МІНІСТЕРСТВО ОСВІТИ НАУКИ УКРАЇНИ

СТРУКТУРНИЙ ПІДРОЗДІЛ «ФАХОВИЙ КОЛЕДЖ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ПРАТ «ПВНЗ «ЗІЕІТ»

Циклова комісія з інформаційних технологій

ДО ЗАХИСТУ ДОПУЩЕНА

Голова циклової комісії,

спеціаліст в/к

_____ С.О. Сабанов

ВИПУСКНА РОБОТА МОЛОШОГО СПЕЦІАЛІСТА

РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ, ЩО ДОЗВОЛЯЄ АВТОМАТИЗУВАТИ ПРОЦЕС СКЛАДАННЯ НАВЧАЛЬНОГО РОЗКЛАДУ

Виконав

ст. гр. ІПЗ – 119к9 _____ Д.Р. Ванін

Керівник

викладач _____ К.С. Суха

Запоріжжя

2023

СТРУКТУРНИЙ ПІДРОЗДІЛ «ФАХОВИЙ КОЛЕДЖ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ПРАТ «ПВНЗ «ЗІЕІТ»

Циклова комісія з інформаційних технологій

ЗАТВЕРДЖУЮ

Голова циклової комісії,

спеціаліст в/к

_____ С.О. Сабанов

17 січня 2023 р.

З А В Д А Н Н Я

НА ВИПУСКНУ РОБОТУ МОЛОДШОГО СПЕЦІАЛІСТА

студенту гр. ІПЗ-119К9 ,

спеціальності 121 - «Інженерія програмного забезпечення»

Ванін Данило Руславнович

1.Тема: Розробка програмного продукту, що дозволяє автоматизувати процес складання навчального розкладу

затверджена наказом № 09.2 – 19 від 04 березня 2023 р.

2. Термін здачі студентом закінченої роботи: 24 червня 2023 р.

3. Перелік питань, що підлягають розробці:

1. Провести огляд літератури, що присвячена тематиці досліджень

…….

.

8. Оформити звіт за результатами роботи

4. Календарний графік підготовки випускної роботи молодшого спеціаліста

№ етапу	Зміст	Терміни виконання	Готовність по графіку %, підпис керівника	Підпис керівника про повну готовність етапу, дата
1	Формулювання (корегування) теми випускної роботи молодшого спеціаліста та збір практичного матеріалу за темою випускної роботи	16.01.23-17.02.23		
2	**І атестація** І розділ випускної роботи молодшого спеціаліста	27.03.23-01.04.23		
3	**ІІ атестація** ІІ розділ випускної роботи молодшого спеціаліста	01.05.23-06.05.23		
4	**ІІІ атестація** ІІІ розділ випускної роботи молодшого спеціаліста, висновки та рекомендації, додатки, реферат	29.05.23-03.06.23		
5	Перевірка випускної роботи молодшого спеціаліста програмою «Антиплагіат»	15.05.23-12.06.23		
6	**Доопрацювання випускної роботи молодшого спеціаліста, підготовка презентації, отримання відгуку керівника і рецензії**	05.06.23-10.06.23		
7	**Попередній захист випускної роботи молодшого спеціаліста**	**12.06.23-18.06.23**		
8	Подача випускної роботи молодшого спеціаліста на кафедру	*за 3 дні до захисту*		
9	**Захист випускної роботи молодшого спеціаліста**	**19.06.23-24.06.23**		

Керівник _____ К.С. Суха

(підпис) (ініціали, прізвище)

«____» _____ 2023 р.

Завдання отримав до виконання _____ Д.Р. Ванін

(підпис студента) (ініціали, прізвище)

«____» _____ 2023 р.

РЕФЕРАТ

Випускна робота молодшого спеціаліста містить 109 сторінок, шість таблиць, 32 рисунки, 10 лістингів, 30 бібліографічних посилань, один додаток.

Метою роботи є розробка програмного продукту, що автоматизує процес складання навчального розкладу, використовуючи сучасні технології програмування.

Об'єктом дослідження є процеси складання навчального розкладу в навчальних закладах, а також використання алгоритмічних методів для їх оптимізації.

Предметом дослідження є програмний продукт, який має бути розроблений для автоматизації цих процесів, зокрема за допомогою генетичного алгоритму.

Проведено глибокий аналіз тематичної області та сучасних аналогів програмного продукту. Висновок дослідження підтвердив, що створення програмного рішення для автоматизації процесу складання навчального розкладу актуальне та важливе. Проект було успішно втілено в життя за допомогою технологій C#, Visual Studio і бази даних MS Access. Було ретельно спроектовано структуру бази даних, а також написано основні модулі та алгоритми.

Розроблений продукт надає користувачам інтуїтивно зрозумілий і зручний інтерфейс, який в свою чергу, допомагає зекономити час, мінімізуючи необхідність в ручному формуванні розкладу. Додаток може автоматично складати розклад, враховуючи ряд критичних факторів, таких як навантаження викладачів, груп студентів та доступність аудиторій.

НАВЧАЛЬНИЙ РОЗКЛАД, АВТОМАТИЗАЦІЯ, ГЕНЕТИЧНИЙ АЛГОРИТМ, C#, MS ACCESS, ВИСОКОРІВНЕВЕ ПРОГРАМУВАННЯ, ОПТИМІЗАЦІЯ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ 7

ВСТУП 8

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ Й ПОСТАНОВКА ЗАДАЧІ 9

1.1 Дослідження проблем складання розкладу 9

1.2 Аналіз існуючих систем 12

1.3 Аналіз основних методів складання навчального розкладу 20

1.4 Постановка задачі 23

1.5 Висновок 24

2 ВИБІР ТЕХНОЛОГІЙ ПРОЕКТУВАННЯ ТА ОСОБЛИВОСТІ АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 25

2.1 Вибір технологій 25

2.1.1 Мова програмування 25

2.1.2 Середовище розробки 27

2.2 Аналіз вимог. Use-case діаграми. Основні прецеденти 28

2.3 Архітектура проекту 35

2.3.1 Особливості розробки рівня BLL 36

2.3.2 Особливості розробки DAL 38

2.3.3 Особливості розробки рівня UI 39

2.4 Висновок 42

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ 43

3.1 Проєктування бази даних 43

3.2 Розробка алгоритму додатку 45

3.3 Розробка основних модулів 46

3.4 Функціональне та модульне тестування 54
3.5 Інструкція користувачеві програми 58
3.5.1 Вимоги до апаратного та програмного забезпечення 58
3.5.2 Використання програмного продукту 59
3.6 Висновок 71
ВИСНОВКИ 73
ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 74
Додаток А. Лістинги програми 78

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ADGO.NET – ActiveX Data Object для .NET

API – Application Programmine Interface

BLL – business logic layer

CIL – Common Intermediate Language

CLR – Common Language Runtime

DAL – Data Access Layer

DB - Data Base

DI - Dependency Injection

ERD – Entity Relationship Diagram

IL – Intermediate Language

JIT – Just In Time

SQL – Structured Query Language

UI – User Interface

VS – Visual Studio

WCF – Windows Communication Foundation

WF – Windows Forms

ВСТУП

Сучасний світ неможливо уявити без комп'ютерних технологій, що проникли у всі сфери людської діяльності. Освітній сектор не є виключенням. Автоматизація процесів стала важливою частиною управління навчальними закладами, починаючи від вишів та закінчуючи школами.

Однією з найбільших проблем, з якими зіштовхуються освітні заклади, є складання навчального розкладу. Традиційно цей процес здійснюється вручну, вимагає величезних зусиль та часу, а його результативність часто залишає бажати кращого. Врахування всіх змінних (кількості груп, викладачів, дисциплін, аудиторій, особливостей робочого часу тощо) представляє собою складну задачу, що ставить під сумнів ефективність ручного методу.

Актуальність вибраної теми обумовлена зростаючою потребою в оптимізації роботи освітніх установ та покращенні якості навчального процесу.

Мета дипломної роботи - розробити ефективний інструмент для автоматизації складання навчального розкладу, який здатний враховувати множину параметрів та забезпечувати гнучкість налаштувань. Завданнями даної роботи є: аналіз існуючих систем автоматизації складання розкладу, визначення їх переваг та недоліків, проектування власної системи з урахуванням виявлених проблем, розробка програмного продукту, тестування та впровадження системи в реальному освітньому закладі.

Практична значущість отриманих результатів полягає в значному полегшенні та прискоренні процесу складання навчального розкладу, вдосконаленні якості навчального процесу за рахунок оптимального розподілу навчального навантаження та ресурсів, а також зниженні ризику помилок та конфліктів у розкладі. Впровадження такого програмного продукту може стати важливим кроком на шляху до цифровізації освітнього процесу та його оптимізації.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ Й ПОСТАНОВКА ЗАДАЧІ

1.1 Дослідження проблем складання розкладу

У сфері вищої освіти (ЗВО) виникає складна проблема розкладу занять, що передбачає призначення конкретних часових проміжків та аудиторій для забезпечення ефективного та якісного навчального процесу. Складання розкладу в ЗВО є важливим завданням, що стикається з рядом обмежень та вимог, пов'язаних з наявними ресурсами, персоналом, студентським контингентом та дисциплінами. Ця проблема полягає у пошуку оптимального рішення, яке б задовольнило всі вимоги та обмеження, враховуючи наявні ресурси та інтереси всіх зацікавлених сторін[1].

При складанні розкладу занять у закладах вищої освіти (ЗВО), виникають низка обмежень та вимог, які необхідно враховувати. Серед цих обмежень та вимог можна відзначити наступне:

– обмеження на кількість доступних аудиторій та їхню місткість, що обмежує можливість проведення занять одночасно в різних групах;

– обмеження на кількість викладачів та їхнє навантаження, оскільки кожен викладач має фіксований робочий час та обмежену кількість годин, які він може викладати;

– обмеження на кількість груп студентів та їхній розмір, оскільки необхідно забезпечити ефективну та комфортну навчальну атмосферу для кожного студента;

– вимоги до послідовності викладання дисциплін, зокрема практичних та лабораторних занять, щоб забезпечити логічну та послідовну структуру навчального процесу;

– вимоги до рівномірного розподілу занять протягом навчального тижня, щоб уникнути перевантаження студентів або викладачів у певні дні або часові періоди;

– вимоги до забезпечення часу на самопідготовку та проведення додаткових занять, щоб студенти мали достатньо часу для усвідомленого засвоєння навчального матеріалу;

– врахування особистих побажань студентів та викладачів щодо розкладу занять, оскільки їхні індивідуальні потреби та переваги також мають бути враховані для створення оптимального навчального середовища [2].

Складання розкладу занять у закладах вищої освіти (ЗВО) представляє собою складну задачу, оскільки вимагає забезпечення оптимального вирішення широкого спектру вимог та обмежень, враховуючи взаємозв'язки та взаємовпливи між різними параметрами розкладу. Ця задача має значну кількість можливих рішень, серед яких потрібно знайти оптимальне або найбільш прийнятне.

Для складання розкладів в ЗВО запропоновано різні методи та алгоритми. Традиційні підходи базуються на експертному методі, де розклад складається вручну або за допомогою простих програмних засобів, що забезпечують задоволення основних обмежень та вимог. Однак такі методи не завжди гарантують досягнення оптимального розкладу з точки зору ефективності та задоволення потреб всіх зацікавлених сторін.

У контексті автоматизації процесу складання розкладу та підвищення якості отриманих рішень, були розроблені комп'ютерні методи та алгоритми, такі як жадібні алгоритми, місцевий пошук, побудова та модифікація графів. Однак, виявилося, що ці підходи мають свої обмеження, зокрема у врахуванні всіх вимог та обмежень, або можуть застрягти в локальних оптимумах.

Останнім часом дослідники пропонують застосовувати метаевристичні алгоритми, такі як генетичні алгоритми, частково оптимізовані рої, мурашині алгоритми та імітацію відпалу, для розв'язання проблеми складання розкладу в ЗВО. Ці алгоритми мають здатність знаходити оптимальні або близькі до оптимальних рішень в складних та багатопараметричних просторах пошуку,

а також адаптуватися до змінних умов та вимог. Використання метаевристичних алгоритмів виявляється перспективним підходом для покращення ефективності та якості розкладів у ЗВО [3].

Незважаючи на значні досягнення у розробці методів та алгоритмів для складання розкладу в закладах вищої освіти (ЗВО), існують певні виклики та проблеми, які залишаються актуальними. До них можна віднести наступне:

– потреба в підвищенні адаптивності та гнучкості алгоритмів для різних умов та вимог, з урахуванням особливостей окремих ЗВО. Кожен заклад має свою унікальну структуру, ресурси та вимоги, тому необхідно розробляти методи, які здатні ефективно пристосовуватись до специфіки конкретного ЗВО;

– важливість врахування особистих побажань та потреб студентів та викладачів під час формування розкладу. Кожен учасник навчального процесу може мати власні пріоритети та обмеження, тому важливо розробляти методи, які забезпечують урахування індивідуальних вимог та забезпечують задоволення потреб кожного студента та викладача;

– потреба у розвитку методів та алгоритмів для динамічного оновлення розкладу у випадку зміни обмежень, ресурсів або вимог протягом навчального процесу. Розклад занять може піддаватись змінам через різноманітні фактори, і необхідно мати ефективні методи, які здатні швидко адаптуватись та оновлювати розклад у відповідь на нові умови та обставини;

– інтеграція розроблених алгоритмів з іншими системами управління та інформаційно-аналітичними інструментами з метою підвищення ефективності навчального процесу. Розклад занять є частиною більш широкої системи управління ЗВО, і необхідно розробляти інтегровані рішення, які сприяють оптимізації та покращенню роботи всієї системи [4].

У зв'язку з викликами та перспективами, перед нами постає актуальна та перспективна задача розробки нових методів та алгоритмів для автоматизованого складання розкладу в закладах вищої освіти (ЗВО) з

використанням генетичних алгоритмів, що враховують всі вимоги та обмеження. Впровадження таких методів та алгоритмів може сприяти покращенню якості навчання, забезпеченню рівномірного розподілу навантаження на студентів та викладачів, а також оптимізації використання ресурсів та зниженню витрат на організацію навчального процесу. Потенційні переваги включають покращення ефективності, забезпечення більш гнучкого та адаптивного розкладу, а також задоволення індивідуальних потреб та побажань учасників навчального процесу. Продовження досліджень у цьому напрямку може призвести до значних покращень у вирішенні проблем складання розкладу в ЗВО [5].

1.2 Аналіз існуючих систем

Перед розробкою програмного продукту необхідно провести дослідження, спрямоване на аналіз існуючих аналогічних рішень, доступних на ринку. Цей процес сприяє розумінню наявних можливостей і виявленню областей для подальшого вдосконалення, щоб створити продукт, який відповідає потребам користувачів. Аналіз аналогічних програмних рішень дозволяє з'ясувати, які функціональні можливості повинен мати новий продукт, щоб бути конкурентоспроможним на ринку.

Prime Timetable

Prime Timetable – це програмний продукт, створений з метою забезпечення конкурентоспроможності на ринку. Він надає ефективний та зручний інструмент для автоматизації процесу формування розкладу, забезпечуючи оптимальні результати.

Основна архітектура Prime Timetable базується на комбінації вибіркових алгоритмів та метаевристик. Застосунок використовує важливі параметри та обмеження, такі як наявні ресурси, кількість аудиторій, викладачів та студентів, а також послідовність викладання дисциплін [6].

На рис. 1.1 зображено графічний інтерфейс продукту Prime Timetable.

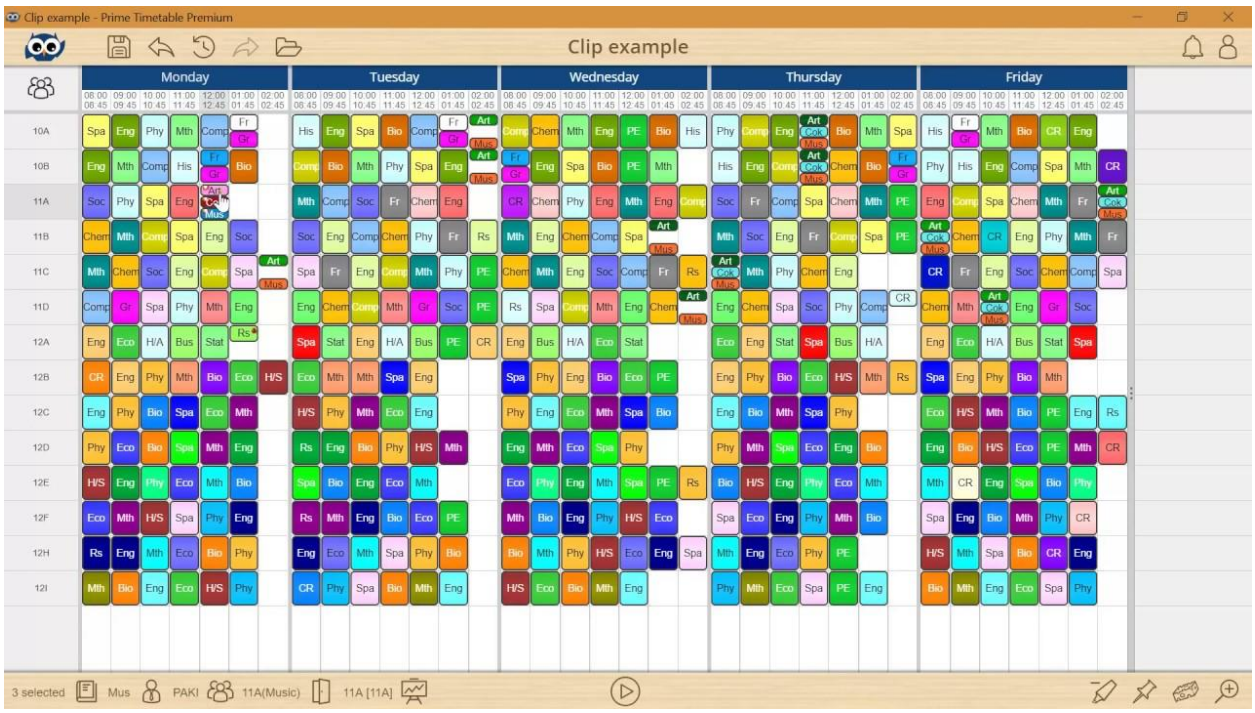


Рисунок 1.1 – Інтерфейс користувача «Prime Timetable»

Prime Timetable має ряд основних функцій. Серед них:

– складання розкладу. Застосунок дозволяє вводити необхідну інформацію, таку як дисципліни, викладачі, групи студентів, аудиторії, обмеження та вимоги. На основі цих даних він автоматично генерує оптимальний розклад занять;

– редагування та оптимізація. Надає можливість вносити зміни в розклад, змінювати розподіл занять, розміщення аудиторій тощо. Крім того, він пропонує функцію оптимізації, яка дозволяє вдосконалити розклад з метою задоволення вимог та покращення його ефективності;

– перегляд та друк. Застосунок надає зручний інтерфейс для перегляду розкладу занять у зручному графічному форматі. Також є можливість роздрукувати розклад для зручного використання в паперовому вигляді.

Переваги Prime Timetable включають:

– ефективність. Застосунок пропонує швидкий та ефективний процес складання розкладу занять. Використання вибіркових алгоритмів та

метаевристик дозволяє досягти оптимальних результатів у складанні розкладу;

– гнучкість. Дозволяє користувачеві налаштовувати розклад з урахуванням різних вимог, обмежень та особистих налаштувань;

– оптимізація. Застосунок надає функцію оптимізації розкладу, яка дозволяє покращити його ефективність та задовольнити вимоги стосовно навантаження студентів, викладачів та розподілу ресурсів.

Недоліки Prime Timetable:

– вимоги до введення даних. Для досягнення оптимального розкладу потрібно ввести велику кількість даних, таких як розклади дисциплін, викладачі, групи студентів та аудиторії;

– залежність від якості вхідних даних. Як у багатьох програмах для складання розкладу, які базуються на алгоритмах, які здатні генерувати різні варіанти розкладу, які можуть бути неоптимальними або не задовольняти всім вимогам;

– обмеження функціональності. В деяких випадках Prime Timetable може мати обмежену функціональність, що не дозволяє задовольнити всім особливим потребам конкретного навчального закладу.

Отже, Prime Timetable є ефективним і зручним інструментом для автоматизації процесу складання розкладу занять в освітніх установах. Він пропонує широкі можливості для врахування обмежень, вимог та особистих налаштувань. Застосунок використовує вибіркові алгоритми та метаевристики для досягнення оптимального розкладу. Незважаючи на потребу у введенні великої кількості даних та залежність від якості вхідних даних, Prime Timetable допомагає покращити ефективність та гнучкість навчального процесу. Проте, слід враховувати його обмежену функціональність та необхідність належного налаштування для відповідності унікальним потребам кожного навчального закладу[7].

Mimosa Scheduling Software

Mimosa Scheduling Software - це програмний продукт, розроблений з метою автоматизації процесу складання розкладу занять (рис. 1.2). Його призначенням є автоматизація процесу планування розкладу в освітніх установах та інших сферах діяльності, де важливо ефективно розподіляти часові ресурси.

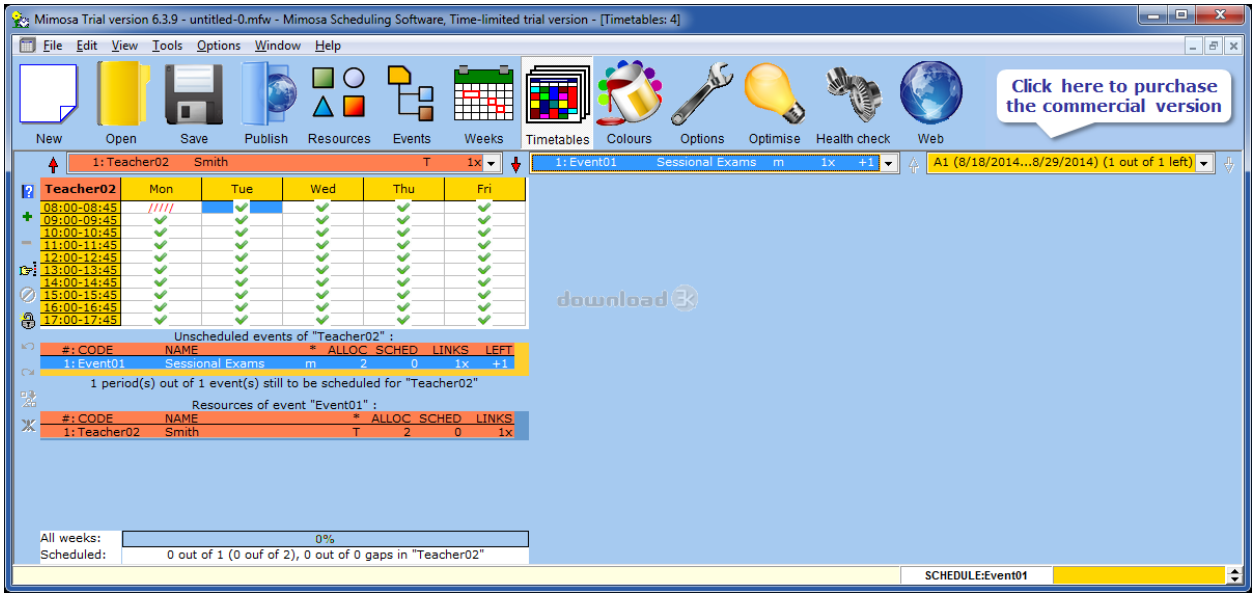


Рисунок 1.2 – Інтерфейс користувача «Mimosa Scheduling Software»

Архітектура Mimosa Scheduling Software базується на клієнт-серверній моделі, де клієнтська частина дозволяє користувачам вводити необхідні дані та налаштування, а серверна частина виконує обчислення та генерацію розкладу занять [8].

