиторія: True)
	11:30-12:50	3. 11:30-12:50 - Teacher02 - Subject02 (Аудиторія: True)	3. 11:30-12:50 - Teacher02 - Subject02 (Аудиторія: True)	3. 11:30-12:50 - Teacher02 - Subject02 (Аудиторія: True)
	13:00-14:20		4. 13:00-14:20 - Teacher02 - Subject02 (Аудиторія: True)	
Четвер 10.04.0003-27.02.0053	13:00-14:20	4. 13:00-14:20 - Teacher02 - Subject02 (Аудиторія: True)	4. 13:00-14:20 - Teacher02 - Subject02 (Аудиторія: True)	4. 13:00-14:20 - Teacher02 - Subject02 (Аудиторія: True)

Рис. 45 – результат третього експерименту

В результаті третього експерименту видно що всі умови було виконано, однакові пари в різних групах мають спільні пари і даний код враховує погодинні побажання викладачів.

Для наступного експерименту спробуємо створити розклади для різних груп. Для даного експерименту пропоную змінити біжані години проведення пар на інші.

Рис. 46, 47 – оновлені тестові викладачі

В даному випадку нехай викладач під номером два має провести пару в першій групі, натомість викладач під номером один, має провести пари в окремих розкладах.

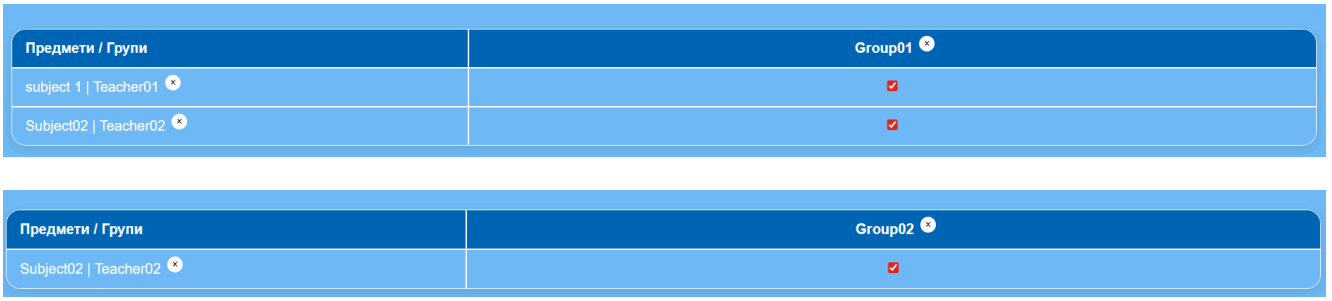


Рис. 48, 49 – вхідні данні четвертого тесту

Для початку створимо розклад для першої групи.

День тижня	Пара	Group01
Вівторок 08.04.0064-01.04.0064	8:30-9:50	1. 8:30-9:50 - Teacher02 - Subject02 (Аудиторія: True)
	10:00-11:20	2. 10:00-11:20 - Teacher01 - subject 1 (Аудиторія: True)
	11:30-12:50	3. 11:30-12:50 - Teacher01 - subject 1 (Аудиторія: True)
	13:00-14:20	4. 13:00-14:20 - Teacher01 - subject 1 (Аудиторія: True)
Середа 09.04.0064-02.04.0064	8:30-9:50	1. 8:30-9:50 - Teacher02 - Subject02 (Аудиторія: True)
	10:00-11:20	2. 10:00-11:20 - Teacher02 - Subject02 (Аудиторія: True)
	11:30-12:50	3. 11:30-12:50 - Teacher02 - Subject02 (Аудиторія: True)

Рис. 50 – перший розклад четвертого експерименту

Потім створимо розклад для іншої групи спираючись на умови описані роаніше.

День тижня	Пара	Group02
Вівторок 14.04.0065-04.04.0006	8:30-9:50	1. 8:30-9:50 - Teacher01 - subject 1 (Аудиторія: True)
Середа 08.04.0065-05.04.0006	8:30-9:50	1. 8:30-9:50 - Teacher01 - subject 1 (Аудиторія: True)
	10:00-11:20	2. 10:00-11:20 - Teacher01 - subject 1 (Аудиторія: True)

Рис. 51 – другий розклад четвертого експерименту

Як можна побачити з результатів експерименту, після першого використання викладача під номером один, данні даного викладача було оновлено в базі данних, і занесені як використані години.