Серед основних функцій Mimosa Scheduling Software можна виділити:

– введення даних. Застосунок надає можливість вводити основні дані, такі як дисципліни, викладачі, групи студентів та доступні аудиторії;

– налаштування параметрів. Користувач може налаштовувати різні параметри, такі як обмеження на кількість занять на день, розподіл часових проміжків, врахування перерв між заняттями та інші;

– автоматична генерація розкладу. Застосунок використовує алгоритми та оптимізаційні методи для автоматичної генерації розкладу занять, забезпечуючи оптимальний розподіл ресурсів та враховуючи задані обмеження.

– перегляд та експорт розкладу. Користувач може переглядати та аналізувати згенерований розклад у зручному графічному форматі.

Переваги Mimosa Scheduling Software:

– ефективність. Застосунок пропонує швидкий та ефективний процес складання розкладу занять. Використання оптимізаційних алгоритмів допомагає досягти оптимального розподілу ресурсів та задовольнити вимоги та обмеження;

– гнучкість. Дозволяє користувачеві налаштовувати різні параметри розкладу, враховуючи особливості конкретного навчального закладу або організації;

– графічне відображення. Застосунок надає зручний графічний інтерфейс для перегляду та аналізу розкладу занять.

Недоліки Mimosa Scheduling Software:

– складність використання. Застосунок може мати складний інтерфейс та вимагати певного рівня технічних навичок для ефективного використання;

– обмежена функціональність. В порівнянні з деякими іншими програмними рішеннями, Mimosa Scheduling Software може мати обмежену функціональність;

– залежність від якості вхідних даних. Як і в будь-якому програмному продукті, якість розкладу та результати можуть залежати від якості та точності введених вихідних даних.

Отже, Mimosa Scheduling Software є ефективним і гнучким інструментом для складання розкладу занять. Він пропонує широкі можливості для оптимізації розподілу ресурсів та врахування вимог та обмежень. Застосунок надає зручний інтерфейс для перегляду та аналізу розкладу, забезпечуючи візуалізацію часових проміжків. Хоча він вимогливий у використанні та має обмежену функціональність, проте залишається цінним інструментом для оптимізації процесу планування розкладу занять у навчальних закладах та інших сферах діяльності [9].

Celcat Timetabler

Celcat Timetabler - це програмний продукт, створений для автоматизації процесу складання розкладу занять в освітніх установах (рис. 1.3). Він надає розширені можливості для ефективного та автоматизованого планування розкладу, забезпечуючи оптимальне використання ресурсів та враховуючи різноманітні вимоги та обмеження.

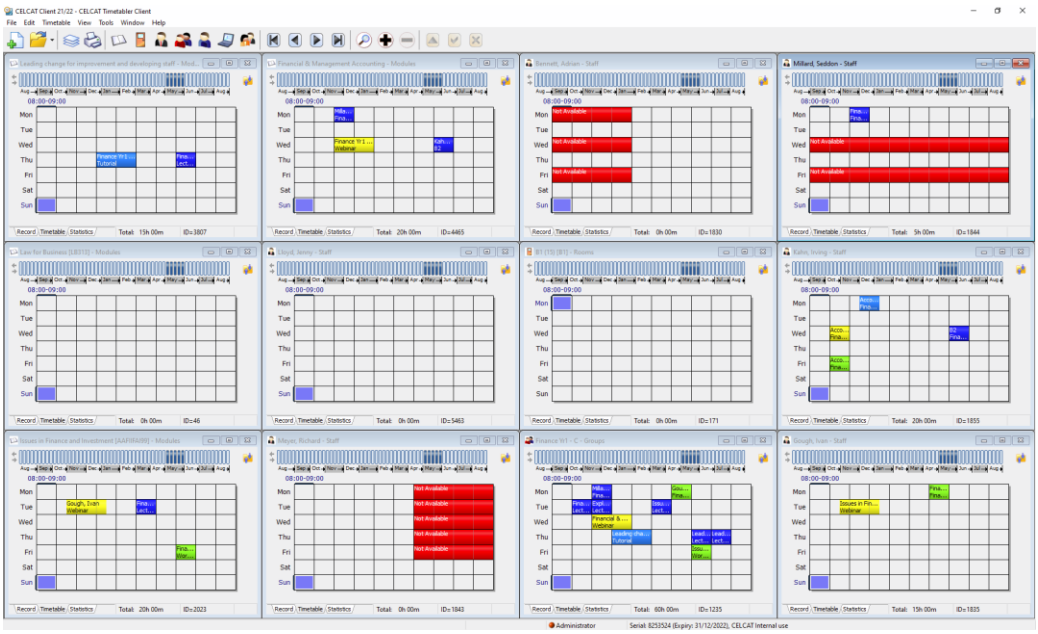


Рисунок 1.3 – Інтерфейс користувача «Prime Timetable»

Основна архітектура Celcat Timetabler базується на клієнт-серверній моделі, де серверна частина виконує обчислення та генерацію розкладу, а клієнтська частина надає інтерфейс для введення даних та настройки параметрів.

Серед основних функцій Celcat Timetabler можна виділити:

– введення даних. Застосунок дозволяє користувачам вводити необхідну інформацію, таку як розклади дисциплін, викладачі, групи студентів, аудиторії та інші важливі дані;

– автоматичне планування. За допомогою розширених алгоритмів та оптимізаційних методів, Celcat Timetabler автоматично генерує розклад занять, враховуючи різні обмеження, такі як кількість аудиторій, викладачів, перерви між заняттями та інші;

– гнучкі налаштування. Застосунок надає можливість налаштовувати різні параметри розкладу, враховуючи особливості конкретного навчального закладу або організації;

– перегляд та експорт розкладу: Користувачі можуть легко переглядати та аналізувати згенерований розклад занять у зручному графічному форматі [10].

Переваги Celcat Timetabler:

– ефективність. Застосунок забезпечує швидке та ефективне планування розкладу занять, використовуючи розширені алгоритми та оптимізаційні методи;

– гнучкість. Надає користувачам можливість налаштовувати різні параметри розкладу, що дозволяє створювати розклад, відповідний конкретним потребам та вимогам користувача;

– графічний інтерфейс. Застосунок має зручний графічний інтерфейс, що спрощує перегляд, аналіз та візуалізацію розкладу занять.

Недоліки Celcat Timetabler:

– складність використання. Застосунок може бути складним у використанні, особливо для нових користувачів;

– вартість. Має значну вартість в порівнянні з іншими програмними рішеннями для планування розкладу занять. Це може бути обмеженням для невеликих освітніх установ або організацій з обмеженим бюджетом;

– залежність від вхідних даних. Результати розкладу та його якість значно залежать від якості та точності введених вхідних даних. Некоректні або неповні дані можуть призвести до неточностей або незадовільного розкладу.

В цілому, Celcat Timetabler є потужним інструментом для планування розкладу занять зі значними перевагами в ефективності та гнучкості, але варто враховувати його складність використання та вартість [11].

У таблиці 1.1 проведено порівняння розглянутих програмних рішень у за загальними характеристикам.

Таблиця 1.1 – Аналіз програмних рішень

Характеристика	Prime Timetable	Mimosa Scheduling Software	Celcat Timetabler
Ефективність	+++	+++	+++
Гнучкість	++	++	+++
Графічний інтерфейс	+++	+++	+++
Складність використання	++	++	+
Вартість	++	++	+
Залежність від вхідних даних	++	++	++
Сумісність з іншими системами	++	+	+++
Підтримка користувачів	+++	++	++
Масштабованість	++	++	+++

Проведений аналіз застосунків Prime Timetable, Mimosa Scheduling Software і Celcat Timetabler показав, що всі вони є ефективними інструментами для складання розкладу занять. Кожен застосунок має свої переваги та обмеження, і вибір між ними залежить від конкретних потреб та пріоритетів організації.

Prime Timetable відрізняється простотою використання та зручним графічним інтерфейсом, а Mimosa Scheduling Software має потужні алгоритми для автоматичного планування розкладу. Celcat Timetabler, з свого боку, пропонує гнучкі настройки та високу масштабованість.

Крім того, сумісність з іншими системами, рівень підтримки користувачів та вартість також є важливими факторами при виборі підходящого рішення.

Розробка подібної системи є обґрунтованою з огляду на значущість планування розкладу занять та управління ресурсами в контексті освітніх закладів та компаній. Така система має потенціал знизити затрати часу на процес створення та оновлення розкладу, сприяти більш ефективному використанню ресурсів та мінімізувати кількість помилок. Крім того, наявність такої системи може забезпечити прозорість та доступність процесу планування для всіх зацікавлених сторін.

1.3 Аналіз основних методів складання навчального розкладу

Задача формування оптимального розкладу занять виникла протягом тривалого часу, від початку існування навчальних закладів. В ході пошуку розв'язків для цієї складної та комплексної задачі було розроблено різноманітні підходи, проте їхні обмеження часто ускладнюють досягнення повного розв'язання цього завдання.

Серед алгоритмів, які використовуються для розв'язання задачі формування розкладу, можна виділити кілька основних підходів:

– алгоритми теорії графів. Ці алгоритми спрямовані на знаходження оптимальних розкладів, де конфлікти та перекриття занять мінімізовані;

– Імовірнісний алгоритм. Використовує статистичні методи та ймовірнісні моделі. Заняття розподіляються за допомогою випадкового вибору, а потім перевіряється відповідність отриманого розкладу заданим обмеженням;

– генетичний алгоритм. Моделює процес природного відбору для пошуку оптимального розкладу. Він використовує популяцію рішень (розкладів) та оператори генетичного алгоритму для еволюції та покращення цих рішень з покоління в покоління [12].

Алгоритми теорії графів

Теорія графів є одним з ключових напрямків дослідження в області дискретної математики та комп'ютерних наук, яка розглядає графи — структури даних, що складаються з вузлів і з'єднань між ними (країв). Алгоритми теорії графів — це математичні та комп'ютерні методики, розроблені для вирішення задач, пов'язаних з графами.

Одними з найважливіших класів алгоритмів теорії графів є алгоритми обходу графа. Ці алгоритми, такі як алгоритми Дейкстри та Беллмана-Форда, застосовуються для знаходження найкоротших шляхів в графах, в той час як алгоритми в глибину та в ширину використовуються для проходження графа [13].

Інший важливий клас алгоритмів пов'язаний з оптимізацією. Такі алгоритми, як Прима, Крускала та Борувки, використовуються для знаходження мінімальних остовних дерев графів. Ці алгоритми мають важливе застосування в областях, таких як мережеве планування та проектування інфраструктури.

Ще один клас алгоритмів теорії графів включає алгоритми для вирішення задач комівояжера, задач кольору графа, та ін. Вони часто включають методики комбінаторної оптимізації та метаевристичні алгоритми, такі як симуляція відпалу або генетичні алгоритми.

Незважаючи на свою складність, алгоритми теорії графів використовуються в широкому спектрі практичних застосувань, включаючи обчислювальну біологію, соціальні мережі, телекомунікації, транспортну логістику та багато інших галузей [14].

Імовірнісні алгоритми

Імовірнісні алгоритми, відомі також як рандомізовані алгоритми, використовують випадковість або випадкові числа у своєму робочому процесі з метою розв'язання обчислювальних проблем. Ці алгоритми розглядаються важливим інструментом в теорії обчислень та прикладній інформатиці, оскільки вони часто дозволяють розв'язувати задачі, які важко або неможливо розв'язати за допомогою детермінованих алгоритмів.

Імовірнісні алгоритми часто класифікуються на дві основні категорії: алгоритми Лас-Вегасу і алгоритми Монте-Карло [15].

Алгоритми Лас-Вегасу завжди видають правильний відповідь, але час їх виконання є випадковим. Вони використовуються в ситуаціях, коли важливо отримати точний відповідь, а час виконання може бути гнучким. Прикладом може бути алгоритм Рабина-Міллера для перевірки простоти числа.

З іншого боку, алгоритми Монте-Карло мають фіксований час виконання, але вони можуть давати неправильні відповіді з деякою ймовірністю. Ці алгоритми використовуються в ситуаціях, де швидкість є критичною, а відповідь може бути приблизною. Прикладом може бути метод Монте-Карло для оцінки числа π.

Важливо зауважити, що імовірнісні алгоритми не гарантують 100% точності, але вони зазвичай можуть дати достатньо точну відповідь для багатьох практичних застосувань [16].

Генетичний алгоритм

Генетичні алгоритми відносяться до класу метаевристичних алгоритмів, які застосовують принципи натуральної еволюції, такі як наслідування, мутація, селекція, та рекомбінація (або схрещування), для вирішення оптимізаційних та пошукових проблем.

В основі генетичного алгоритму лежить концепція «хромосоми», яка представляє можливе рішення проблеми та є набором «генів», кожен з яких відповідає конкретному аспекту рішення. В рамках алгоритму, популяція хромосом еволюціонує через покоління за допомогою операторів селекції, схрещування та мутації.

Генетичні алгоритми виявилися дуже ефективними для розв'язання складних оптимізаційних проблем, включаючи задачі маршрутизації, розкладу, а також проблеми у галузі машинного навчання та біоінформатики [17].

1.4 Постановка задачі

Тема даної роботи присвячена проектуванню автоматизованої системи для генерації розкладу занять. Головною ціллю роботи є розробка інструментарію, який є ефективним, гнучким та зручним у використанні для максимальної оптимізації процесу створення розкладу занять в освітніх установах. Для розв'язання задач, викладених в даній роботі, вибрано генетичні алгоритми як основний математичний інструмент, оскільки вони показали ефективність у рішенні складних оптимізаційних проблем.

Вхідна інформація

Для ефективного проектування та реалізації системи генерації розкладу занять, вхідні дані повинні включати наступні елементи:

– деталізований перелік навчальних груп, включаючи їх кількість та спеціалізації;

– інформацію про викладацький склад, що включає кількість викладачів, їх спеціалізації та обсяг роботи;

– перелік предметів, включаючи їх кількість, а також відповідні групи та викладачі, які їх викладають;

– дані про аудиторії, їх кількість та вмістимість;

– відомості про тривалість навчальних сесій та кількість занять на тиждень для кожної навчальної групи;

– обмеження, що стосуються годин проведення занять.

Вихідна інформація

Вихідними даними розробленої системи слугує оптимізований розклад занять для всіх груп освітньої установи, який створено з урахуванням усіх вхідних параметрів та обмежень. Даний розклад надається у формі організованої таблиці, в якій відображено дати проведення занять, час початку та завершення, локації (аудиторії), а також відповідних викладачів для кожного конкретного заняття.

Система, що розроблюється, повинна задовольняти наступні критерії:

– створення користувацького інтерфейсу, що сприяє легкому введенню вхідних даних і конфігурації параметрів;

– використання всіх необхідних обмежень та вимог, що стосуються процесу планування розкладу;

– здатність системи адаптуватися та масштабуватися відповідно до різних розмірів та структур академічних установ.

Щодо технологічних інструментів, які планується використати для реалізації цієї системи автоматизованої генерації розкладу занять за допомогою генетичних алгоритмів:

– мова програмування C# буде використана, вона надає обширні можливості для виконання генетичних алгоритмів, обробки даних та створення користувацького інтерфейсу; [18]

– система керування базами даних MS Access буде застосована для зберігання та обробки вхідних даних та результатів [19].

1.5 Висновок

В даному розділі було проаналізовано предметну область та поставлено задачу створення автоматизованої системи генерації розкладу занять, яка використовує генетичні алгоритми. Виявлено ключові проблеми, що виникають при розробці розкладу, які включають врахування численних вимог та обмежень.

Проведено аналіз наявних на ринку систем генерації розкладу та визначено їхні переваги та недоліки. Це важливе дослідження дозволяє визначити, які функції мають бути застосовані в новій системі.

Оцінено основні методи складання розкладу. На основі даного аналізу, було вирішено використати генетичні алгоритми, які виявилися найбільш ефективними для вирішення складних оптимізаційних задач, типових для планування розкладу.

2 ВИБІР ТЕХНОЛОГІЙ ПРОЕКТУВАННЯ ТА ОСОБЛИВОСТІ АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вибір технологій

2.1.1 Мова програмування

З урахуванням потреби досягнення високої продуктивності, масштабованості та сумісності з різноманітними платформами, наступні мови програмування можуть бути вважатися належним вибором: C#, C++ та Java.

Мова програмування C# є об'єктно-орієнтованою мовою, розробленою корпорацією Microsoft. Ця мова була створена з метою розробки програм для платформи .NET, яка забезпечує можливість виконання програм на різних операційних системах. C# має синтаксис, схожий на мову програмування C++, але відрізняється вищим рівнем абстракції, що дозволяє зменшити обсяг коду, необхідного для вирішення поставлених завдань. Крім цього, C# підтримує різні парадигми програмування, такі як процедурне програмування, функціональне програмування та інші [20].

Мова програмування C++ представляє собою універсальну мову програмування з високим рівнем абстракції, що була розроблена як розширення мови C з метою надання можливостей об'єктно-орієнтованого програмування та інших функціональних можливостей. Ця мова знаходить широке застосування у розробці різноманітних програм, включаючи операційні системи, відеоігри, банківські системи та інші сфери використання. Завдяки підтримці та багатому набору функцій, C++ дозволяє розробникам ефективно використовувати пам'ять та системні ресурси з метою досягнення високої продуктивності та оптимальної роботи програмних за стосунків [21].

Мова програмування Java представляє собою об'єктно-орієнтовану мову програмування, розроблену компанією Sun Microsystems [22]. Однією з визначальних особливостей Java є високий рівень переносимості, що дозволяє програмам, написаним на цій мові, виконуватися на різних операційних системах та пристроях. Java широко використовується у розробці різноманітних програм, зокрема веб-додатків, мобільних додатків, сервісів хмарних технологій та інших областей застосування. Особлива увага була приділена вбудованій підтримці багатопоточності, мережевих додатків та інших функцій у мові Java, що дозволяє розробникам ефективно використовувати ресурси комп'ютера для досягнення оптимальної продуктивності [23].

У табл. 2.1 наведено табличний порівняльний аналіз функціональних можливостей кожної з розглянутих мов програмування.

Таблиця 2.1 – Порівняльний аналіз мов програмування

Характеристика	C#	Java	C++
Парадигми програмування	Об'єктно-орієнтоване, процедурне, функціональне	Об'єктно-орієнтоване, процедурне, функціональне	Об'єктно-орієнтоване, процедурне, функціональне
Переносимість	Висока, завдяки платформі .NET	Низька, оскільки код має компілюватись для кожної операційної системи окремо	Висока, завдяки вбудованій віртуальній машині Java
Використання пам'яті	Автоматична збірка сміття	Ручне керування пам'яттю	Автоматична збірка сміття
Швидкодія	Середня	Висока	Середня

На підставі проведеного аналізу, можна сказати, що мова програмування C# є відмінним вибором для розробки програм з високим рівнем складності та вимог до безпеки. Розгляд такого вибору обґрунтовується наявністю платформи .NET, підтримкою об'єктно-орієнтованого підходу, вбудованою підтримкою мультипоточності, здатністю розробляти веб-додатки з підтримкою мультипоточності, доступністю вихідного коду та підтримкою великої спільноти.

2.1.2 Середовище розробки

Середовища розробки відкривають перед розробниками можливості створення, налагодження та тестування програм у зручному та ефективному середовищі. У випадку розробки застосунку, що дозволяє автоматизувати процес складання навчального розкладу можна розглянути наступні середовища розробки:

– Visual Studio - це потужне інтегроване середовище розробки, створене компанією Microsoft. Воно надає комплексну підтримку для розробки програм на різноманітних мовах програмування, включаючи C#, C++, Java, Python та інші. Visual Studio включає в себе набір високоякісних інструментів для розробки, які дозволяють програмістам писати код, відлагоджувати, тестувати та розгортати різні типи програмного забезпечення - від простих консольних додатків до складних веб-сервісів і веб-додатків. [24];

– Eclipse. IDE, розроблене фондом Eclipse, спрямоване на розробку програм на мові програмування Java та інших мовах. Воно пропонує широкий спектр інструментів для розробки, налагодження та тестування програм [25];

– PyCharm. IDE, створене компанією JetBrains, яке спеціалізується на розробці програм на мові програмування Python. Воно також надає

різноманітні інструменти для розробки, налагодження та тестування програм, написаних на мові Python [26].

Таблиця 2.2 – Порівняльний аналіз середовищ розробки

Характеристика	Visual Studio	Eclipse	PyCharm
Мови програмування	C#, C++, Java	Java	Python
Платформи	Windows	Cross-OS	Cross-OS
Підтримка Git	Так	Так	Так
Вбудований дебагер	Так	Так	Так
Редагування коду	Так	Так	Так
Аналіз коду	Так	Так	Так
Підтримка тестування	Так	Так	Так
Підтримка мультиплатформи	Ні	Так	Так
Візуалізація даних	Так	Ні	Так

У випадку розробки застосунку, вибір був зроблений на користь Visual Studio, оскільки це інтегроване середовище розробки, яке забезпечує оптимальну та надійну розробку програм на мові C#. Крім того, Visual Studio має розширений набір інструментів, що сприяє ефективній розробці, налагодженню та тестуванню програм.

2.2 Аналіз вимог. Use-case діаграми. Основні прецеденти

З метою докладного опису функціональності розробленої системи та забезпечення зрозуміння її використання користувачами, використання діаграм прецедентів виявляється важливим та потужним інструментом. Кожен прецедент може розглядатися як окремий сервіс, що надається системою та визначає, як користувач може взаємодіяти з системою з метою досягнення конкретного результату. Діаграма прецедентів дозволяє виділити та описати поведінку системи з точки зору користувачів, уточнити вимоги до

системи та забезпечити ефективну взаємодію між користувачами та системою. Використання діаграм прецедентів є важливим етапом у процесі проектування будь-якої системи, оскільки це дозволяє детальніше уточнити функціональні вимоги та надати більш конкретний та точний опис функціональності системи.

В табл. 2.3 наведені функціональні вимоги до програмного продукту. Вони визначають конкретні дії, які програмний продукт повинен виконувати або конкретні дії, які він повинен дозволяти користувачу виконувати. Це можуть включати, наприклад, задачі, що пов'язані з обробкою даних, користувацьким інтерфейсом, інтеграцією з іншими системами, тощо. Вони є важливою складовою специфікації вимог до програмного продукту і служать основою для подальшого проектування, розробки та тестування.

Таблиця 2.3 – Функціональні вимоги до застосунку

Вимоги	Опис
REQ-1	Можливість реєстрації нових користувачів та авторизації вже зареєстрованих користувачів у системі
REQ-2	Система повинна здійснювати запис та зберігання журналу подій, що відбулися в системі, з метою моніторингу та аудиту активності користувачів
REQ-3	Додавати та редагувати інформацію облікових записів користувачів
REQ-4	Додати та редагувати інформацію про навчальні групи
REQ-5	Додати та редагувати інформацію про спеціальності
REQ-6	Додати та редагувати інформацію про предмети навчального закладу
REQ-7	Додати та редагувати інформацію про аудиторії
REQ-8	Можливість формування розкладів занять навчального закладу в залежності від наданої інформації

У табл. 2.4 представлені нефункціональні вимоги до програмного продукту. Вони визначають критерії, за якими оцінюється робота системи, а не конкретні характеристики її поведінки. Це може включати вимоги до продуктивності, надійності, зручності використання, безпеки, сумісності, масштабованості та інших аспектів, які стосуються якості системи та її впровадження.

Таблиця 2.4 – Нефункціональні вимоги до додатку

Вимоги	Опис
REQ-8	Додаток повинен працювати швидко та ефективно в умовах потенційно великого навантаження користувачів, забезпечуючи швидкий відгук на запити та мінімальні часи відповіді
REQ-9	Додаток повинен бути стійким до помилок та неперервно працювати без несправностей. Він повинен коректно обробляти можливі виняткові ситуації та запобігати витоку даних або системних збоїв
REQ-10	Додаток повинен забезпечувати захист конфіденційності даних користувачів та забезпечувати механізми автентифікації та авторизації, щоб запобігти несанкціонованому доступу до системи
REQ-11	Додаток повинен мати зрозумілий, інтуїтивно зрозумілий та легко використовуваний користувацький інтерфейс, що сприяє зручній навігації та забезпечує зручне взаємодію користувача з системою

Ідентифікація акторів та визначення їх цілей - це важливий етап аналізу вимог, який впливає на всі аспекти розробки програмного продукту. Кожен актор має одну або декілька цілей, які він прагне досягти, використовуючи систему. Ці цілі допомагають утворити портрети користувачів, що, у свою

чергу, використовуються для проектування та розробки програмного продукту.

Цілі акторів стають основою для визначення основних сценаріїв використання (use-cases). Сценарії використання допомагають визначити функціональні можливості та описують, як система повинна відповідати на дії користувача, а також, як вона повинна взаємодіяти з іншими системами.

У таблиці 2.5 представлена ідентифікація основних учасників, або акторів, які взаємодіють з системою, а також визначення їх ключових цілей у контексті програмного продукту. Актором може бути будь-яка особа, група осіб або система, що взаємодіє з додатком.

Таблиця 2.5 – Актори та цілі додатку

Актори	Цілі
Адміністратор	Виконання операцій, пов'язаних з обробкою та керуванням обліковими записами у системі. Його завдання включає управління правами доступу задля безпеки системи та вирішення пов'язаних з аутентифікацією та авторизацією завдань.
Користувач	Взаємодія з системою та виконання різноманітних завдань, пов'язаних зі своїми потребами та вимогами
База даних	Збереженні, організації та забезпеченні доступу до інформації, яка використовується системою

У таблиці 2.6 наведено деталізований перелік сценаріїв використання системи різними акторами. Кожен варіант використання містить унікальне ім'я, яке однозначно ідентифікує його, опис, що відображає основну мету або результат використання, а також послідовність кроків, які актор повинен виконати для досягнення даної мети.

Таблиця 2.6 – Опис варіантів використання додатку

Варіант використання	Ім’я	Опис
UC1	Реєстрація в системі	Дозволяє користувачам пройти процедуру реєстрації
UC2	Автентифікація в системі	Дозволяє користувачам пройти процедуру автентифікації, що підтверджує їхню ідентичність
UC3	Вивід каталогу користувачів	Користувачі з правами системного адміністратора мають можливість переглядати каталог всіх користувачів системи
UC4	Додати користувача	Користувачі з правами системного адміністратора можуть додавати нових користувачів до системи
UC5	Редагувати користувача	Користувачі з правами системного адміністратора мають можливість редагувати інформацію про обраного користувача зі списку
UC6	Вивід каталогу викладачів	Дозволяє користувачеві вивести каталог всіх викладачів
UC7	Додати нового викладача	Дозволяє користувачеві додати інформацію про нового викладача
UC8	Редагувати викладача	Дозволяє користувачеві редагувати інформацію вибраного викладача
UC9	Вивід каталогу груп	Дозволяє користувачеві вивести каталог всіх груп
UC10	Додати нову групу	Дозволяє користувачеві додати інформацію про нову групу

Продовження таблиці 2.6

UC11	Редагувати групу	Дозволяє користувачеві редагувати інформацію вибраної групи
UC12	Вивід каталогу спеціальностей	Дозволяє користувачеві вивести каталог всіх спеціальностей
UC13	Додати нову спеціальність	Дозволяє користувачеві додати інформацію про нову спеціальність
UC14	Редагувати спеціальність	Дозволяє користувачеві редагувати інформацію вибраної спеціальності
UC15	Вивід каталогу предметів	Дозволяє користувачеві вивести каталог всіх предметів
UC16	Додати новий предмет	Дозволяє користувачеві додати інформацію про новий предмет
UC17	Редагувати предмет	Дозволяє користувачеві редагувати інформацію вибраного предмету
UC18	Вивід каталогу аудиторій	Дозволяє користувачеві вивести каталог всіх аудиторій
UC19	Додати нову аудиторію	Дозволяє користувачеві додати інформацію про нову аудиторію
UC20	Редагувати аудиторію	Дозволяє користувачеві редагувати інформацію вибраної аудиторії
UC21	Формування розкладу	Дозволяє користувачеві здійснити формування розкладу за допомогою генетичного алгоритму
UC22	Вивід системних подій	Дозволяє вивести всі події, які відбулися в системі

Користувач в даній системі має можливість взаємодіяти з каталогами викладачів, груп, спеціальностей, предметів та аудиторій, здійснюючи

операції додавання, редагування та перегляду. Також користувач має можливість формувати розклад.

Системний адміністратор, крім всіх функцій, доступних користувачу, має доступ до каталогу користувачів системи, з можливістю їх додавання та редагування. Окрім цього, системний адміністратор має можливість перегляду системних подій.

На підставі наданих даних були розроблені use-case (діаграми прецедентів) для системного адміністратора та користувача, які відображені на рисунках 2.1 та 2.2 відповідно.

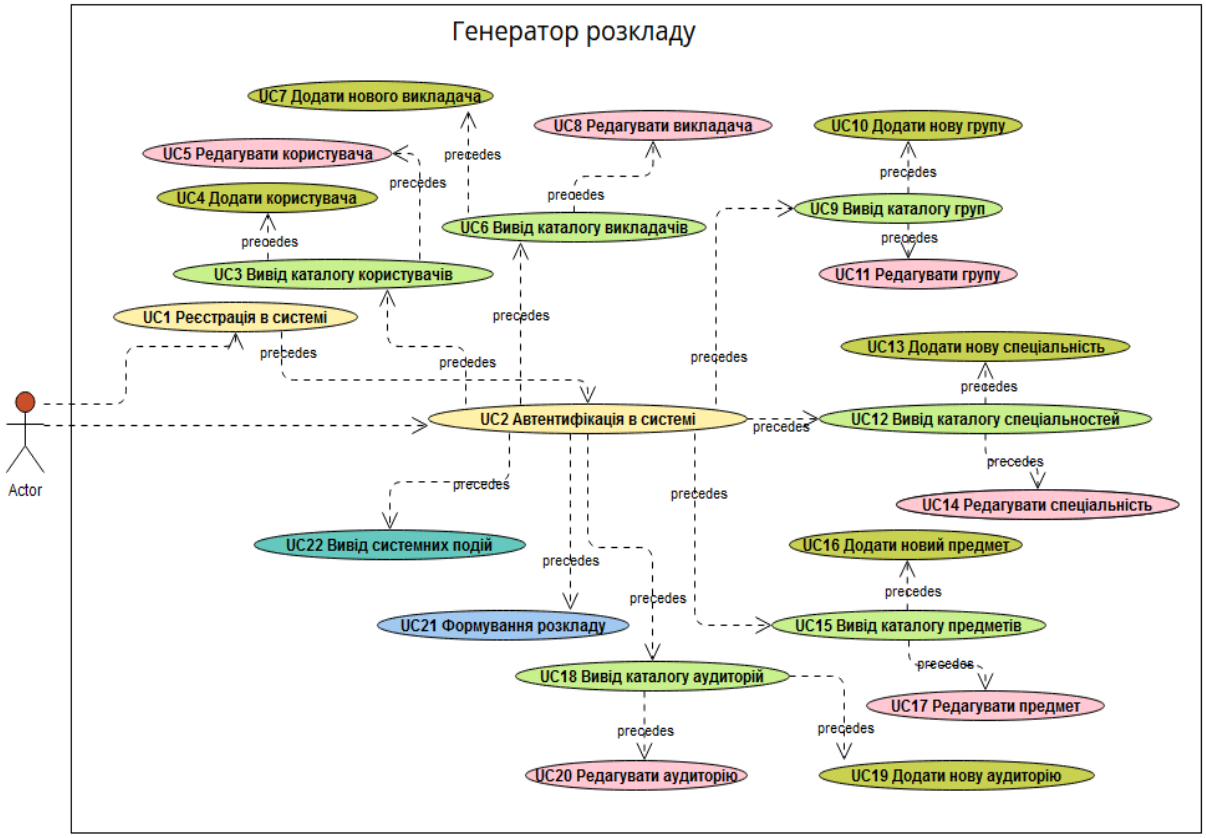


Рисунок 2.1 – Діаграма use-case для ролі «системний адміністратор»

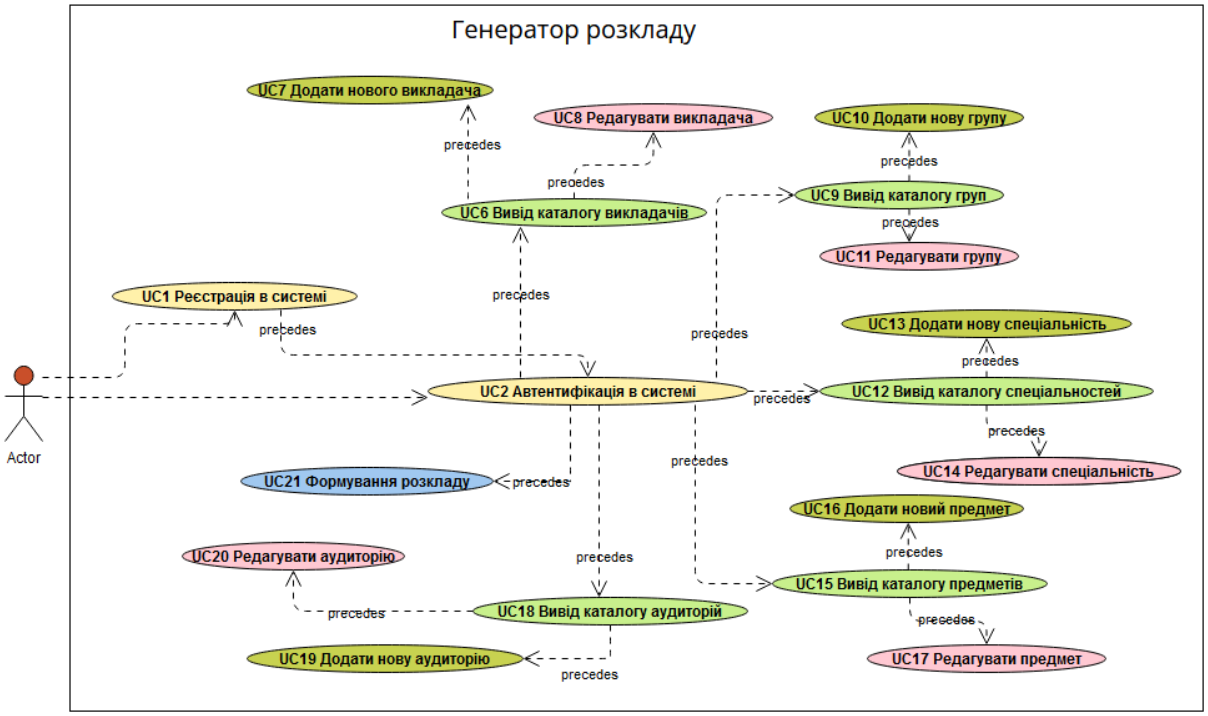


Рисунок 2.2 – Діаграма use-case для ролі «користувач»

2.3 Архітектура проекту

Для розробки даного проекту було використано трьохрівневу архітектуру, яка є поширеним типом архітектури програмного забезпечення. Цей підхід ґрунтується на розділенні системи на три рівні: клієнтський, логічний та рівень даних, кожен з яких має свої обов'язки, відповідальності та обмеження.

На клієнтському рівні розташовані компоненти, що відповідають за інтерфейс користувача та забезпечують взаємодію з користувачем. Це можуть бути різні типи інтерфейсів, такі як веб-сторінки, мобільні додатки або десктопні програми. На цьому рівні розташовані засоби, що дозволяють користувачам спілкуватися з системою та використовувати її функціональні можливості.

На логічному рівні розташовані компоненти, які відповідають за бізнес-логіку системи. Цей рівень включає обробку даних, логіку додатку та взаємодію з базою даних. Основне завдання цього рівня полягає в

забезпеченні виконання бізнес-правил та операцій, обробки даних та керування станом системи. На цьому рівні розташовані сервіси, що надають інтерфейси для взаємодії між клієнтським та рівнями даних, а також контролюють доступ до даних та забезпечують їх цілісність.

На рівні даних знаходиться база даних або інші джерела даних, які використовуються системою. Цей рівень відповідає за зберігання та керування даними, що використовуються на інших рівнях. Він забезпечує доступ до даних та виконання операцій збереження, оновлення та видалення даних.

2.3.1 Особливості розробки рівня BLL

Розробка бізнес-логіки сприяє розчленуванню функціональності додатку на логічні блоки, що сприяє досягненню більшої модульності та гнучкості у майбутньому. Цей підхід дозволяє незалежно змінювати окремі частини функціоналу без впливу на інші компоненти додатку, спрощуючи підтримку та подальший розвиток проекту.

Додатково, відокремлення бізнес-логіки від бази даних та користувацького інтерфейсу дозволяє використовувати різні системи управління базами даних та різні типи інтерфейсів без необхідності внесення змін до самої бізнес-логіки. Це забезпечує більшу гнучкість та адаптивність додатку до змін у вимогах бізнесу та технологічному середовищі [27].

Даний підхід до структури додатку відображає максимальну розділеність логічних компонентів, що гарантує їх взаємну незалежність та сприяє модульності та гнучкості проекту. Як результат, зміни, внесені до одного компонента, не вимагають необхідності внесення змін до інших компонентів, що дозволяє ефективно управляти та розвивати додаток, враховуючи змінні потреби бізнесу та технологій .

Діаграму даного шару показано на рис. 2.3.

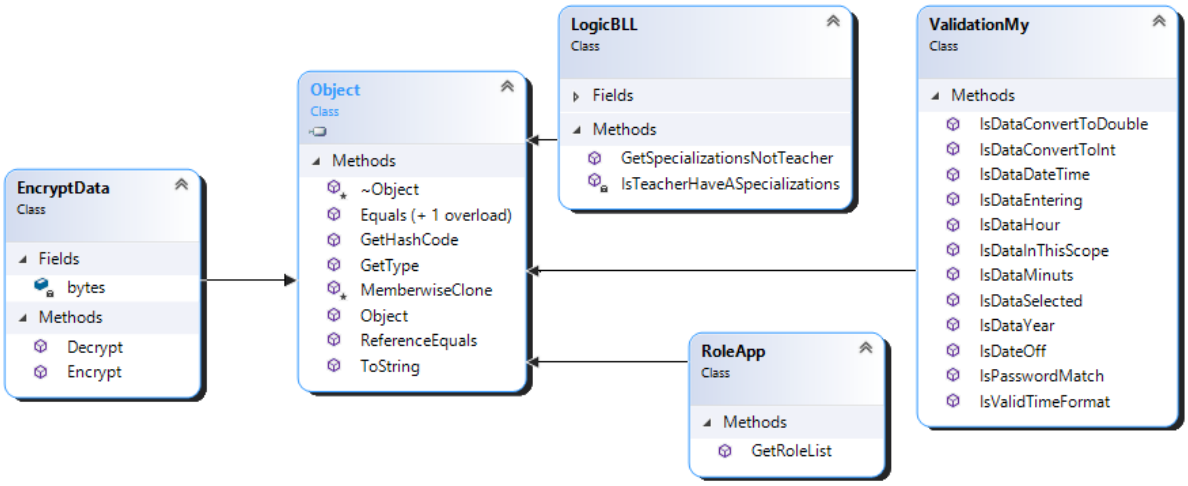


Рисунок 2.3 – Діаграма класів рівня бізнес логіки

Представлена діаграма включає наступні класи:

– клас LogicBLL відповідає за логіку генерації розкладу занять з використанням генетичних алгоритмів. Він містить методи, які реалізують генетичний алгоритм, включаючи створення початкової популяції, вибірку, схрещування, мутацію та оцінку фітнесу. Крім того, цей клас взаємодіє з класами рівня даних для отримання необхідної інформації про групи, предмети, викладачів, аудиторії та інше;

– клас EncryptData відповідає за шифрування та розшифрування даних, пов'язаних з обліковими записами користувачів. Він забезпечує механізми шифрування для захисту конфіденційності та безпеки даних, які пов'язані з користувачами системи;

– клас RoleApp відповідає за управління ролями та доступом користувачів в системі генерації розкладу занять. Він забезпечує можливість встановлення ролей для користувачів та контролює доступ до різних функціональних можливостей системи в залежності від призначених ролей;

– клас ValidationMy відповідає за валідацію даних, пов'язаних з автоматизованою генерацією розкладу занять за допомогою генетичних алгоритмів. Він забезпечує перевірку вхідних даних на відповідність вимогам системи та забезпечує цілісність та правильність введених даних для успішної генерації розкладу.

2.3.2 Особливості розробки DAL

Метою компоненти DAL (Data Access Layer) є забезпечення доступу до даних, що знаходяться у базі даних, з різних рівнів системи, включаючи сервісний рівень (BLL). У рамках даної системи, компонента DAL була реалізована з використанням технології ADO.NET. Для досягнення цієї мети був використаний об'єкт System.Data.OleDb, який дозволяє встановлювати з'єднання з базою даних за допомогою різних провайдерів даних (рис. 2.4). Компонента DAL виконує завдання роботи з даними, такі як отримання, зміна, видалення та додавання даних до бази даних [28]. Вона надає абстракцію бази даних від інших компонентів системи, що дозволяє досягти незалежності від конкретного постачальника бази даних та спрощує взаємодію з даними.

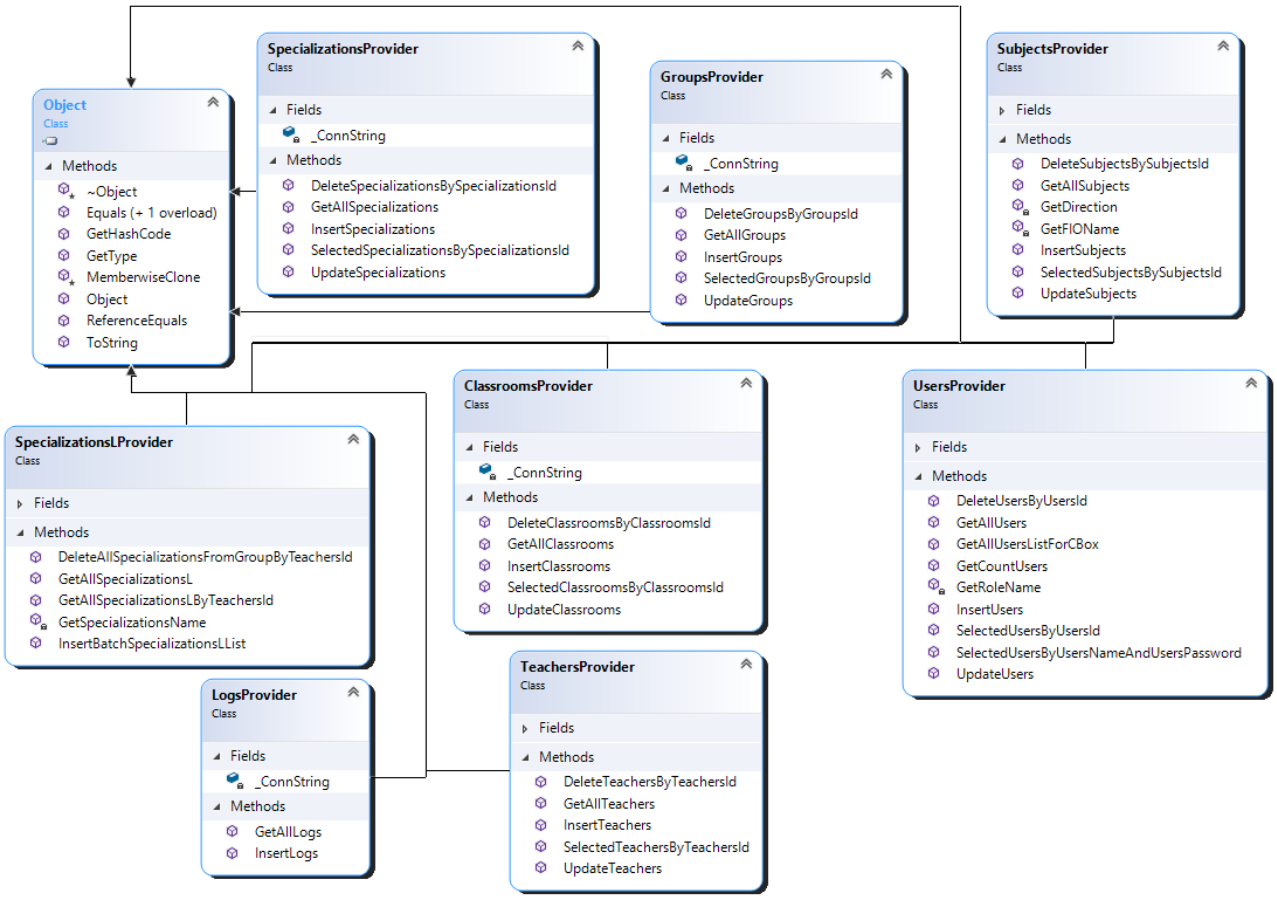


Рисунок 2.4 – Діаграма класів рівня даних

Даний рівень системи складається з восьми класів:

– клас ClassroomsProvider призначений для забезпечення доступу до інформації про аудиторії. Він містить методи для отримання списку аудиторій, додавання нової аудиторії, оновлення інформації про існуючу аудиторію та видалення аудиторії;

– клас GroupsProvider призначений для забезпечення доступу до інформації про групи. Він містить методи для отримання списку груп, додавання нової групи, оновлення інформації про існуючу групу та видалення групи;

– клас LogsProvider призначений для збереження та управління журналами або логами, пов'язаними з генерацією розкладу занять;

– клас SpecializationsLProvider призначений для забезпечення доступу до інформації про спеціальності, які викладаються викладачами навчальних закладів;

– клас SpecializationsProvider призначений для забезпечення доступу до інформації про спеціальності. Він містить методи для отримання списку спеціалізацій, додавання нової спеціалізації, оновлення інформації про існуючу спеціалізацію та видалення спеціалізації;

– клас SubjectsProvider призначений для забезпечення доступу до інформації про предмети. Він забезпечує методи для отримання списку предметів та виконання інших операцій, пов'язаних з предметами;

– клас TeachersProvider призначений для забезпечення доступу до інформації про викладачів. Він надає методи для отримання списку викладачів та управління інформацією про них;

– клас UsersProvider призначений для забезпечення доступу до інформації про користувачів системи.