3.5 Висновки до четвертого пункту

У результаті тривалого, послідовного та всебічного аналізу предметної області, який охопив як теоретичні джерела, присвячені проблематиці автоматизованого складання навчальних розкладів, так і практичне вивчення вже існуючих на ринку програмних продуктів, а також після ґрунтовного дослідження цілої низки алгоритмічних підходів, що традиційно застосовуються для вирішення даного класу задач, у межах цієї роботи було спроєктовано, детально опрацьовано і, що найважливіше, практично реалізовано у вигляді робочого програмного коду логіку, яка відповідає за один із найскладніших і найвідповідальніших етапів

функціонування системи - розподілення викладачів і навчальних пар по конкретних днях тижня із прив'язкою до визначених часових слотів. Цей етап є центральним у всьому процесі генерації розкладу, оскільки саме від коректності та виваженості рішень, прийнятих на цьому кроці, залежить загальна якість кінцевого результату, рівень задоволеності викладачів, раціональність використання аудиторного фонду та, зрештою, сама можливість організації безперебійного навчального процесу без конфліктів і накладок.

Принциповою особливістю, яка вирізняє розроблений механізм серед багатьох спрощених підходів, є те, що він працює не лише з формальними, кількісними параметрами, такими як загальна кількість годин навантаження чи місткість аудиторій, а й з обов'язковим, детальним урахуванням погодинних побажань, висловлених безпосередньо викладачами щодо найбільш зручного для них часу проведення занять протягом робочого тижня. Ідеться про те, що система не просто механічно заповнює порожні комірки, а намагається, наскільки це дозволяють жорсткі обмеження, брати до уваги суб'єктивні людські фактори: хтось із педагогів надає перевагу ранковим годинам, хтось уникає вечірніх занять, а хтось узагалі може бути присутнім у закладі лише в певні дні тижня через сумісництво або інші обставини. Уся ця інформація обробляється системою, формалізується у вигляді м'яких обмежень і стає невід'ємною частиною процесу прийняття рішень під час побудови розкладу.

Алгоритмічним ядром цього механізму виступає так званий жадібний алгоритм, вибір якого був обумовлений кількома важливими міркуваннями практичного характеру [1][7]. На відміну від методів повного перебору, які хоча й гарантують знаходження глобально оптимального рішення, проте вимагають експоненційно зростаючих обчислювальних ресурсів при збільшенні кількості вхідних даних, жадібний підхід дозволяє отримувати цілком прийнятні, близькі до оптимальних результати за значно коротший час. Суть його роботи полягає в тому, що алгоритм послідовно, крок за кроком, ітеративно заповнює доступні часові слоти, приймаючи на кожному кроці локально оптимальне рішення, виходячи з

поточного стану розкладу, який вже сформований на поточний момент. При цьому пріоритет надається найбільш обмеженим ресурсам - передусім тим викладачам, чия доступність за часом є найжорсткішою, а також тим навчальним групам, чий тижневий графік є найщільнішим і залишає мало вільного простору для маневру. Такий підхід, попри свою концептуальну простоту та відносну легкість програмної реалізації, довів свою ефективність у задачах подібного класу, де потрібно швидко знаходити нехай і не ідеальний, але цілком робочий і безконфліктний варіант розкладу.

Окрім суто теоретичного обґрунтування обраного алгоритмічного рішення, яке спирається на загальновідомі принципи побудови жадібних стратегій та аналізу їхньої складності, працездатність, коректність і практична придатність запропонованого підходу були всебічно підтверджені експериментальним шляхом. Для проведення випробувань було підготовлено серію тестових наборів вхідних даних, які моделювали доволі різноманітні ситуації, що є типовими для навчальних закладів різного масштабу - від невеликого коледжу з обмеженою кількістю викладачів і груп до відносно великого факультету з розгалуженою структурою спеціальностей, значною кількістю потоків і підгруп, а також із відчутним дефіцитом аудиторного фонду в пікові години. У ході цих експериментів було наочно і переконливо продемонстровано, на що реально здатен кінцевий програмний продукт, доведений до стадії прототипу, готового до тестування.