2.3.3 Особливості розробки рівня UI

Розробка рівня користувацького інтерфейсу (UI) є ключовим етапом для досягнення ефективної автоматизації процесів підтримки навчального

процесу. Успішна реалізація цього рівня має велике значення, оскільки він забезпечує взаємодію між користувачами та системою, і його інтерфейс повинен бути добре продуманим та зрозумілим для користувачів.

На рівні користувацького інтерфейсу були розроблені форми, що містять набір елементів керування, таких як кнопки, текстові поля та таблиці. Ці елементи взаємодіють з іншими рівнями системи через відповідні обробники подій. Користувач має можливість взаємодіяти з системою через цей рівень, вводити необхідну інформацію та отримувати результати у зручному форматі [29].

Для представлення класів системи рівня користувацького інтерфейсу була використана діаграма класів, яка наведена на рис. 2.5. Ця діаграма відображає взаємозв'язки та структуру класів на цьому рівні системи.

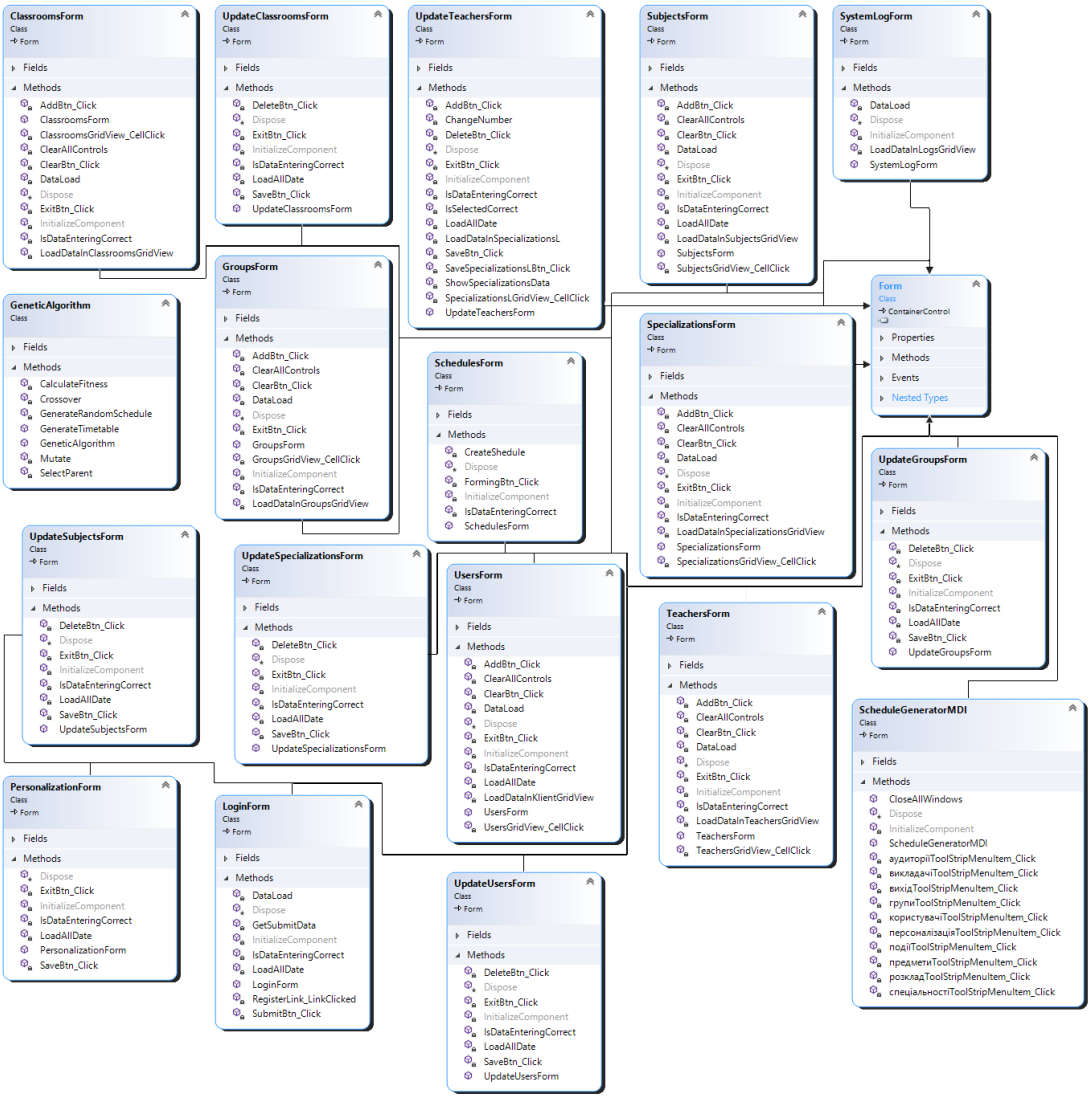


Рисунок 2.5 – Діаграма класів рівня користувацького інтерфейсу

У складі діаграми рівня користувацького інтерфейсу присутні такі основні класи:

– клас SchedulesForm відповідає за відображення та взаємодію з формою для генерації та відображення розкладу занять. Він забезпечує представлення цієї функціональності та взаємодію з користувачем через відповідну форму;

– клас ClassroomsForm відповідає за представлення та взаємодію з формою для управління аудиторіями. Його методи дозволяють відображати список аудиторій та додавати нові аудиторії;

– клас GroupsForm відповідає за представлення та взаємодію з формою для управління групами. Він надає функціональність для відображення списку груп та виконання операцій, таких як додавання нової групи, оновлення та видалення існуючих груп;

– клас SpecializationsForm відповідає за представлення та взаємодію з формою для управління спеціальностями вищого навчального закладу. Він надає функціональність для відображення списку спеціальностей, додавання нових спеціальностей, а також оновлення та видалення існуючих спеціальностей.

– клас SubjectsForm відповідає за представлення та взаємодію з формою для управління дисциплінами навчання. Він дозволяє відображати список дисциплін, а також додавати нові дисципліни та виконувати інші операції пов'язані з керуванням цими об'єктами.

– клас TeachersForm відповідає за представлення та взаємодію з формою для управління викладачами. Він надає можливість відображати список викладачів та здійснювати операції, такі як додавання нового викладача, оновлення та видалення існуючих викладачів;

– клас LoginForm виконує функцію представлення та взаємодії з формою входу до системи генерації розкладу занять;

– клас SystemLogForm відповідає за представлення та взаємодію з формою системного журналу;

– клас UpdateUsersForm виконує завдання представлення та взаємодії з формою для оновлення існуючих даних про користувачів;

– клас UsersForm відповідає за представлення та взаємодію з формою для управління користувачами системи;

– клас ScheduleGeneratorMDI є відповідальним за головне вікно додатку автоматизованої генерації розкладу занять.

2.4 Висновок

В даному розділі виконано комплексний аналіз інструментарію, потрібного для реалізації поставленої задачі.

Було проведено порівняльний аналіз потенційних мов програмування, включаючи C#, Java та C++. В результаті цього аналізу, вибрано мову програмування C# для подальшої розробки програмного продукту через її високу продуктивність, сучасність та потужні засоби створення користувацького інтерфейсу.

У процесі аналізу середовищ розробки було враховано такі варіанти, як Visual Studio, Eclipse та PyCharm. На основі цих оцінок, обрано Visual Studio, що надає потужні інструменти для розробки на C# і має широкі можливості для тестування та налагодження програмного забезпечення.

В рамках розділу також було проведено аналіз вимог, сформовано use-case діаграми та визначені основні прецеденти, що допомогли в деталізації процесу розробки та плануванні архітектури проекту.

Архітектура програмного забезпечення була розроблена на основі трьохрівневого підходу, який включає рівні бізнес-логіки (BLL), користувацького інтерфейсу (UI) та доступу до даних (DAL). Така структура забезпечує гнучкість, масштабованість та чіткість проекту, що є критично важливим для успішної реалізації складного проекту, як автоматизована система генерації розкладу.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Проєктування бази даних

Під час проектування бази даних для предметної області «Генератор розкладу» був використаний метод «сутність-зв'язок» (ER метод). Цей метод передбачає розв'язання завдань на нижчих рівнях перед тим, як переходити до завдань більш високого рівня. Згідно з цим підходом була розроблена логічна модель за допомогою ER-діаграми, в якій були визначені такі сутності та їх атрибути [30]:

– спеціалізація: ідентифікатор спеціалізації, назва та опис;

– викладач: ідентифікатор викладача, прізвище, номер телефону, адреса проживання, електронний адрес та навантаження (у годинах);

– список спеціалізацій: ідентифікатор списку спеціалізацій, ідентифікатор викладача, ідентифікатор спеціалізації;

– групи: ідентифікатор групи, напрямок навчання та кількість студентів;

– події: ідентифікатор події, дата події, ідентифікатор користувача та опис події;

– користувачі: ідентифікатор користувача, прізвище, ім’я, назва облікового запису, пароль, ідентифікатор ролі, електронна адреса та додатковий опис;

– предмет: ідентифікатор предмету, назва, ідентифікатор викладача та ідентифікатор групи;

– аудиторії: ідентифікатор аудиторії, назва та максимальна кількість місць.

Під час проектування бази даних для додатку "Генератор розкладу" використовувався метод "сутність-зв'язок" (ER-метод). Цей метод передбачає розгляд завдань на низькому рівні перед тим, як переходити до вирішення завдань більш високого рівня. За допомогою ER-діаграми було розроблено

логічну модель, в якій були визначені сутності та атрибути для предметної області "Генератор розкладу".

Під час встановлення зв'язків між сутностями, використовувалися зв'язки між полями однакових типів або полями, що мають взаємозв'язок. Зв'язки між таблицями були встановлені за допомогою зовнішніх ключів однієї таблиці та внутрішніх ключів іншої таблиці. У логічній моделі бази даних використовувалися зв'язки типу "один до одного", "один до багатьох" та "багато до багатьох". Це дозволило коректно побудувати зв'язки між таблицями, що забезпечує належне функціонування бази даних для потреб додатку "Генератор розкладу".

Організовано такі зв'язки між таблицями:

– «Спеціалізація» (поле «SpecializationId») – «Список спеціалізацій» (поле «SpecializationId»), вид зв’язку 1:N;

– «Викладач» (поле «TeacherId») – «Список спеціалізацій» (поле «TeacherId»), вид зв’язку 1:N;

– «Викладач» (поле «TeacherId») – «Предмет» (поле «TeacherId»), вид зв’язку 1:N;

– «Групи» (поле «GroupId») – «Предмет» (поле «GroupId»), вид зв’язку 1:N;

– «Користувач» (поле «UserId») – «Події» (поле «UserId»), вид зв’язку 1:N.

У процесі проектування бази даних для додатку "Генератор розкладу" було використано ER-діаграму, яка є ключовим етапом у розробці. ER-діаграма дозволяє відображати зв'язки між сутностями та їх атрибутами, створюючи візуальне представлення структури бази даних (рис. 3.1).

ER-діаграма допомагає розуміти структуру бази даних та визначити взаємозв'язки між сутностями, що впливає на подальшу роботу з даними та розробку функціональності системи.

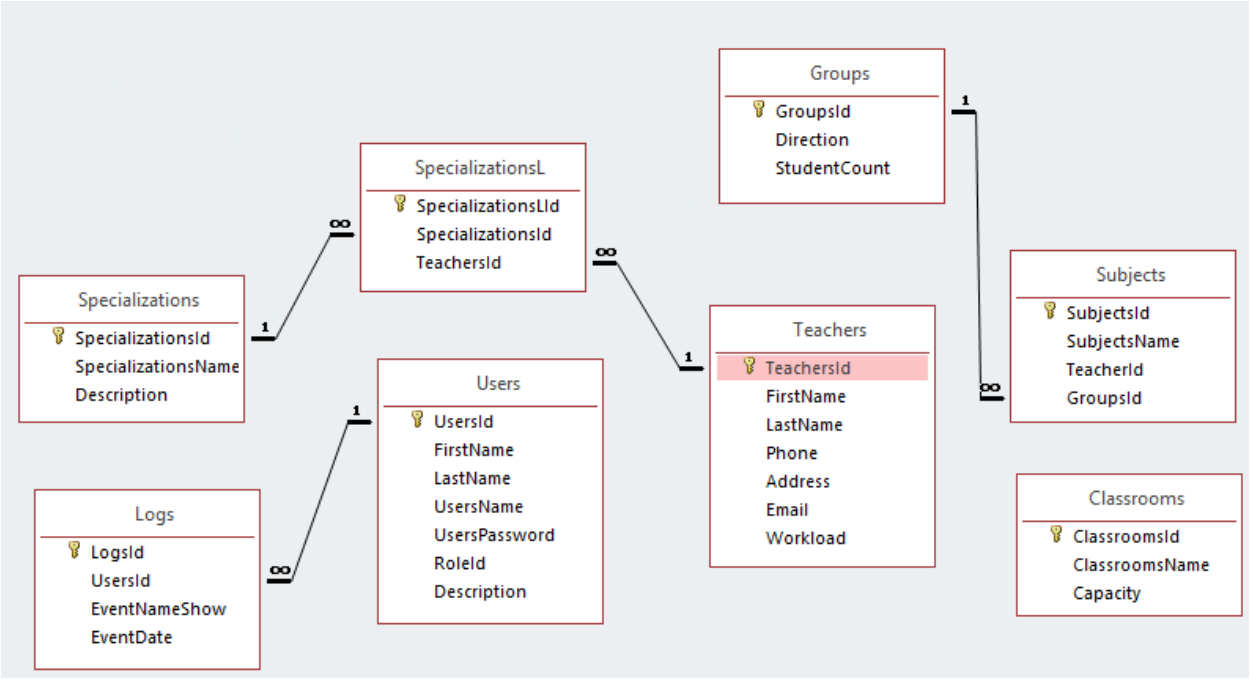


Рисунок 3.1 – Діаграма бази даних

3.2 Розробка алгоритму додатку

Алгоритм генерації розкладу складається з наступних етапів:

1. Ініціалізація. На початку створюється початкова популяція розкладів. Кожен розклад генерується випадковим чином, враховуючи вхідні дані про групи, вчителів, предмети, класи і кількість уроків на тиждень.

2. Оцінка адаптивності. Для кожного розкладу в популяції обчислюється його адаптивність. Це визначається шляхом порівняння розкладу з заданими обмеженнями. Розклади, які краще відповідають обмеженням, отримують вищу оцінку адаптивності.

3. Селекція. На основі оцінки адаптивності вибираються розклади, які будуть перейти до наступного покоління. Це здійснюється через елітну селекцію, де найкращі розклади залишаються, а також за допомогою турнірної селекції, де обираються розклади на основі їхньої адаптивності.

4. Схрещування. Вибрані розклади схрещуються, що призводить до створення нових розкладів. Цей процес здійснюється за допомогою

одноточкового схрещування, де вибирається одна точка обміну генами між розкладами.

5. Мутація. Деякі гени в розкладі змінюються випадковим чином для внесення різноманітності в популяцію. Це допомагає уникнути застрягання в локальних оптимумах і досліджувати нові рішення.

6. Оновлення популяції. Нова популяція, яка складається з найкращих розкладів попередньої популяції та нових розкладів, отриманих в результаті схрещування і мутації, замінює стару популяцію.

7. Перевірка умови зупинки. Алгоритм перевіряє, чи було досягнуто максимальної кількості поколінь або чи адаптивність найкращого розкладу досягла "ідеального" значення. Якщо ці умови виконуються, алгоритм зупиняється. В іншому випадку, алгоритм повертається до кроку 2 і продовжує цикл для наступного покоління.

8. Результат. Після завершення алгоритму, найкращий розклад з останньої популяції вважається кінцевим результатом. Цей розклад представляє найкраще рішення, яке алгоритм зміг знайти для задачі генерації розкладу з урахуванням обмежень.

3.3 Розробка основних модулів

Після успішного створення бази даних, наступним кроком є інтеграція бази даних з системою за допомогою інтегрованої середи розробки Visual Studio 2019. Це надає розробникам зручний спосіб взаємодії з базою даних шляхом використання графічного інтерфейсу та створення зв'язків між таблицями бази даних та формами додатку.

Для успішного підключення бази даних до проекту використовується файл «App.config». У цьому файлі необхідно створити змінну під назвою «CONNECT», яка буде містити параметри підключення до бази даних (див. рис. 3.2). Ці параметри можуть включати ім'я сервера, назву бази даних,

ідентифікаційні дані користувача та інші налаштування, які визначають спосіб доступу до бази даних.

Після визначення змінної «CONNECT» у файлі «App.config», розробники можуть зручно керувати параметрами підключення до бази даних без необхідності внесення змін у вихідний код програми. Це забезпечує більшу гнучкість та легкість управління базою даних під час розробки та експлуатації додатку.

```
<appSettings>
  <!-- Підключення до бази даних -->
  <add key="CONNECT" value="Provider=Microsoft.Jet.OLEDB.4.0;
       Data Source=|DataDirectory|\DataBase.mdb;"/>
</appSettings>
```

Рисунок 3.2 – Параметри налаштування з’єднання з базою даних

Після успішного налаштування змінної "CONNECT" в файлі "App.config", система отримує доступ до бази даних, що дозволяє підключати таблиці бази до форм додатку. Це надає користувачам можливість взаємодіяти з даними, додавати нові записи, редагувати існуючі та видаляти непотрібні дані. Такий підхід спрощує роботу з базою даних та забезпечує зручне управління збереженою інформацією.

Для забезпечення ефективної роботи з базою даних використовується простір імен System.Data.OleDb. Цей простір містить набір класів, які дозволяють взаємодіяти з базою даних, виконувати запити та зберігати зміни.

Один із ключових класів в просторі імен System.Data.OleDb - це OleDbConnection. Він забезпечує зв'язок з базою даних і надає можливість виконувати різноманітні операції з даними, такі як вставка, оновлення та видалення записів. Цей клас пропонує широкі можливості для роботи з даними у базі даних та виконання різних запитів.

У процесі розробки проекту можна використовувати елемент menuStrip для створення меню у головному вікні додатку (рис. 3.3). Цей елемент додається до інтерфейсу програми і дозволяє створити зручну навігацію та

взаємодію з користувачами шляхом вибору опцій та виклику функцій програми.

Використання меню у проекті допомагає створити зручний та інтуїтивно зрозумілий інтерфейс, що полегшує користування програмою та сприяє покращенню загального досвіду користувача.

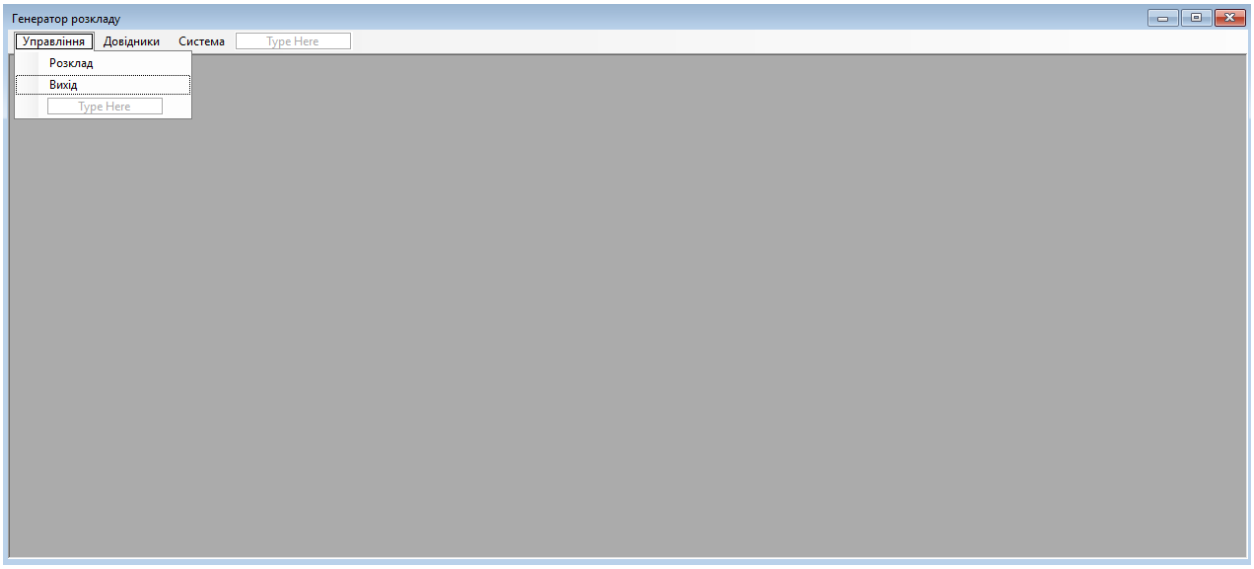


Рисунок 3.3 – Додавання головного меню

Для кожного пункту меню було розроблено відповідний код, що відповідає за створення нового екземпляру форми при виборі пункту. При відкритті певного пункту меню, відповідна форма створюється, і її вікно стає активним, що дозволяє користувачу взаємодіяти з нею.

Для забезпечення коректного відображення вікон та уникнення можливих конфліктів, був доданий код, що забезпечує закриття попередньо відкритого вікна перед відкриттям нового. Застосування цього коду дозволяє відкривати лише одне вікно одночасно, замінюючи попереднє вікно новим при виборі пункту меню.

Подібний код застосовується для всіх пунктів меню, забезпечуючи їх правильну роботу та взаємозв'язок з відповідними формами. Це дозволяє зручно переходити між різними функціональними модулями програми та забезпечує коректне відображення та взаємодію з вікнами.

На рис. 3.4 наведено приклад коду, що ілюструє цю логіку для одного з пунктів меню. Цей код відповідає за створення відповідної форми при виборі пункту меню та заміну попередньо відкритого вікна новим. Такий підхід забезпечує правильну роботу програми та зручну взаємодію з користувачем.

```
private void розкладToolStripMenuItem_Click(object sender, EventArgs e) {
  CloseAllWindows();
  SchedulesForm schedulesForm = new SchedulesForm();
  schedulesForm.MdiParent = this;
  schedulesForm.WindowState = FormWindowState.Maximized;
  schedulesForm.Show();
}
```

Рисунок 3.4 – Код пунктів меню

Після цього розроблено спеціальні класи для ефективної маніпуляції базою даних. Наступний опис містить фрагменти реалізації важливих методів, які використовуються в цих класах. Наприклад, для збереження інформації про викладача у базі даних був створений метод з назвою «InsertTeachers», і його вихідний код представлено на рис. 3.5.

```
1 reference
public void InsertTeachers(string LastName, string FirstName, string Phone,
  string Address, string Email, int Workload) {
  string SqlString = "INSERT INTO Teachers (LastName, FirstName, Phone, Address, " +
    "Email, Workload) Values(?, ?, ?, ?, ?, ?)";

  using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
    using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
      cmd.CommandType = CommandType.Text;
      cmd.Parameters.AddWithValue("LastName", LastName);
      cmd.Parameters.AddWithValue("FirstName", FirstName);
      cmd.Parameters.AddWithValue("Phone", Phone);
      cmd.Parameters.AddWithValue("Address", Address);
      cmd.Parameters.AddWithValue("Email", Email);
      cmd.Parameters.AddWithValue("Workload", Workload);
      conn.Open();
      cmd.ExecuteNonQuery();
      conn.Close();
    }
  }
}
```

Рисунок 3.5– Код методу «InsertTeachers»

Метод "InsertTeachers" виконує вставку даних про викладача до бази даних. Запит формується для таблиці "Teachers" і включає поля "LastName", "FirstName", "Phone", "Address", "Email" та "Workload". Значення цих полів

передаються як параметри. Після виконання запиту, з'єднання з базою даних закривається.

Для видалення даних про викладача, також був розроблений метод, код якого представлено на рис. 3.6.

```
public void DeleteTeachersByTeachersId(int TeachersId) {
  string SqlString = "DELETE FROM Teachers WHERE TeachersId=" + TeachersId.ToString();
  using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
    using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
      conn.Open();
      cmd.ExecuteNonQuery();
      conn.Close();
    }
  }
}
```

Рисунок 3.6 – Код методу «DeleteTeachersByTeachersId»

Метод "DeleteTeachersByTeachersId" використовується для видалення даних про викладача з бази даних за допомогою їх унікального ідентифікатора. Цей метод є частиною розробки програмного продукту, який спрямований на автоматизацію процесу складання навчального розкладу.

У методі виконується формування SQL-запиту для видалення запису з таблиці "Teachers". Запит містить умову WHERE, де поле "TeachersId" порівнюється з переданим значенням параметра "TeachersId", що передається методу. Це дозволяє вибрати конкретного викладача для видалення.

Далі, з використанням об'єкта OleDbConnection, встановлюється з'єднання з базою даних за допомогою рядка з'єднання, який зберігається у змінній "_ConnString".

Об'єкт OleDbCommand використовується для виконання SQL-запиту, переданого як перший аргумент конструктору. SQL-запит, що видаляє дані з таблиці, передається в якості рядка запиту.

Після виконання підключення до бази даних за допомогою методу Open() об'єкту OleDbConnection, викликається метод ExecuteNonQuery() об'єкту OleDbCommand. Цей метод виконує SQL-запит і не повертає результат.

Наприкінці, метод Close() викликається для закриття підключення до бази даних.

Після завершення розробки всіх класів рівня даних, було розроблено відповідні форми, які дозволяють користувачам взаємодіяти з програмою та виконувати різні операції. Одним з таких важливих елементів інтерфейсу є форма для розкладу занять на тиждень.

Дана форма використовує функціональність, що була реалізована у класах рівня даних, з метою отримання та збереження даних про розклад занять. Вона взаємодіє з методами, що реалізовані в інших класах, для забезпечення потрібного функціоналу. Детальні описи цих методів наведені на рис. 3.7.

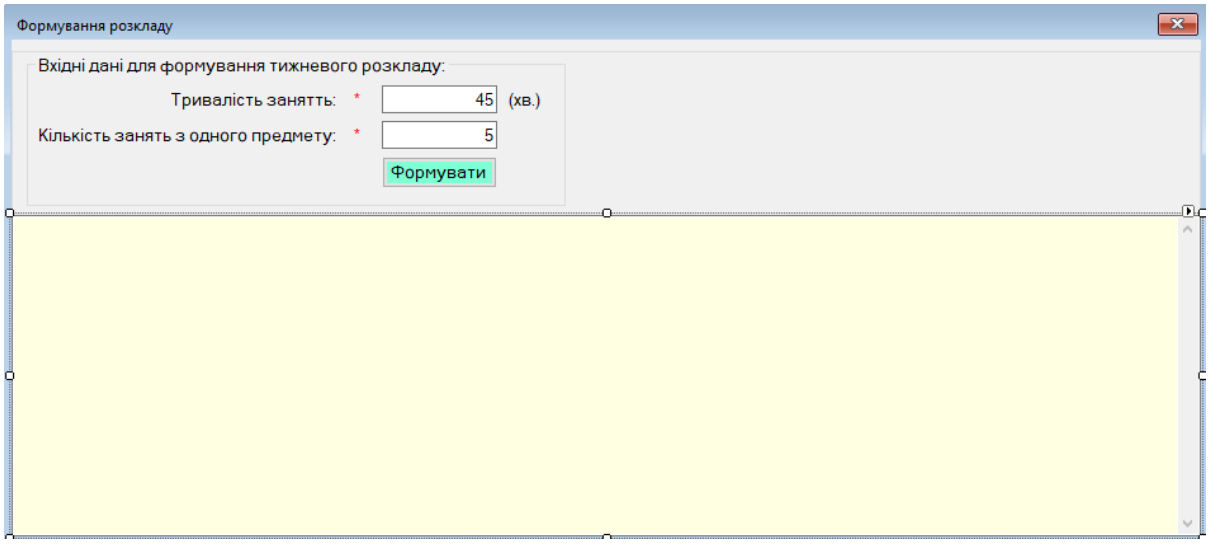


Рисунок 3.7 – Розробляння форми формування розкладу занять

Під час ініціалізації форми, у її конструкторі, виконується завантаження списків даних про групи, викладачів, предмети та аудиторії (рис. 3.8). Цей процес забезпечує наявність актуальних даних для подальшої взаємодії з формою.

```
public SchedulesForm() {
  InitializeComponent();
  _GroupsList = _GroupsProvider.GetAllGroups();
  _TeachersList = _TeachersProvider.GetAllTeachers();
  _SubjectsList = _SubjectsProvider.GetAllSubjects();
  _ClassroomsList = _ClassroomsProvider.GetAllClassrooms();
}
```

Рисунок 3.8– Конструктор класу «SchedulesForm»

У всіх створених формах забезпечується перевірка коректності введених даних. Кожна форма має власний метод під назвою "IsDataEnteringCorrect", який виконує перевірку даних у всіх обов'язкових полях. На рисунку 3.9 наведено приклад такого методу для форми "TeachersForm".

Метод "IsDataEnteringCorrect" використовується для перевірки введених даних у формі "TeachersForm". Він здійснює різноманітні перевірки відповідно до вимог та обмежень, пов'язаних з полями у даній формі. Наприклад, можуть проводитись перевірки на наявність обов'язкових полів, правильний формат електронної пошти або номеру телефону, допустимий діапазон числових значень та інші.

Цей метод гарантує, що перед передачею даних з форми для подальшої обробки вони будуть перевірені на коректність, сприяючи уникненню можливих помилок або некоректної обробки даних. Користувач буде проінформований про будь-які виявлені помилки і матиме можливість виправити їх перед продовженням роботи. Метод для перевірки даних на коректність у формі "TeachersForm" представлений на рис. 3.9.

```
1 reference
private bool IsDataEnteringCorrect() {
  bool isCorrect = true;
  if (_validation.IsDataEntering(LastNameTBox.Text)) {
    LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_validation.IsDataEntering(FirstNameTBox.Text)) {
    FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_validation.IsDataEntering(PhoneTBox.Text)) {
    PhoneValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    PhoneValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_validation.IsDataConvertToInt(WorkloadTBox.Text)) {
    WorkloadValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    WorkloadValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  return isCorrect;
}
```

Рисунок 3.9– Код методу для перевірки коректності введених даних

Наступним етапом розробки було створення події під назвою "FormingBtn_Click", яка спрацьовує після натискання кнопки "Формувати". Ця подія виконується згідно з кодом, наведеним на рис. 3.10.

```
private void FormingBtn_Click(object sender, EventArgs e) {
  if (IsDataEnteringCorrect()) {
    CreateShedule();
  }
}
```

Рисунок 3.10– Код події «FormingBtn_Click»

Вказана подія викликає метод з назвою "CreateShedule", який відповідає за створення розкладу занять за допомогою генетичного алгоритму. Цей метод використовує надані вхідні дані про групи, викладачів, предмети та аудиторії, а також деякі параметри, щоб згенерувати оптимальний розклад занять. Детальний опис цього методу наведений на рис. 3.11.

```
1 reference
private void CreateShedule() {
  // Set the lesson duration and number of lessons per week
  int lessonDuration = Convert.ToInt32(LessonDurationTBox.Text); // minutes
  int lessonsPerWeek =  Convert.ToInt32(LessonsPerWeekTBox.Text); //відображає кількість занять, які повинні відбуватись протягом одного тижня
                                                                  //для кожної групи. Ця змінна використовується для задання обмежень щодо розкладу занять
                                                                  //та для створення випадкових розкладів під час генерації початкової популяції в генетичному алгоритмі.

  // Create the Genetic Algorithm instance
  GeneticAlgorithm ga = new GeneticAlgorithm(_GroupsList, _TeachersList, _SubjectsList, _ClassroomsList,
    lessonDuration, lessonsPerWeek);
  // Generate the optimal timetable
  Schedule optimalSchedule = ga.GenerateTimetable();
  string raport = "";
  // Output the optimal schedule
  Console.WriteLine("Оптимальний розклад:");
  RaportTBox.Text += String.Format("{0,-10}|{1, -25}|{2, -30}|{3, -15}|{4, -6}|{5, -4}|{6, -6}|\r\n",
    "Група", "Предмет", "Викладач", "Аудиторія", "Вміст.", "День", "Пара");
  foreach (var lesson in optimalSchedule.Lessons) {
    Groups group = _GroupsList.FirstOrDefault(g => g.GroupsId == lesson.GroupId);
    Subjects subject = _SubjectsList.FirstOrDefault(s => s.SubjectsId == lesson.SubjectId);
    Teachers teacher = _TeachersList.FirstOrDefault(t => t.TeachersId == lesson.TeacherId);
    Classrooms classroom = _ClassroomsList.FirstOrDefault(c => c.ClassroomsId == lesson.ClassroomId);

    raport += String.Format("{0,-10}|{1, -25}|{2, -30}|{3, 15}|{4, 6}|{5, 4}|{6, 6}|\r\n", group.Direction, subject.SubjectsName,
      teacher.FIO, classroom.ClassroomsName, classroom.Capacity, lesson.DayOfWeek, lesson.Period);
  }
  RaportTBox.Text += raport;
}
```

Рисунок 3.11– Код методу «CreateShedule»

Метод "CreateShedule" виконує процес створення розкладу занять. Він працює наступним чином:

1. Спочатку встановлюються значення для тривалості одного заняття та кількості занять на тиждень, які отримуються з відповідних текстових полів (LessonDurationTBox та LessonsPerWeekTBox).

2. Створюється екземпляр генетичного алгоритму (GeneticAlgorithm) з використанням списків груп, викладачів, предметів та аудиторій (_GroupsList, _TeachersList, _SubjectsList, _ClassroomsList), а також вказаними значеннями тривалості заняття та кількості занять на тиждень.

3. Запускається генерація оптимального розкладу за допомогою методу GenerateTimetable() генетичного алгоритму. Оптимальний розклад зберігається у змінній optimalSchedule.

4. Виконується виведення оптимального розкладу. Кожне заняття виводиться у вигляді рядка, який містить інформацію про групу, предмет, викладача, аудиторію, обсяг матеріалу, день тижня та період (пару). Ці рядки додаються до змінної raport.

5. Наприкінці, згенерований розклад змінної raport додається до текстового поля RaportTBox, яке відображає оптимальний розклад.

Приватний метод "FormingBtn_Click" викликається при натисканні кнопки "Формувати". Він перевіряє правильність введених даних за допомогою методу "IsDataEnteringCorrect" і, якщо дані введені коректно, викликає метод "CreateShedule" для створення розкладу занять.

3.4 Функціональне та модульне тестування

Тестування відіграє надзвичайно важливу роль у розробці програмного забезпечення "Генератор розкладу" та сприяє забезпеченню його якості та надійності. Цей процес передбачає детальне перевіряння різних аспектів системи, з метою впевненості в її відповідності вимогам та очікуванням користувачів.

Під час тестування виконуються різноманітні тести, що охоплюють різні аспекти функціональності системи. Наприклад, тести продуктивності дозволяють перевірити швидкість та ефективність обробки великих обсягів даних системою. Тести безпеки виявляють потенційні проблеми

забезпечення безпеки та сприяють захисту від можливих загроз. Тести надійності перевіряють стабільність системи за різних умов та навантажень.

Помилки та недоліки, що були виявлені під час тестування фіксуються та виправляються, що сприяє покращенню якості та функціональності системи. Крім того, тестування забезпечує впевненість у роботі системи згідно очікувань та потреб користувачів.

Тест-кейс №1. Вхід до системи. Перевірка на наявність правильної авторизації користувача:

– відкрити форму авторизації;

– ввести існуючий логін та пароль користувача;

– перевірити, що користувач має доступ до головної форми застосунку.

В результаті, користувач має мати доступ до головної форми застосунку.

Тест-кейс №2. Додавання нового запису про викладача. Перевірка на правильність заповнення полів:

– відкрити форму додавання нового запису про викладача;

– заповнити всі обов'язкові поля (ім'я, прізвище, тощо);

– натиснути кнопку «Додати»;

– перевірити, що новий запис про викладача збережений у базі даних.

В результаті, запис про викладача має бути доданий в базу даних та відображається на сторінці списку викладачів.

Тест-кейс №3. Редагування запису про викладача. Перевірка на правильність редагування полів:

– вибрати необхідний запис про викладача;

– змінити значення одного з полів (наприклад, прізвище);

– натиснути кнопку «Зберегти»;

– перевірити, що відредагований запис про викладача збережений у базі даних.

В результаті, запис про викладача має бути відредагований в базі даних та відображається на сторінці списку викладачів.

Тест-кейс №4. Формування розкладу занять ВНЗ. Перевірка на правильність формування розкладу:

– відкрити форму для формування розкладу;

– ввести параметри: тривалості заняття та кількості занять з одного предмету;

– натиснути кнопку «Формувати»;

– перевірити, що у відповідному вікні відобразились дані та перевірити їхню адекватність.

В результаті формується правильний список розкладу згідно поставленій задачі.

Тест-кейс №5. Видалення запису про викладача. Перевірка на правильність видалення:

– відкрити форму видалення запису про викладача;

– вибрати викладача, якого потрібно видалити;

– натиснути кнопку «Видалити»;

– перевірити, що обраний запис про викладача був успішно видалений з бази даних.

Після завершення функціонального тестування системи, проводиться модульне тестування, що є важливим етапом розробки програмного забезпечення. Модульне тестування дозволяє перевірити роботу окремих модулів програми згідно вимог та специфікації.

Для проведення модульного тестування використовується окремий проект типу "Unit Test Project" у середовищі Visual Studio. У межах цього проекту визначаються тести для функцій програми, які підлягають перевірці. Кожен тест охоплює окремий функціональний або методичний аспект програми. Після визначення тестів запускається модульне тестування, яке проводиться для кожного модуля окремо. Це дозволяє перевірити роботу та результати кожного модуля відповідно до очікуваних вимог та специфікації.

Результати модульного тестування подаються детально, зокрема, на рис. 3.12, що дає змогу перевірити успішне виконання тестів.

Модульне тестування допомагає виявити проблеми та помилки на ранніх етапах розробки, що сприяє покращенню якості та надійності програмного забезпечення. Воно дозволяє впевнитись у правильному функціонуванні окремих модулів програми та їх відповідності очікуванням.

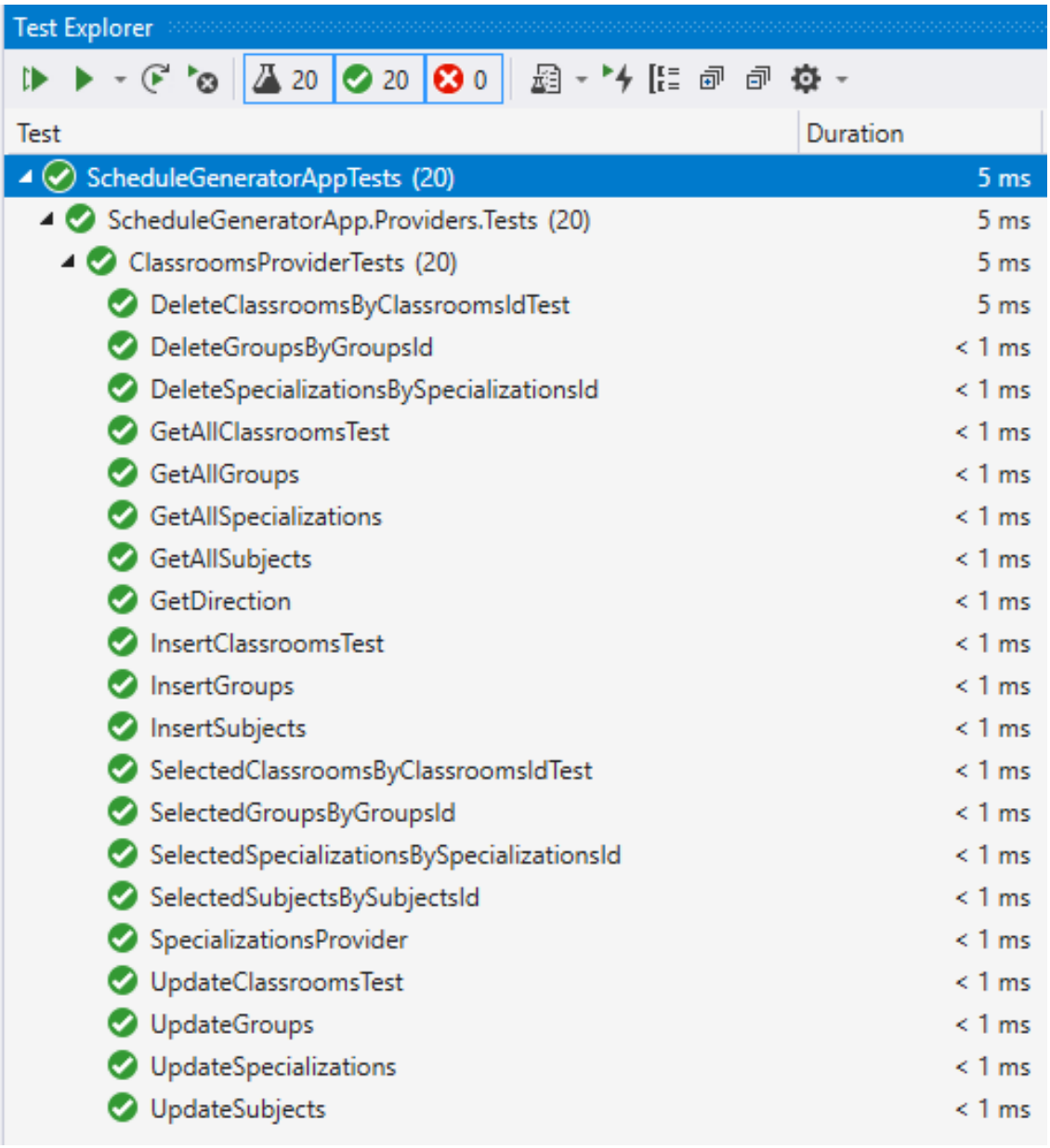


Рисунок 3.12 – Результати проведеного модульного тестування

3.5 Інструкція користувачеві програми

3.5.1 Вимоги до апаратного та програмного забезпечення

Для ефективного функціонування розробленого програмного продукту потрібно врахувати певні вимоги до апаратного та програмного забезпечення.

Вимоги до апаратного забезпечення

Система розроблена з огляду на середні параметри обладнання, доступного для сучасних користувачів. Таким чином, базові вимоги до апаратного забезпечення включають:

– процесор. Середнього класу процесор, спроможний виконувати операції в режимі реального часу без значних затримок. Рекомендується використовувати 64-бітні процесори для максимальної продуктивності;

– оперативна пам'ять. Мінімум 4GB RAM, рекомендовано 8GB або більше для оптимального виконання важких операцій;

– місце на диску. Не менше 1 GB вільного місця на диску для встановлення програми та зберігання вхідних та вихідних даних;

– монітор. Рекомендується монітор з роздільною здатністю не менше 1280 x 720 для зручного користування інтерфейсом програми;

– миша та клавіатура: необхідні для вводу даних та управління програмою.

Вимоги до програмного забезпечення

Розроблене програмне забезпечення базується на мові програмування C#, що диктує певні вимоги до програмного середовища:

– операційна система. Windows 7 або новіше з останніми оновленнями, які підтримують запуск .NET Framework;

– .NET Framework. Версія 4.5 або новіше, яка необхідна для виконання програм, написаних на мові C#. Це ПЗ зазвичай вже встановлене на сучасних версіях Windows;

– Microsoft Visual Studio. Для розробки та змін, якщо потрібно, рекомендується Visual Studio з встановленими оновленнями;

– система керування базами даних (СУБД). MS Access або сумісна система для зберігання та обробки вхідних даних та результатів.

Загалом, ці вимоги дозволяють користувачам з різними конфігураціями систем використовувати розроблене програмне забезпечення для автоматизації процесу складання навчального розкладу.

3.5.2 Використання програмного продукту

Після запуску додатка "Генератор розкладу" на екрані з'являється вікно авторизації, де користувач повинен ввести свій логін та пароль (рис. 3.13). Цей крок є важливим, оскільки він дозволяє ідентифікувати користувача та забезпечує безпеку його персональних даних.

У разі успішної авторизації, користувач отримує доступ до головного інтерфейсу додатка. В цьому інтерфейсі користувач може виконувати різноманітні дії відповідно до своєї ролі у системі.

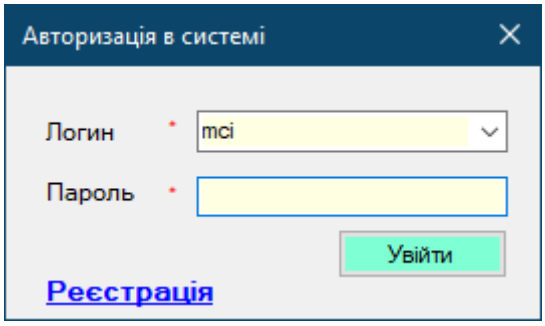


Рисунок 3.13 – Авторизація користувача в системі

В додатку "Генератор розкладу" доступна можливість реєстрації для користувачів, які ще не мають облікового запису. На екрані авторизації розміщена кнопка "Реєстрація", яку можна використовувати для переходу до вікна реєстрації, де необхідно ввести необхідну інформацію.