Передусім, система успішно генерує повноцінний, логічно несуперечливий та безконфліктний розклад занять, витрачаючи на це цілком прийнятний часовий проміжок, який не перевищує розумних меж навіть при опрацюванні кількох сотень об'єктів. Крім того, алгоритм демонструє здатність мінімізувати кількість неврахованих побажань викладачів у межах тих м'яких обмежень, які були задані на старті, що свідчить про адекватність реалізованої стратегії пріоритезації. Також під час випробувань було виявлено, що система коректно і передбачувано обробляє складні граничні ситуації, пов'язані з відчутним дефіцитом аудиторного

фонду, сигналізуючи про неможливість повного задоволення всіх запитів, але при цьому не втрачаючи працездатності. І, нарешті, окремо варто відзначити, що система адекватно й достатньо оперативно реагує на зміну вхідних параметрів - таких як додавання нових навчальних груп, зміна індивідуального графіка доступності окремого викладача або коригування обсягу годин із певної дисципліни - і здатна за потреби перебудувати раніше сформований розклад, зберігаючи при цьому його загальну структуру та не руйнуючи вщент ті частини, яких зміни не торкнулися. Таким чином, сукупність отриманих теоретичних обґрунтувань і практичних результатів свідчить про високу практичну придатність розробленого продукту до впровадження в реальних умовах та про його повну відповідність тим вимогам, які були сформульовані на початкових етапах даної роботи.

ВИСНОВКИ

Результатом виконання даної кваліфікаційної роботи є розробка спеціалізованої веб-орієнтованої системи автоматизованого формування навчального розкладу, яка здатна забезпечити повний життєвий цикл управління розкладом - від уведення первинних даних до публікації фінального графіку занять та його оперативного коригування. Розроблена система є універсальним рішенням, яке може бути адаптоване для використання в навчальних закладах різного масштабу, проте вона висуває певні вимоги до вхідних даних: інформація про викладачів, групи, дисципліни та аудиторний фонд має бути повною, несуперечливою та попередньо структурованою для внесення до системи. У результаті роботи система формує безконфліктний тижневий розклад занять із прив'язкою до конкретних днів, часових слотів та аудиторій, враховуючи при цьому як жорсткі обмеження (неможливість присутності одного викладача у двох місцях одночасно, місткість приміщень), так і м'які побажання викладачів щодо зручного для них часу проведення занять.

Реалізацію системи було виконано за допомогою мови програмування Python із використанням мікрофреймворку Flask, який забезпечив необхідний баланс між простотою розробки та гнучкістю архітектури, дозволивши зосередитись на реалізації складної бізнес-логіки, зокрема алгоритмів оптимізації розкладу [8][15]. Для формування динамічних веб-сторінок та відображення згенерованого розкладу в зручному табличному вигляді було застосовано шаблонізатор Jinja2 [14]. У якості системи керування базами даних було обрано реляційну СКБД MySQL, яка гарантує надійність, цілісність та ефективну обробку складних запитів у багатокористувацькому середовищі, а на етапі прототипування також використовувалася SQLite як легковагова альтернатива для локального тестування [9][10]. Для спрощення розгортання системи та забезпечення стабільності її роботи передбачено використання контейнеризації через Docker та розміщення на віртуальному приватному сервері [11][16]. Контроль версій початкового коду здійснювався за допомогою платформи GitHub [19][21].

Центральним компонентом системи, розробленим у межах даної роботи, є алгоритмічна логіка розподілу викладачів і навчальних пар по днях тижня, в основу якої покладено жадібний алгоритм, що на кожному кроці приймає локально оптимальне рішення, виходячи з поточного стану розкладу та пріоритезації найбільш обмежених ресурсів [1][7]. Експериментальним шляхом на серії тестових наборів даних, що моделювали реальні ситуації, було підтверджено здатність системи генерувати повноцінний безконфліктний розклад за прийнятний час, адекватно реагувати на зміну вхідних параметрів та мінімізувати кількість неврахованих побажань.