У формі реєстрації користувачу потрібно вказати своє ім'я, прізвище, логін та пароль. Ці дані використовуються для створення нового облікового

запису в системі. Після введення всіх необхідних даних користувач повинен натиснути кнопку "Зареєструватися".

Після успішної реєстрації користувач отримує свій новий обліковий запис та може використовувати його для входу в систему. Зареєстрований користувач отримує повний доступ до всіх функцій додатка "Генератор розкладу" і може здійснювати різні дії, пов'язані з управлінням розкладом занять, внесенням змін та іншими операціями, відповідно до свого рівня доступу в системі (рис. 3.14).

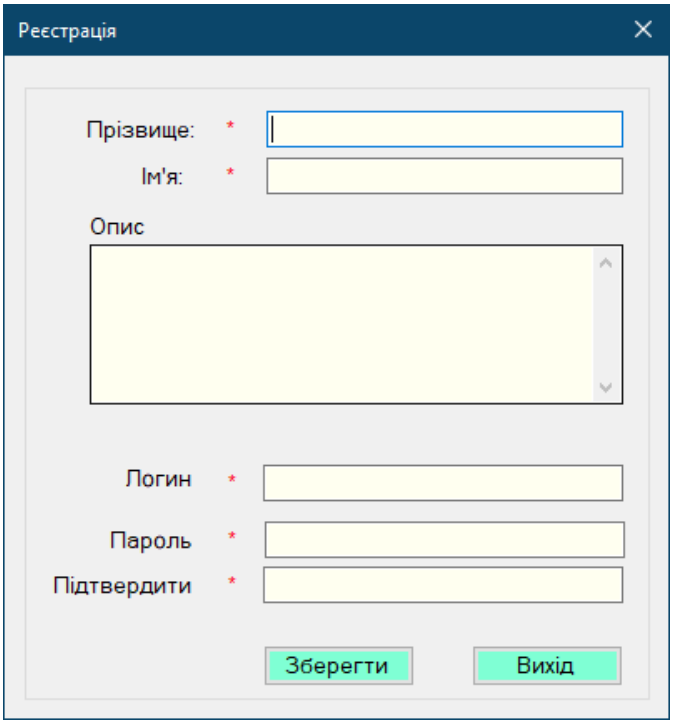


Рисунок 3.14 – Форма реєстрації користувачів

Додаток "Генератор розкладу" пропонує зручний інструмент для пошуку імен у вигляді випадаючого списку, який містить імена користувачів. Це забезпечує швидкий доступ до потрібного імені без необхідності вводити його повністю. Крім того, якщо користувач вводить ім'я з клавіатури, програма автоматично переміщує його на верхню частину списку, спрощуючи процес пошуку та дозволяючи швидше знайти потрібне ім'я.

Після успішної автентифікації користувача в системі, відкривається головне вікно програми. Це вікно містить основне меню з доступними опціями та функціями (рис. 3.15). Головне вікно є центральною точкою

взаємодії користувача з програмою, де він може виконувати різноманітні дії, пов'язані з управлінням розкладом занять, налаштуваннями, переглядом звітів та іншими функціями, які відповідають його ролям та правам доступу в системі. Головне вікно забезпечує зручний та централізований доступ до всіх функціональних можливостей програми.

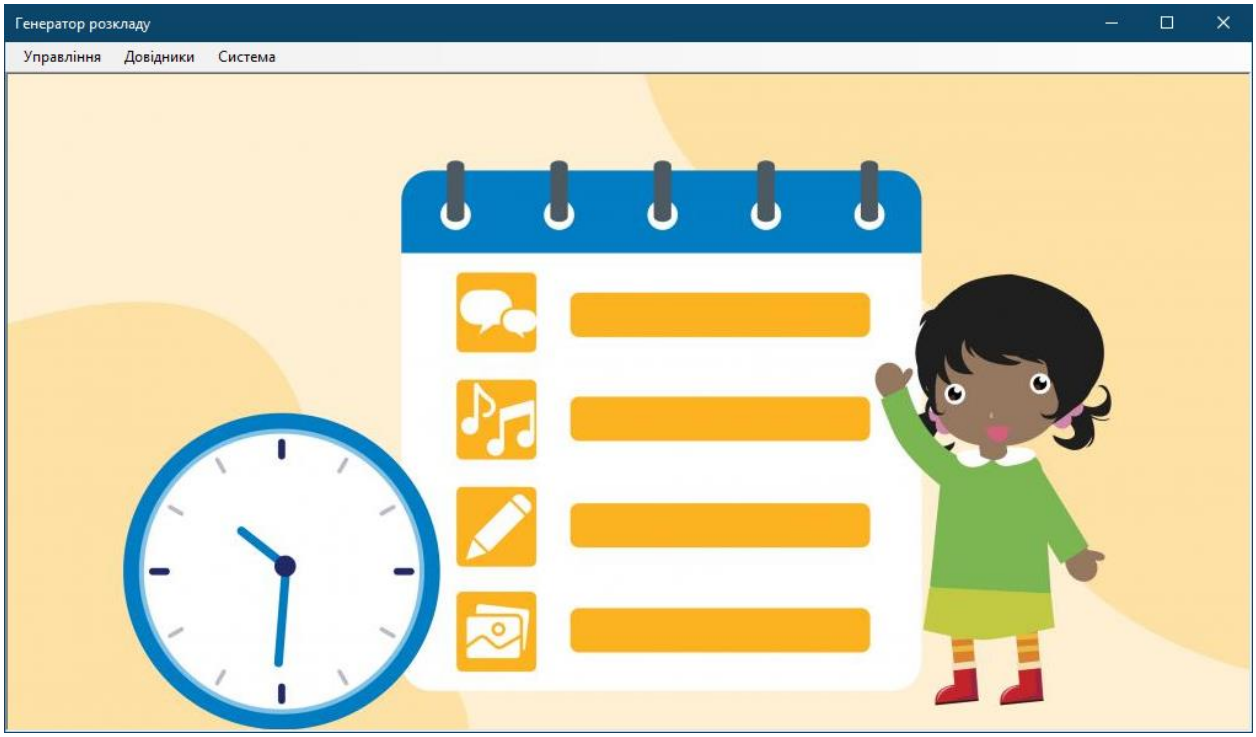


Рисунок 3.15 – Головне меню програми

У програмі реалізовано дві ролі: системний адміністратор та звичайний користувач. Системний адміністратор має привілеї для керування обліковими записами та перегляду подій системи.

Перш ніж почати використовувати систему, необхідно внести інформацію про викладачів, групи, аудиторії, спеціальності та предмети.

Для додавання нової навчальної групи користувачу системи потрібно перейти до меню програми "Довідники" -> "Групи". Після цього відкриється відповідна форма, яку можна побачити на рис. 3.16. В цій формі користувач може ввести необхідну інформацію про нову групу.

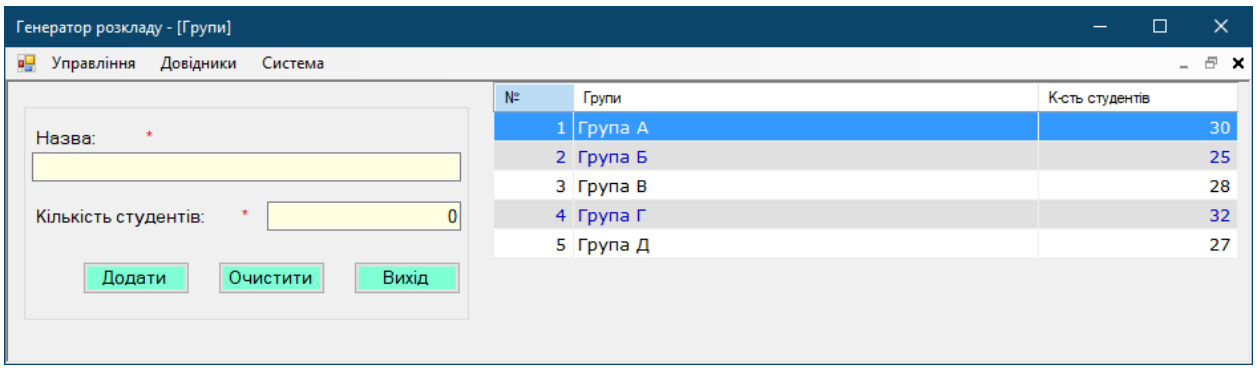


Рисунок 3.16 – Форма опрацювання даних про групи навчального закладу

Крім того, в системі існує можливість редагування збережених тем. Для цього користувачу потрібно в правій частині форми вибрати відповідну групу, що призведе до відкриття відповідної форми, яка зображена на рисунку 3.17. В цій формі користувач може змінити дані теми залежно від своїх потреб та вимог.

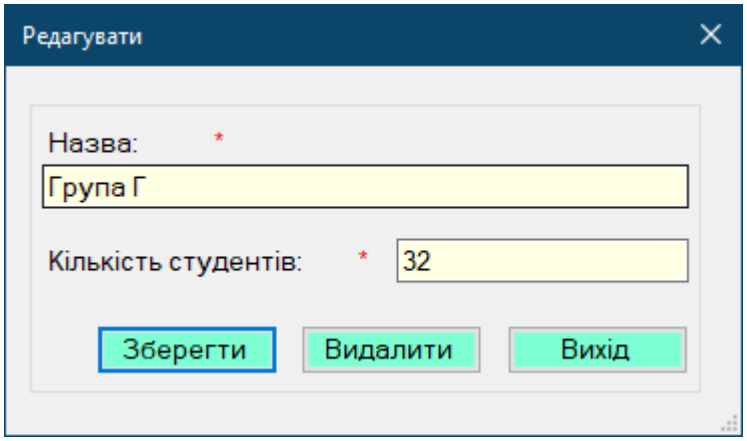


Рисунок 3.17 – Форма редагування інформації вибраної групи

У програмі "Генератор розкладу" також є форма, що призначена для роботи з даними про спеціальності навчального закладу. Ця форма надає користувачу можливість додавати та змінювати інформацію про спеціальності.

Для додавання нової спеціальності користувачу необхідно перейти до меню програми "Довідники" і обрати пункт "Спеціальності". Після цього відкриється відповідна форма, яка зображена на рисунку 3.18. У цій формі користувач може ввести необхідну інформацію про нову спеціальність, таку

як назва, опис, код тощо. Крім того, у формі доступні опції для редагування та видалення існуючих спеціальностей, що дозволяє користувачу змінювати дані відповідно до потреб та вимог.

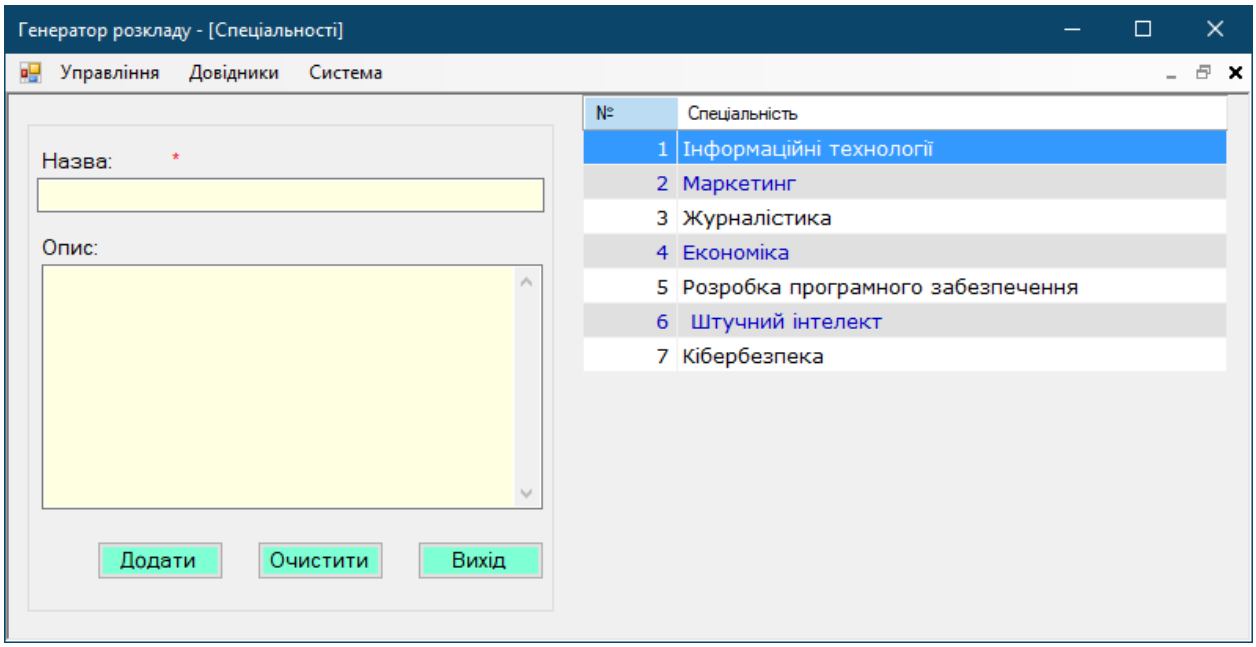


Рисунок 4.6 – Форма опрацювання інформації вибраної спеціальності

Форма, що використовується для опрацювання даних про спеціальності в програмі "Генератор розкладу", надає можливість користувачу змінювати збережені дані про спеціальності. Це здійснюється шляхом вибору потрібної спеціальності зі списку, який розташований на правій частині форми. Після вибору спеціальності, відкривається форма редагування, яка дозволяє користувачу внести необхідні зміни до даних про спеціальність.

У формі редагування, яка зображена на рисунку 3.18, користувач має можливість внести зміни до різних атрибутів спеціальності, таких як назва, опис, код тощо. Після внесення змін користувач може зберегти оновлену інформацію, щоб зміни набули чинності. Цей процес дозволяє користувачу актуалізувати дані про спеціальності відповідно до потреб та змін, які виникають в навчальному закладі.

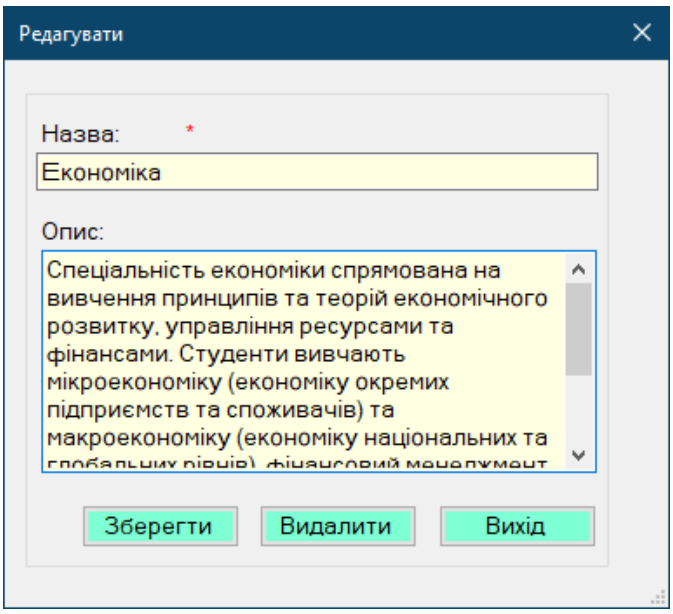


Рисунок 3.18 – Форма редагування інформації вибраної спеціальності

У програмі користувач має можливість опрацьовувати інформацію про аудиторії та предмети за допомогою відповідних форм, які зображені на рис. 3.19 і 3.20. Робота з цими формами відбувається за аналогічним принципом.

Форма для опрацювання даних про аудиторії, зображена на рис. 3.19, надає користувачу можливість додавати нові аудиторії або вносити зміни до існуючих. Аналогічно, форма для опрацювання даних про предмети, зображена на рисунку 3.20, дозволяє користувачу додавати нові предмети або редагувати вже існуючі.

Обидві форми мають схожий інтерфейс та функціонал. Користувач може внести необхідну інформацію про аудиторію або предмет, таку як назва, номер, опис та інші атрибути. Після внесення змін користувач може зберегти оновлені дані, щоб вони враховувалися в системі.

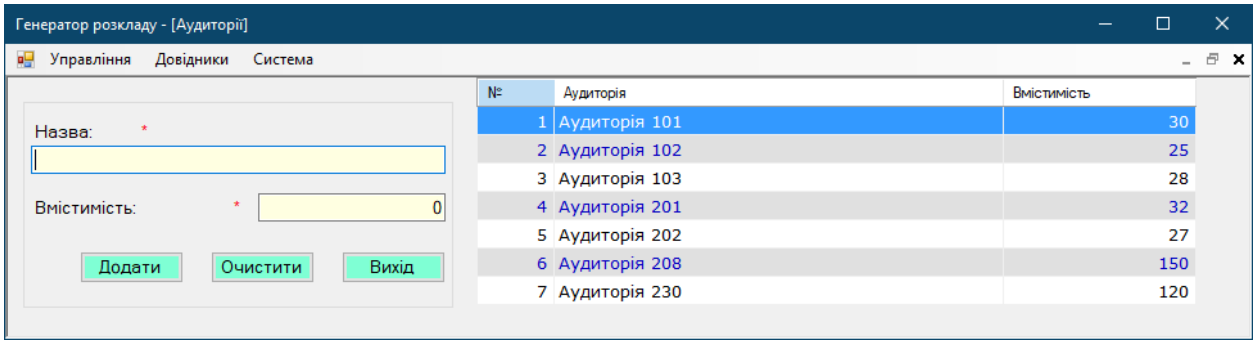


Рисунок 3.19 – Форма для опрацювання інформації вибраної аудиторії

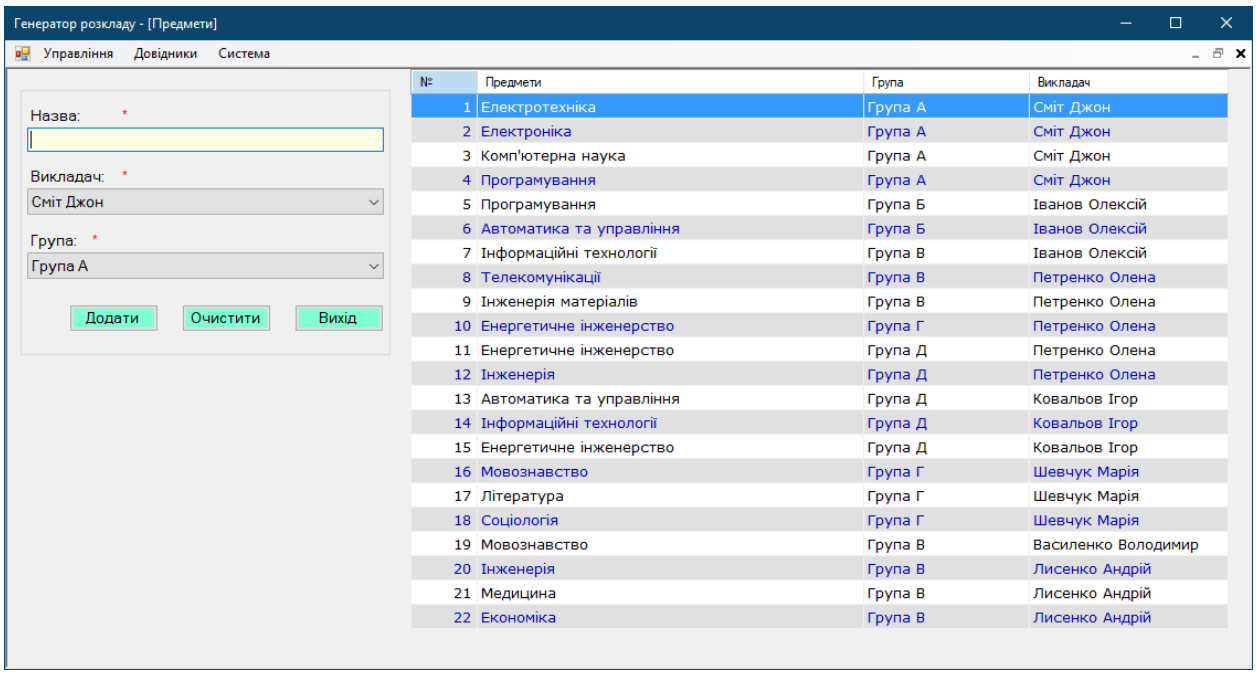


Рисунок 3.20 – Форма для опрацювання інформації вибраного предмету

Форма для опрацювання даних про викладачів надає можливість користувачу додавати нову інформацію про викладачів та переглядати список вже існуючих викладачів (рис. 3.21).

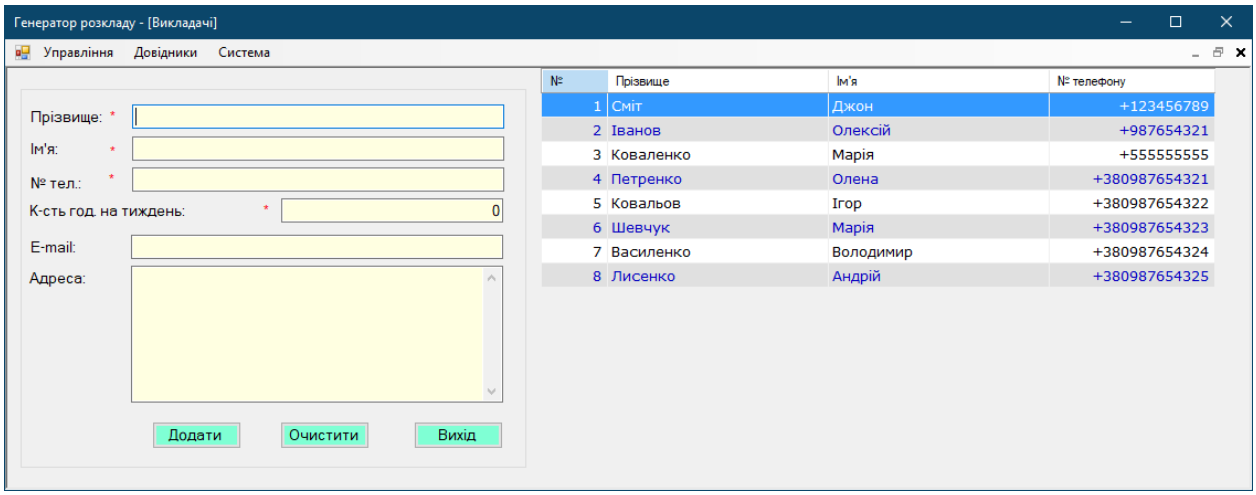


Рисунок 3.21 – Форма опрацювання інформації про викладачів

Ліва частина вікна призначена для додавання нової інформації про викладачів у програмі "Генератор розкладу". У цій частині знаходяться поля, де користувач може ввести дані про викладача, такі як прізвище, ім'я, телефон, адреса, кількість робочих годин на тиждень та електронна пошта.

Користувач може заповнити ці поля згідно своїх потреб і після цього натиснути кнопку "Додати", щоб зберегти нову інформацію про викладача.