Під час написання кваліфікаційної роботи були виконані наступні завдання:

- проаналізовано літературні джерела та сучасні наукові публікації, присвячені проблематиці автоматизованого складання навчальних розкладів [1][2];
- виконано аналіз існуючих програмних засобів для формування розкладу, зокрема Schedulebuilder, TimetableMaster, aSc Timetables, та виявлено їхні обмеження щодо врахування погодинних побажань викладачів і глибокої інтеграції з наявними базами даних [3][5][6];
- виконано аналіз сучасних технологій та інструментів для розробки веб-орієнтованих систем, обґрунтовано вибір мови Python, фреймворку Flask та СКБД MySQL [8][10][15];
- виконано аналіз алгоритмічних підходів до задач складання розкладів, обґрунтовано вибір жадібного алгоритму як такого, що забезпечує прийнятну якість результату без експоненційного зростання обчислювальних витрат [1][7];
- розроблено архітектуру системи з використанням патерну MVC, що забезпечує модульність, масштабованість і зручність подальшої підтримки [12];

- розроблено зручний веб-інтерфейс для різних категорій користувачів - методистів, викладачів та студентів;
- реалізовано логіку автоматичного розподілу викладачів і пар по днях тижня з урахуванням погодинних побажань, обмежень аудиторного фонду та навчального навантаження;
- розроблено систему автоматизованого формування навчального розкладу, яка є теоретично обґрунтованим, практично реалізованим і масштабованим рішенням для навчальних закладів.

Подальшим розвитком системи є доповнення її додатковими модулями, а саме модулем інтеграції з існуючими в навчальних закладах реляційними базами даних для усунення необхідності ручного введення інформації про викладачів, групи та дисципліни, що суттєво підвищить зручність впровадження. Також перспективним напрямом розвитку є вдосконалення алгоритмічного ядра шляхом доповнення жадібного алгоритму елементами метаевристичних методів оптимізації або генетичних алгоритмів, що дозволить покращити якість кінцевого розкладу в умовах великої кількості м'яких обмежень і підвищити загальну задоволеність викладачів. Крім того, доцільним є розширення функціональності системи модулем для автоматичного сповіщення учасників навчального процесу про зміни в розкладі через месенджери або електронну пошту, а також створення мобільного застосунку для оперативного доступу студентів і викладачів до актуального графіку занять.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Wikipedia - https://www.wikipedia.org/
2. ResearchGate - https://www.researchgate.net
3. Schedulebuilder (Офіційний сайт застосунку) - https://schedulebuilder.org/
4. Canva (Офіційний сайт застосунку) - https://www.canva.com/
5. Timetablemaster (Офіційний сайт застосунку) - https://www.timetablemaster.com/
6. aSc Timetables (Офіційний сайт застосунку) - https://www.asctimetables.com/
7. Geeksforgeeks (Спеціалізований форум) - https://www.geeksforgeeks.org/
8. Palletsprojects (Документація фреймворка flask) - https://flask.palletsprojects.com/en/stable/
9. SQLite (Інформація про базу данних sqlite)- https://sqlite.org/
10. MySQL (Інформація про базу данних mysql)- https://www.mysql.com/
11. VPS (Інформація про віртуальні приватні сервери)- https://hostpro.ua/en/vps/
12. MVC (Інформація про патерн MVC)- https://dotnet.microsoft.com/en-us/apps/aspnet/mvc
13. Stack overflow (Спеціалізований форум)- https://stackoverflow.com/
14. Jinja2 (документація на фреймворк)- https://jinja.palletsprojects.com/en/stable/
15. Python (мова програмування) - https://www.python.org/
16. Docker (застосунок для додаткової зручності) - https://www.docker.com/

17. FreeCodeCamp (артиклі, туторіали та книги) - https://www.freecodecamp.org/

18. Python HOWTO (Офіційний посібник python) - https://docs.python.org/uk/3.9/howto/index.html

19. GIThub (платформа для керування версіями) - https://github.com/

20. M. Fowler Patterns of Enterprise Application Architecture (опис стандартної архітектури пз та патерн MVC)

21. Git book (документація про Git) - https://git-scm.com/docs

22. Google OR-Tools Documentation (опис технічних деталей SCP) - https://developers.google.com/optimization/cp

23. Docker: Lightweight Linux Containers for Consistent Development and Deployment (переваги використання docker) - https://www.seltzer.com/margo/teaching/CS508.19/papers/merkel14.pdf