Права частина вікна відображає список викладачів, які вже зареєстровані в системі. Кожен викладач представлений як окремий рядок у списку, де зазначені його прізвище, ім'я та контактні дані. Користувач має можливість переглядати список викладачів і здійснювати різні дії з ними, такі як вибір конкретного викладача для редагування даних або видалення викладача зі списку (рис. 3.22).

Такий розподіл вікна дозволяє користувачу зручно керувати інформацією про викладачів, додавати нових викладачів та редагувати чи видаляти існуючих зі списку, спрощуючи процес управління цими даними у контексті генерації розкладу занять.

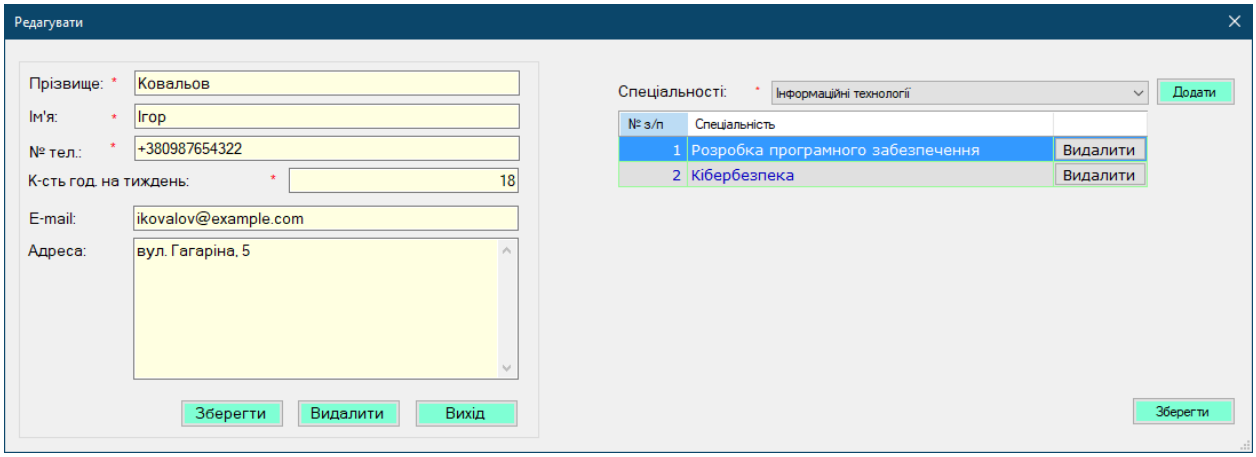


Рисунок 3.22 – Форма редагування інформації про викладачів

У даній формі також можна додавати спеціальності для викладачів.

Для створення розкладу занять на тиждень, користувачу програми потрібно виконати наступні кроки: перейти до меню "Управління", після чого обрати опцію "Розклад". Ці дії відкриють вікно, яке зображене на рисунку 3.23.

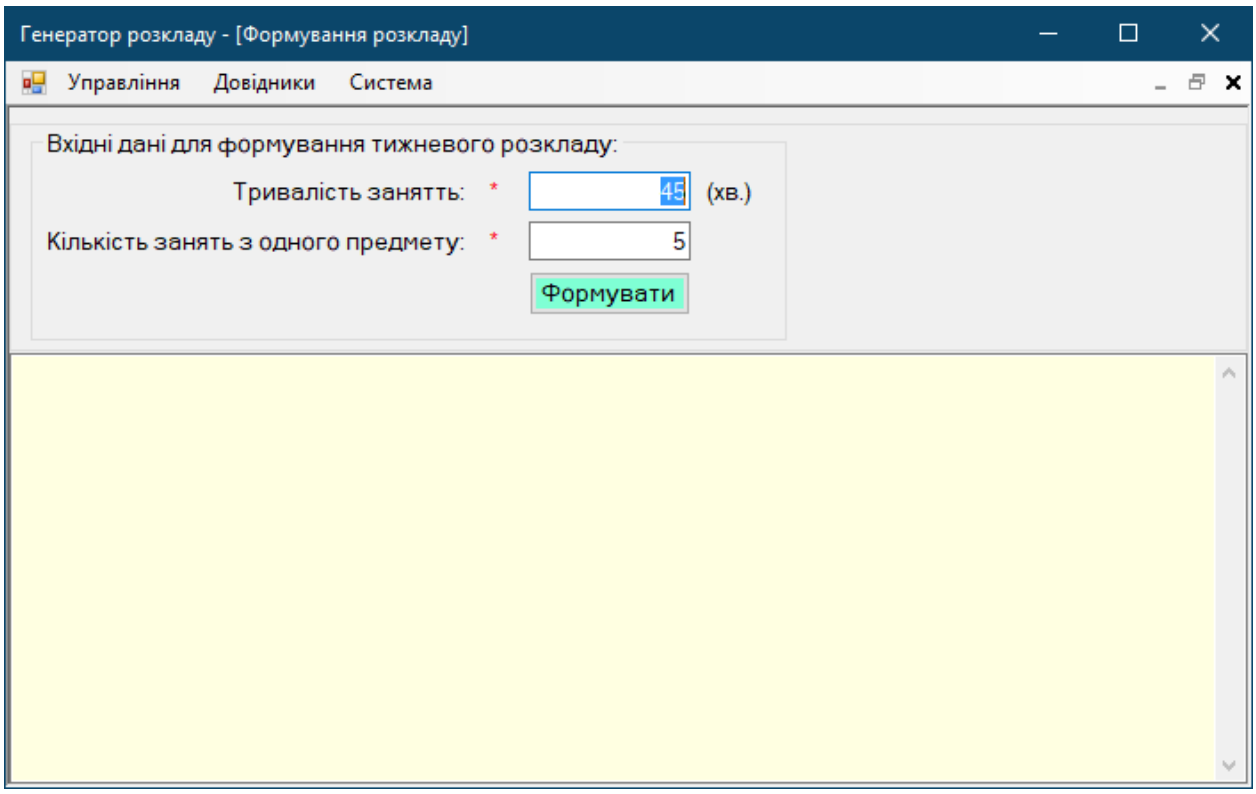
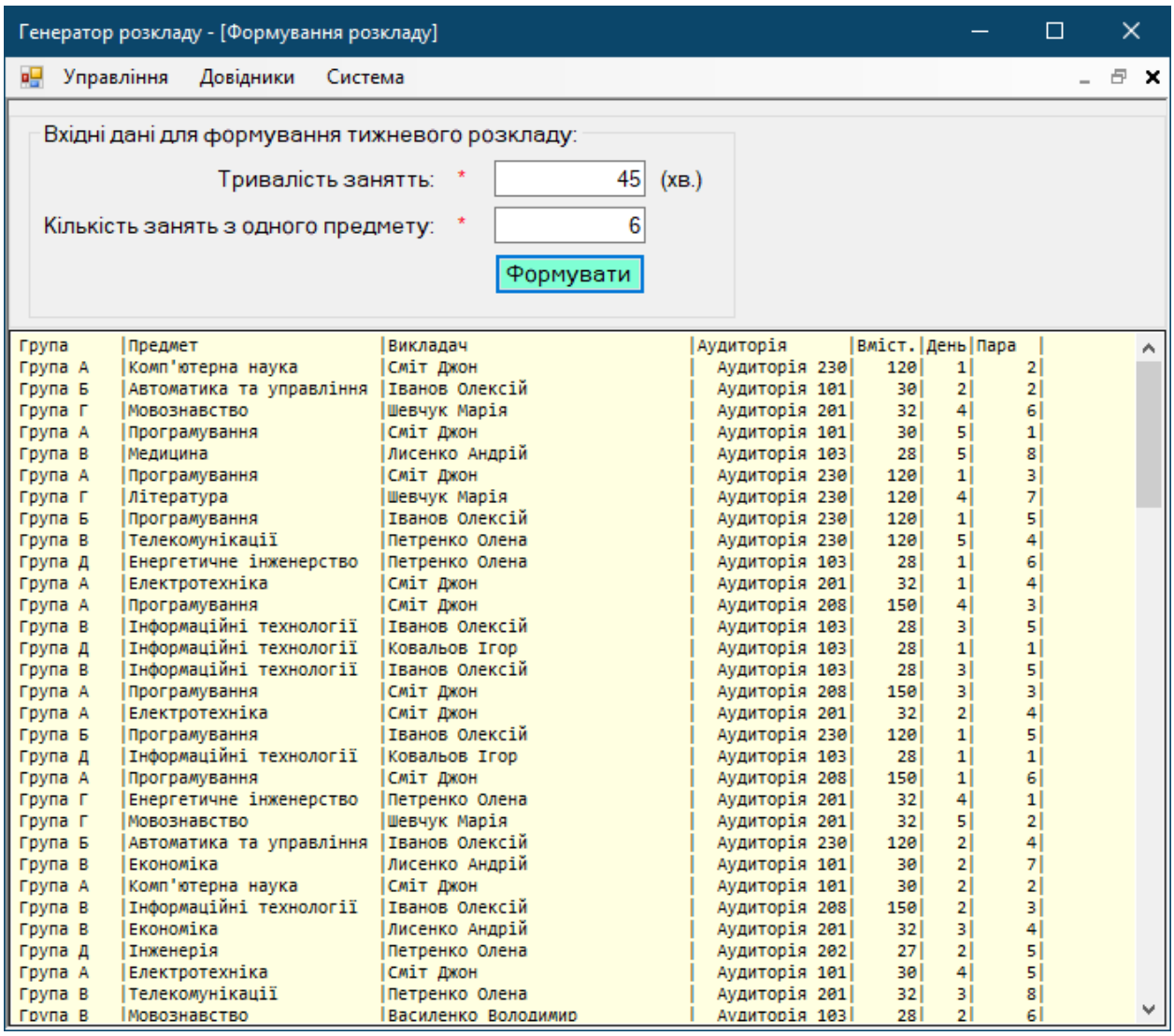


Рисунок 3.23 – Вікно для формування розкладів

Форма, призначена для складання розкладів у програмі "Генератор розкладу", надає користувачеві можливість вказати необхідні вхідні дані та згенерувати розклад занять для навчального закладу. У цій формі присутнє поле для введення тривалості заняття та кількості занять з одного предмету, що користувач може заповнити.

Після введення вхідних даних користувач натискає кнопку "Формувати", що запускає процес генерації розкладу занять. Результат генерації розкладу представлений у вигляді таблиці, де кожен рядок містить інформацію про конкретне заняття. В таблиці вказані дані про групу, предмет, викладача, аудиторію, вміст заняття, день тижня та номер пари (рис. 3.24).



Група	Предмет	Викладач	Аудиторія	Вміст.	День	Пара
Група А	Комп'ютерна наука	Сміт Джон	Аудиторія 230	120	1	2
Група Б	Автоматика та управління	Іванов Олексій	Аудиторія 101	30	2	2
Група Г	Мовознавство	Шевчук Марія	Аудиторія 201	32	4	6
Група А	Програмування	Сміт Джон	Аудиторія 101	30	5	1
Група В	Медицина	Лисенко Андрій	Аудиторія 103	28	5	8
Група А	Програмування	Сміт Джон	Аудиторія 230	120	1	3
Група Г	Література	Шевчук Марія	Аудиторія 230	120	4	7
Група Б	Програмування	Іванов Олексій	Аудиторія 230	120	1	5
Група В	Телекомунікації	Петренко Олена	Аудиторія 230	120	5	4
Група Д	Енергетичне інженерство	Петренко Олена	Аудиторія 103	28	1	6
Група А	Електротехніка	Сміт Джон	Аудиторія 201	32	1	4
Група А	Програмування	Сміт Джон	Аудиторія 208	150	4	3
Група В	Інформаційні технології	Іванов Олексій	Аудиторія 103	28	3	5
Група Д	Інформаційні технології	Ковальов Ігор	Аудиторія 103	28	1	1
Група В	Інформаційні технології	Іванов Олексій	Аудиторія 103	28	3	5
Група А	Програмування	Сміт Джон	Аудиторія 208	150	3	3
Група А	Електротехніка	Сміт Джон	Аудиторія 201	32	2	4
Група Б	Програмування	Іванов Олексій	Аудиторія 230	120	1	5
Група Д	Інформаційні технології	Ковальов Ігор	Аудиторія 103	28	1	1
Група А	Програмування	Сміт Джон	Аудиторія 208	150	1	6
Група Г	Енергетичне інженерство	Петренко Олена	Аудиторія 201	32	4	1
Група Г	Мовознавство	Шевчук Марія	Аудиторія 201	32	5	2
Група Б	Автоматика та управління	Іванов Олексій	Аудиторія 230	120	2	4
Група В	Економіка	Лисенко Андрій	Аудиторія 101	30	2	7
Група А	Комп'ютерна наука	Сміт Джон	Аудиторія 101	30	2	2
Група В	Інформаційні технології	Іванов Олексій	Аудиторія 208	150	2	3
Група В	Економіка	Лисенко Андрій	Аудиторія 201	32	3	4
Група Д	Інженерія	Петренко Олена	Аудиторія 202	27	2	5
Група А	Електротехніка	Сміт Джон	Аудиторія 101	30	4	5
Група В	Телекомунікації	Петренко Олена	Аудиторія 201	32	3	8
Група В	Мовознавство	Василенко Володимир	Аудиторія 103	28	2	6

Рисунок 3.24 – Згенерований розклад

Управління обліковими записами є важливою функцією в програмі "Генератор розкладу" і доступна для користувачів з роллю "Системний адміністратор". Ця функція дозволяє здійснювати контроль над обліковими записами користувачів системи.

Для доступу до управління обліковими записами, користувач з роллю "Системний адміністратор" повинен обрати опцію "Користувачі" у головному меню програми (див. рисунок 3.25). Після цього відкривається вікно управління обліковими записами, де можна переглядати список існуючих облікових записів користувачів.

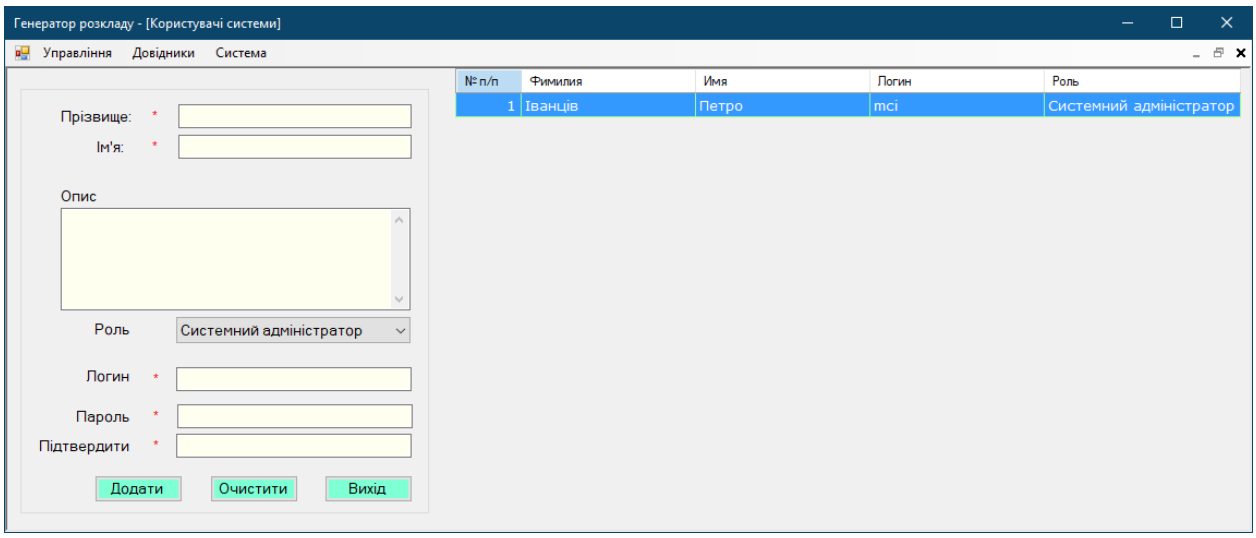


Рисунок 3.25 – Управління обліковими записами

У вікні управління обліковими записами, користувач з роллю "Системний адміністратор" може здійснювати такі дії:

– переглядати інформацію про облікові записи користувачів, таку як ім'я, роль, права доступу та обмеження;

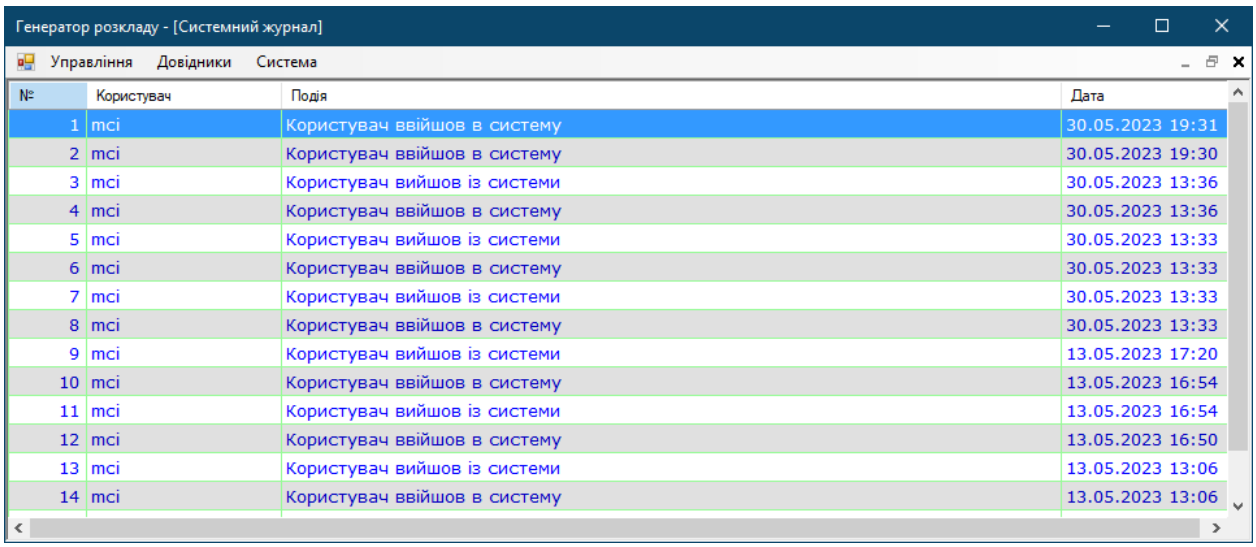
– редагувати інформацію облікових записів, змінюючи параметри, наприклад, ролі, права доступу або обмеження;

– створювати нові облікові записи для користувачів, встановлюючи їх ролі та параметри доступу;

– видаляти старі облікові записи, якщо це необхідно.

Ця функція дозволяє системному адміністратору здійснювати повний контроль над обліковими записами користувачів і налаштовувати їхні параметри відповідно до вимог та потреб системи.

Функція "Події", розташована в меню "Система" (рис. 3.26), дозволяє відстежувати активність користувачів та їхні дії. Вона надає можливість системному адміністратору контролювати внесення змін до облікових записів, налаштування системи та інші дії користувачів. За допомогою функції "Події", адміністратор може відслідковувати історію подій у системі, що допомагає забезпечити безпеку та виявляти можливі порушення.



№	Користувач	Подія	Дата
1	mci	Користувач ввійшов в систему	30.05.2023 19:31
2	mci	Користувач ввійшов в систему	30.05.2023 19:30
3	mci	Користувач вийшов із системи	30.05.2023 13:36
4	mci	Користувач ввійшов в систему	30.05.2023 13:36
5	mci	Користувач вийшов із системи	30.05.2023 13:33
6	mci	Користувач ввійшов в систему	30.05.2023 13:33
7	mci	Користувач вийшов із системи	30.05.2023 13:33
8	mci	Користувач ввійшов в систему	30.05.2023 13:33
9	mci	Користувач вийшов із системи	13.05.2023 17:20
10	mci	Користувач ввійшов в систему	13.05.2023 16:54
11	mci	Користувач вийшов із системи	13.05.2023 16:54
12	mci	Користувач ввійшов в систему	13.05.2023 16:50
13	mci	Користувач вийшов із системи	13.05.2023 13:06
14	mci	Користувач ввійшов в систему	13.05.2023 13:06

Рисунок 3.26 – Події системи

Системний журнал є невід'ємною складовою системи і містить інформацію про події, що відбуваються в системі. Його використання є важливим для виявлення потенційних проблем безпеки, вразливостей та відстеження дій користувачів. Він дозволяє системним адміністраторам переглядати записи про події, що відбуваються в системі. Це можуть бути події, які стосуються несанкціонованого доступу, змін налаштувань, створення та видалення облікових записів, виконання основних операцій та багато іншого.

Аналіз системного журналу дозволяє виявити можливі проблеми безпеки, що можуть виникнути в системі, та сприяє швидкій реакції на них. Наприклад, якщо системний журнал відображає незвичну активність або спроби несанкціонованого доступу, адміністратор може вжити відповідних заходів для захисту системи та її даних.

Крім того, перегляд системного журналу дозволяє відстежувати дії користувачів у системі. Це може бути корисним для контролю за їхніми діями та виявлення можливих порушень політик безпеки або недостовірних дій.

Щодо зміни облікових даних, користувачі можуть скористатися меню "Персоналізація", де вони можуть змінити своє ім'я, пароль та інші особисті

дані. Це надає можливість користувачам змінювати свої дані в будь-який момент за необхідності.

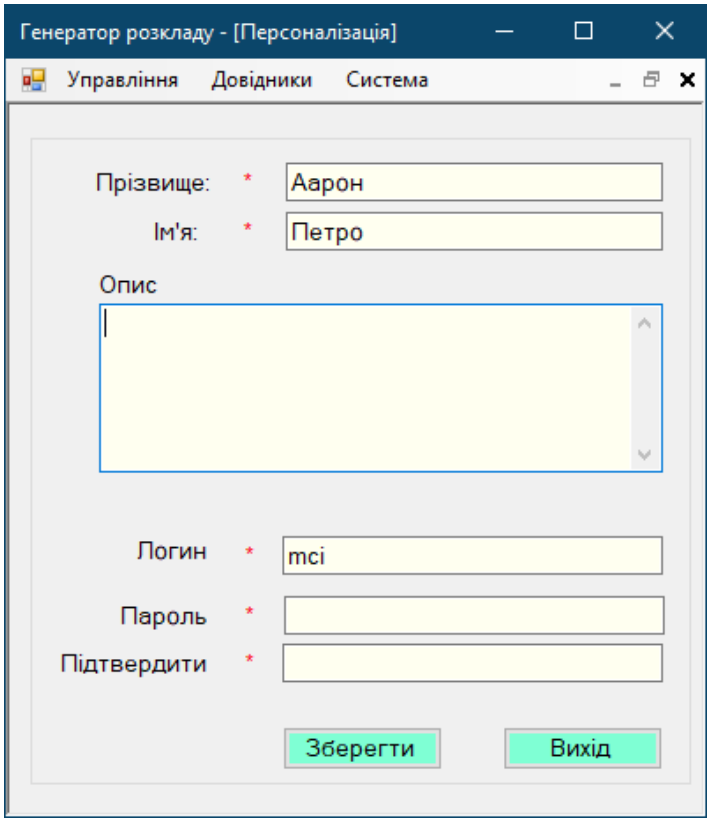


Рисунок 3.27 – Зміна даних персоналізації користувача

Закриття програми здійснюється за допомогою опції "Вихід", яка доступна у меню "Управління". Цей крок є важливим для забезпечення безпеки та правильного завершення роботи програми, а також для запобігання втраті даних та можливим проблемам з безпекою. Користувач, вибравши опцію "Вихід", може безпечно завершити роботу з додатком і забезпечити збереження своїх даних та налаштувань.