24. D. Hellmann (2017). The Python 3 Standard Library by Example. (книга про стандарті бібліотеки python) - https://github.com/bindrat/PYTHON/blob/main/The%20Python%203%20Standard%20Library%20by%20Example%20(%20PDFDrive%20).pdf

25. Python Full Course for Beginners (базовий курс по python) - https://www.youtube.com/watch?v=K5KVEU3aaeQ

26. Kavun, S., etc. Acknowledgement to Reviewers of Sustainability in 2019, Sustainability; Basel, Volume 12, No. 2, (2020): 742. doi:10.3390/su12020742. https://www.proquest.com/docview/2443211140. Indicators-Markers for Assessment of Probability of Insurance Companies Relatedness in Implementation of Risk-Oriented Approach

27. Kavun, Sergii, Multiple Data-Driven Missing Imputation for Wind Energy Datasets. Preprint article. Available at SSRN: https://ssrn.com/abstract=4524725 or http://dx.doi.org/10.2139/ssrn.4524725

28. Kavun, Sergii, Multiple Data-Driven Missing Imputation. Preprint article. Available at SSRN: https://arxiv.org/abs/2507.03061

29. S. Kavun and A. Zamula "Jobs Distribution Task in Computer Networks," 2022 IEEE 9th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T), Kharkiv, Ukraine, 2022, pp. 502-506, doi: 10.1109/PICST57299.2022.10238554 https://ieeexplore.ieee.org/document/10238554/ [Scopus]
30. Kavun, S., Levkin, D., Levkin, A., Kotko, Y., Levkina, R. Methods of Mathematical Programming for Designing a Safe Environment for Bioobject, Proceedings of the Cybersecurity Providing in Information and Telecommunication Systems II (CPITS-II 2023) co-located with International Conference on Problems of Infocommunications. Science and Technology (PICST 2023), Kyiv, Ukraine, October 26, 2023 (online), pp. 255-260, CEUR-WS Vol. 3550, October 26, 2023 (URN: urn:nbn:de:0074-3550-0, EID: 2-s2.0-85178352393, DBLP: conf/cpits/cpits2023) https://ceur-ws.org/Vol-3550/short9.pdf [Scopus, Q4]
31. Kavun, S., Levkina, R., Kotko, Y., Levkin, D., Levkin, A. Information Security in Project Management for the Financial and Budgetary Capacity of the National Economy, Proceedings of the Cybersecurity Providing in Information and Telecommunication Systems II (CPITS-II 2023) co-located with International Conference on Problems of Infocommunications. Science and Technology (PICST 2023), Kyiv, Ukraine, October 26, 2023 (online), pp. 246-254, CEUR-WS Vol. 3550, October 26, 2023 (URN: urn:nbn:de:0074-3550-0, EID: 2-s2.0-85178357070, DBLP: conf/cpits/cpits2023) https://ceur-ws.org/Vol-3550/short9.pdf [Scopus, Q4]
32. S. Kavun and A. Zamula, "Exploratory Data Analysis in Wind Energy Datasets," 2023 IEEE 4th KhPI Week on Advanced Technology (KhPIWeek), Kharkiv, Ukraine, 2023, pp. 1-6, doi: 10.1109/KhPIWeek61412.2023.10312958. https://ieeexplore.ieee.org/document/10312958 [Scopus]

33. Pavlyshyn, I., Petrenko, A., Opryshko, B., Oliinyk, B., Kavun, S. (2024). Social Media Impact on the ‘Cosmos’ Blockchain Ecosystem: State and Prospect. Data Science Journal, 23(1), p. 8. https://doi.org/10.5334/dsj-2024-008 [Scopus, Q1]
34. Kavun, Sergii, Theory: Multidimensional Space of Events. Preprint article. Available at SSRN: https://ssrn.com/abstract=4944538 or http://dx.doi.org/10.2139/ssrn.4944538 https://ssrn.com/abstract=5187140 or https://arxiv.org/pdf/2505.11566
35. Correction: Pavlyshyn, I., Petrenko, A., Opryshko, B., Oliinyk, B., Kavun, S., & Vasylchuk S. (2024). Social Media Impact on the ‘Cosmos’ Blockchain Ecosystem: State and Prospect. Data Science Journal, 23(1), p. 17. https://doi.org/10.5334/dsj-2024-017 [Scopus, Q1]