3.6 Висновок

В даному розділі розглянуто практичні аспекти розробки програмного продукту, що автоматизує процес складання навчального розкладу. Першим етапом було проведено проєктування бази даних з використанням MS Access, яка забезпечує необхідний функціонал для зберігання, обробки та відтворення вхідних даних та результатів.

Подальша розробка системи включала створення алгоритму додатку, основаного на генетичному алгоритмі, що дозволяє генерувати оптимальний розклад занять, враховуючи всі вхідні параметри та обмеження.

Наступний крок об'єднав розробку основних модулів системи, що забезпечують її функціональність та взаємодію з користувачем. Зокрема, були реалізовані модулі введення даних, обробки даних, генерації розкладу та відображення результатів.

Напружений процес розробки було доповнено функціональним та модульним тестуванням, щоб забезпечити стабільність роботи та коректність виконання основних функцій програмного продукту.

Закінчується розділ детальною інструкцією користувача програми, що допоможе зрозуміти вимоги до апаратного та програмного забезпечення, а також основи використання програмного продукту. Такий підхід дозволить користувачам швидко ознайомитися з системою і почати ефективно використовувати її для складання оптимального розкладу занять.

ВИСНОВКИ

Отже, в процесі випускної роботи було проведено глибоке дослідження в області автоматизації процесу складання навчального розкладу. Це включає ретельний аналіз викликів та проблем, що виникають при ручному складанні розкладу, та порівняння доступних на ринку систем. Зокрема, було розглянуто такі системи як Prime Timetable, Mimosa Scheduling Software, Celcat Timetabler. Аналіз дозволив з'ясувати вимоги до нової системи та вибрати найефективніші алгоритми для її реалізації, зокрема генетичний алгоритм.

Технологічною основою програмного продукту стала мова програмування C#, що вирізняється своєю надійністю та широкими можливостями. Для забезпечення гнучкості та продуктивності розробки було обрано середовище Visual Studio. Основу архітектури продукту складає трьохрівнева структура, що включає бізнес-логіку (BLL), інтерфейс користувача (UI) та доступ до даних (DAL), кожен з яких був детально розроблений та налаштований.

У процесі практичної реалізації системи була розроблена база даних в MS Access, розроблено генетичний алгоритм для автоматичного складання розкладу, реалізовано основні модулі системи. Усі елементи системи були пройшли модульне та функціональне тестування для забезпечення стабільної роботи.

Окрім технічної реалізації, важливим аспектом роботи було створення зрозумілої та детальної інструкції користувача, що включає не лише інформацію про використання програмного продукту, але й вимоги до апаратного та програмного забезпечення.

У загальному контексті, результати випускної роботи підтверджують, що розробка високоякісного програмного продукту для автоматизації процесу складання навчального розкладу є складним, але вкрай актуальним завданням.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Рибак І. І. Системний підхід до аналізу організаційної структури факультету ЗВО. Розвиток освіти і науки: проблеми, теорія, досвід і перспективи : матер. ІІ заоч. Всеукр. наук.-практ. конф. Електронне видання / за ред. В. Ф. Русакова, І. М. Зарішняк. Вінниця: ДонНУ імені Василя Стуса, 2021. С. 37-40. URL: https://jron.donnu.edu.ua/index .
2. Снитюк В.Є. Про особливості формування цільової функції та обмежень в задачі складання розкладу занять / Снитюк В.Є., Сіпко Є.Н. // Математичні машини і системи – 2014 - №3 – С. 67-76
3. Снитюк В.Є. Аспекти формування цільової функції в задачі складання розкладу занять у вищих навчальних закладах на основі суб'єктивних переваг / Снитюк В.Є., Сіпко Є.Н. // Автоматика. Автоматизація. Електротехнічні комплекси і системи - 2013 – №2 – С.98-104
4. Деканова М.В. Математична модель и алгоритм побудови розкладу навчальних занятий університету / Деканова М.В. // Вісник Полоцкого державного університету. Серія С. – 2013. – №12. – С. 24-33.
5. Бабкіна Т.С. Задача складання розкладу: рішення на основі багатоагентного підходу / Бабкіна Т.С. // Бізнес-інформатика. – 2008. – №1. – С.23-28.
6. Prime Timetable [Електронний ресурс] – Режим доступу: https://primetimetable.com/#intro&id=b918a5b4-7da5-4809-9e51-1722b3ea3207 (дата звернення 01.05.2023).
7. Using Prime Timetable [Електронний ресурс] – Режим доступу: https://primetimetable.com/pdf/prime-timetable-user-guide.pdf (дата звернення 01.05.2023).
8. Mimosa Scheduling Software [Електронний ресурс] – Режим доступу: https://download.cnet.com/Mimosa-Scheduling-Software-Free-Edition/3000-20414_4-75186172.html (дата звернення 01.05.2023).

9. Related System - Mimosa Scheduling Software [Електронний ресурс] – Режим доступу: http://users.csc.calpoly.edu/~gfisher/classes/309/specs/scheduler-f07-afternoon/requirements/related3.html (дата звернення 01.05.2023).

10. Celcat Timetabler [Електронний ресурс] – Режим доступу: https://www.celcat.com/ (дата звернення 01.05.2023).

11. CELCAT Timetabler - Adapt IT Education [Електронний ресурс] – Режим доступу: https://education.adaptit.tech/solutions/celcat-timetabler/ (дата звернення 01.05.2023).

12. Т.В. Січко, І.І. Рибак. Системний підхід до аналізу організаційних структур. Вісник ХНУ. Хмельницький, 2021. (№4). С. 70-74.

13. Коцовський В. М. Технології розподілених систем та паралельних обчислень. Частина І: Методичний посібник — Ужгород: Видавництво УжНУ "Говерла", 2017. — 51 с.

14. Kenneth H. Rosen Discrete Mathematics and Its Applications 2002 by McGrawHill Science, 928 p.

15. Alex F Bielajew. Fundamentals of the Monte Carlo method for neutral and charged particle transport. 2001 W. M. C. Foulkes, L. Mitas, R. J. Needs and G. Rajagopal Quantum Monte Carlo simulations of solids, — Reviews of Modern Physics 73 (2001) 33.

16. Kent D. Lee. Data Structures and Algorithms with Python / Kent D. Lee, Steve Hubbard., – Springer, 2015. – 363 c.

17. Юрчак І.Ю., Москович Т.Р. Дослідження генетичних алгоритмів та їх застосування їхнього в автоматизованій системі розподілу навантаження для викладачів і студентів. [Електронний ресурс] – Режим доступу: http://eom.lp.edu.ua/sntk/doc/ksm2018/moskovytch.pdf (дата звернення 01.05.2023).

18. MATT WATSON. What is C# used for? [Електронний ресурс] / MATT WATSON // STACKIFY PRODUCT & COMPANY UPDATES. – 2020.

– Режим доступу до ресурсу: https://stackify.com/what-is-c-used-for/ (дата звернення 01.05.2023).

19. MS Access [Електронний ресурс] – Режим доступу до ресурсу: https://www.microsoft.com/uk-ua/microsoft-365/access (дата звернення 01.05.2023).

20. C# [Електронний ресурс] – Режим доступу до ресурсу: https://beetroot.academy/blog/courses/what-is-C (дата звернення 01.05.2023).

21. C++ [Електронний ресурс] – Режим доступу: http://www.znannya.org/?view=Cpp_basics (дата звернення 01.05.2023).

22. Sun Microsystems [Електронний ресурс] – Режим доступу: https://www.britannica.com/topic/Sun-Microsystems-Inc (дата звернення 01.05.2023).

23. Java [Електронний ресурс] – Режим доступу: https://dou.ua/lenta/articles/how-to-learn-java/ (дата звернення 01.05.2023).

24. Visual Studio Code [Електронний ресурс]. — Режим доступу: https://code.visualstudio.com/ (дата звернення 01.05.2023).

25. Eclipse [Електронний ресурс] – Режим доступу до ресурсу: https://www.eclipse.org/ide/ (дата звернення 01.05.2023).

26. PyCharm [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/lp/pycharm-pro/ (дата звернення 01.05.2023).

27. Creating a Business Logic Layer [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/web-forms/overview/data-access/introduction/creating-a-business-logic-layer-cs (дата звернення 01.05.2023).

28. Creating a Data Access Layer [Електронний ресурс]. – 2020. – Режим доступу до ресурсу https://learn.microsoft.com/en-us/aspnet/web-forms/overview/data-access/introduction/creating-a-data-access-layer-cs (дата звернення 01.05.2023).

29. UML examples and algorithms [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу http://www.omg-portal.ru/articlematerial17 (дата звернення 01.05.2023).

30. Perkins B. What is ERP? Key features of top enterprise resource planning systems [Електронний ресурс] / Bart Perkins. – 2020. – Режим доступу до ресурсу: https://www.cio.com/article/2439502/what-is-erp-key-features-of-top-enterprise-resourceplanning-systems.html

Додаток А. Лістинги програми

Лістинг 1. Код класу «TeachersProvider»

```
using ScheduleGeneratorApp.AppCode;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.OleDb;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ScheduleGeneratorApp.Providers {
  class TeachersProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];
    public void InsertTeachers(string LastName, string FirstName, string Phone,
      string Address, string Email, int Workload) {
      string SqlString = "INSERT INTO Teachers (LastName, FirstName, Phone, Address, " +
        "Email, Workload) Values(?, ?, ?, ?, ?, ?)";

      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("LastName", LastName);
          cmd.Parameters.AddWithValue("FirstName", FirstName);
          cmd.Parameters.AddWithValue("Phone", Phone);
          cmd.Parameters.AddWithValue("Address", Address);
          cmd.Parameters.AddWithValue("Email", Email);
          cmd.Parameters.AddWithValue("Workload", Workload);
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

    public List<Teachers> GetAllTeachers() {
      int i = 0;
      string SqlString = "SELECT * FROM Teachers";

      List<Teachers> listAllTeachers = new List<Teachers>();
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          conn.Open();
          using (OleDbDataReader reader = cmd.ExecuteReader()) {
            while (reader.Read()) {
              Teachers oneTeachers = new Teachers();
              oneTeachers.Number = ++i;
              oneTeachers.TeachersId = Convert.ToInt32(reader["TeachersId"]);
              oneTeachers.FirstName = reader["FirstName"].ToString();
```

```
            oneTeachers.LastName = reader["LastName"].ToString();
            oneTeachers.FIO = oneTeachers.LastName + " " + oneTeachers.FirstName;
            oneTeachers.Phone = reader["Phone"].ToString();
            oneTeachers.Address = reader["Address"].ToString();
            oneTeachers.Email = reader["Email"].ToString();
            oneTeachers.Workload = Convert.ToInt32(reader["Workload"]);
            listAllTeachers.Add(oneTeachers);
          }
        }
        conn.Close();
      }
    }

    if (listAllTeachers.Count == 0) {
      Teachers noTeachers = new Teachers();
      noTeachers.TeachersId = 0;
      noTeachers.Message = NamesMy.NoDataNames.NoDataInTeachers;
      listAllTeachers.Add(noTeachers);
    }
    return listAllTeachers;
  }

  public Teachers SelectedTeachersByTeachersId(int TeachersId) {
    string SqlString = "SELECT * FROM Teachers Where TeachersId=" +
TeachersId.ToString();

    Teachers oneTeachers = new Teachers();
    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        using (OleDbDataReader reader = cmd.ExecuteReader()) {
          while (reader.Read()) {
            oneTeachers.TeachersId = Convert.ToInt32(reader["TeachersId"]);
            oneTeachers.FirstName = reader["FirstName"].ToString();
            oneTeachers.LastName = reader["LastName"].ToString();
            oneTeachers.FIO = oneTeachers.LastName + " " + oneTeachers.FirstName;
            oneTeachers.Phone = reader["Phone"].ToString();
            oneTeachers.Address = reader["Address"].ToString();
            oneTeachers.Email = reader["Email"].ToString();
            oneTeachers.Workload = Convert.ToInt32(reader["Workload"]);
          }
        }
      }
      conn.Close();
    }
    return oneTeachers;
  }

  public void UpdateTeachers(string LastName, string FirstName, string Phone, string Address,
string Email, int Workload, int TeachersId) {
    string SqlString = "UPDATE Teachers SET FirstName=?, LastName=?, Phone=?, " +
 "Address=?, Email=?, Workload=? WHERE TeachersId=?";
```

```
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("FirstName", FirstName);
          cmd.Parameters.AddWithValue("LastName", LastName);
          cmd.Parameters.AddWithValue("Phone", Phone);
          cmd.Parameters.AddWithValue("Address", Address);
          cmd.Parameters.AddWithValue("Email", Email);
          cmd.Parameters.AddWithValue("Workload", Workload);
          cmd.Parameters.AddWithValue("TeachersId", TeachersId);
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

    public void DeleteTeachersByTeachersId(int TeachersId) {
      string SqlString = "DELETE FROM Teachers WHERE TeachersId=" +
TeachersId.ToString();
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

  }
}



public class Teachers {
  private int _Number;
  private int _TeachersId;
  private string _FirstName;
  private string _LastName;
  private string _Phone;
  private string _Address;
  private string _Email;
  private string _FIO;
  private int _Workload;
  private string _Message;

  public Teachers() {
    _Number = 0;
    _TeachersId = 0;
    _FirstName = String.Empty;
    _LastName = String.Empty;
```

```
    _Phone = String.Empty;
    _Address = String.Empty;
    _Email = String.Empty;
    _FIO = String.Empty;
    _Workload = 0;
    _Message = String.Empty;
   }

   public int Number {
    set { _Number = value; }
    get { return _Number; }
   }
   public int TeachersId {
    set { _TeachersId = value; }
    get { return _TeachersId; }
   }
   public string FirstName {
    set { _FirstName = value; }
    get { return _FirstName; }
   }
   public string LastName {
    set { _LastName = value; }
    get { return _LastName; }
   }
   public string Phone {
    set { _Phone = value; }
    get { return _Phone; }
   }
   public string Address {
    set { _Address = value; }
    get { return _Address; }
   }
   public string Email {
    set { _Email = value; }
    get { return _Email; }
   }
   public string FIO {
    set { _FIO = value; }
    get { return _FIO; }
   }
   public int Workload {
    set { _Workload = value; }
    get { return _Workload; }
   }
   public string Message {
    set { _Message = value; }
    get { return _Message; }
   }
  }
```

Лістинг 2. Код класу «SpecializationsLProvider»

```
using System;
using System.Collections.Generic;
using System.Data.OleDb;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ScheduleGeneratorApp.Providers {
  class SpecializationsLProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];
    private SpecializationsProvider _SpecializationsProvider = new SpecializationsProvider();
    private List<Specializations> _SpecializationsList = new List<Specializations>();



    public void InsertBatchSpecializationsLList(List<SpecializationsL> SpecializationsLList, int
TeachersId) {
      DeleteAllSpecializationsFromGroupByTeachersId(TeachersId);
      OleDbConnection connection = new OleDbConnection(_ConnString);
      string query = "INSERT into SpecializationsL (TeachersId, SpecializationsId) " +
        "VALUES (?, ?)";
      OleDbCommand command = new OleDbCommand(query, connection);
      connection.Open();
      for (int i = 0; i < SpecializationsLList.Count; i++) {
        command.Parameters.AddWithValue("TeachersId", SpecializationsLList[i].TeachersId);
        command.Parameters.AddWithValue("SpecializationsId",
SpecializationsLList[i].SpecializationsId);
        command.ExecuteNonQuery();
        while (command.Parameters.Count > 0) {
          command.Parameters.RemoveAt(0);
        }
      }
      connection.Close();
    }

    public List<SpecializationsL> GetAllSpecializationsLByTeachersId(int TeachersId) {
      int i = 0;
      _SpecializationsList = _SpecializationsProvider.GetAllSpecializations();
      List<SpecializationsL> SpecializationsLList = new List<SpecializationsL>();
      string sqlExpression = "SELECT * FROM SpecializationsL WHERE TeachersId=" +
TeachersId;
      using (OleDbConnection connection = new OleDbConnection(_ConnString)) {
        connection.Open();
        OleDbCommand command = new OleDbCommand(sqlExpression, connection);
        OleDbDataReader reader = command.ExecuteReader();

        if (reader.HasRows) {
          while (reader.Read()) {
            SpecializationsL selectedGroups = new SpecializationsL();
            selectedGroups.Number = ++i;
            selectedGroups.SpecializationsLId = Convert.ToInt32(reader["SpecializationsLId"]);
```

```
            selectedGroups.TeachersId = Convert.ToInt32(reader["TeachersId"]);
            selectedGroups.SpecializationsId = Convert.ToInt32(reader["SpecializationsId"]);
            SpecializationsLList.Add(selectedGroups);
          }
        }
        reader.Close();
      }

      for (int j = 0; j < SpecializationsLList.Count; j++) {
        SpecializationsLList[j].SpecializationsName =
GetSpecializationsName(SpecializationsLList[j].SpecializationsId, _SpecializationsList);
      }
      return SpecializationsLList;
    }

    public List<SpecializationsL> GetAllSpecializationsL() {
      int i = 0;
      _SpecializationsList = _SpecializationsProvider.GetAllSpecializations();
      List<SpecializationsL> SpecializationsLList = new List<SpecializationsL>();
      string sqlExpression = "SELECT * FROM SpecializationsL ";
      using (OleDbConnection connection = new OleDbConnection(_ConnString)) {
        connection.Open();
        OleDbCommand command = new OleDbCommand(sqlExpression, connection);
        OleDbDataReader reader = command.ExecuteReader();

        if (reader.HasRows) {
          while (reader.Read()) {
            SpecializationsL selectedGroups = new SpecializationsL();
            selectedGroups.Number = ++i;
            selectedGroups.SpecializationsLId = Convert.ToInt32(reader["SpecializationsLId"]);
            selectedGroups.TeachersId = Convert.ToInt32(reader["TeachersId"]);
            selectedGroups.SpecializationsId = Convert.ToInt32(reader["SpecializationsId"]);
            SpecializationsLList.Add(selectedGroups);
          }
        }
        reader.Close();
      }

      for (int j = 0; j < SpecializationsLList.Count; j++) {
        SpecializationsLList[j].SpecializationsName =
GetSpecializationsName(SpecializationsLList[j].SpecializationsId, _SpecializationsList);
      }
      return SpecializationsLList;
    }



    private string GetSpecializationsName(int SpecializationsId, List<Specializations>
SpecializationsList) {
      for (int i = 0; i < SpecializationsList.Count; i++) {
        if (SpecializationsId == SpecializationsList[i].SpecializationsId) {
          return SpecializationsList[i].SpecializationsName;
```

```
        }
      }
      return null;
    }


    public void DeleteAllSpecializationsFromGroupByTeachersId(int TeachersId) {
      string sqlExpression = "DELETE  FROM SpecializationsL WHERE TeachersId=" +
TeachersId;
      using (OleDbConnection connection = new OleDbConnection(_ConnString)) {
        connection.Open();
        OleDbCommand command = new OleDbCommand(sqlExpression, connection);
        command.ExecuteNonQuery();
        connection.Close();
      }
    }

  }
}


public class SpecializationsL {
  private int _Number;
  private int _SpecializationsLId;
  private int _TeachersId;
  private int _SpecializationsId;
  private string _SpecializationsName;
  private string _Message;

  public SpecializationsL() {
    _Number = 0;
    _SpecializationsLId = 0;
    _TeachersId = 0;
    _SpecializationsId = 0;
    _SpecializationsName = String.Empty;
    _Message = String.Empty;
  }
  public int Number {
    set { _Number = value; }
    get { return _Number; }
  }
  public int SpecializationsLId {
    set { _SpecializationsLId = value; }
    get { return _SpecializationsLId; }
  }
  public int TeachersId {
    set { _TeachersId = value; }
    get { return _TeachersId; }
  }
  public int SpecializationsId {
    set { _SpecializationsId = value; }
    get { return _SpecializationsId; }
```

```
  }
  public string SpecializationsName {
    set { _SpecializationsName = value; }
    get { return _SpecializationsName; }
  }
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```

Лістинг 3. Код класу «SpecializationsProvider»

```
using ScheduleGeneratorApp.AppCode;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.OleDb;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ScheduleGeneratorApp.Providers {
  class SpecializationsProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void InsertSpecializations(string SpecializationsName, string Description) {
      string SqlString = "INSERT INTO Specializations (SpecializationsName, Description" +
        ") Values(?, ?)";

      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("SpecializationsName", SpecializationsName);
          cmd.Parameters.AddWithValue("Description", Description);
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

    public List<Specializations> GetAllSpecializations() {
      int i = 0;
      string SqlString = "SELECT * FROM Specializations";

      List<Specializations> listSpecializations = new List<Specializations>();
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          conn.Open();
          using (OleDbDataReader reader = cmd.ExecuteReader()) {
            while (reader.Read()) {
```

```
            Specializations oneSpecializations = new Specializations();
            oneSpecializations.Number = ++i;
            oneSpecializations.SpecializationsId = Convert.ToInt32(reader["SpecializationsId"]);
            oneSpecializations.SpecializationsName = reader["SpecializationsName"].ToString();
            oneSpecializations.Description = reader["Description"].ToString();
            listSpecializations.Add(oneSpecializations);
          }
        }
        conn.Close();
      }
    }

    if (listSpecializations.Count == 0) {
      Specializations noSpecializations = new Specializations();
      noSpecializations.SpecializationsId = 0;
      noSpecializations.Message = NamesMy.NoDataNames.NoDataInSpecializations;
      listSpecializations.Add(noSpecializations);
    }
    return listSpecializations;
  }

  public Specializations SelectedSpecializationsBySpecializationsId(int SpecializationsId) {
    string SqlString = "SELECT * FROM Specializations Where SpecializationsId=" +
SpecializationsId.ToString();

    Specializations oneSpecializations = new Specializations();
    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        using (OleDbDataReader reader = cmd.ExecuteReader()) {
          while (reader.Read()) {
            oneSpecializations.SpecializationsId = Convert.ToInt32(reader["SpecializationsId"]);
            oneSpecializations.SpecializationsName = reader["SpecializationsName"].ToString();
            oneSpecializations.Description = reader["Description"].ToString();
          }
        }
      }
      conn.Close();
    }
    return oneSpecializations;
  }

  public void UpdateSpecializations(string SpecializationsName, string Description, int
SpecializationsId) {
    string SqlString = "UPDATE Specializations SET SpecializationsName=?, Description=?  "
+
 "WHERE SpecializationsId=?";

    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        cmd.CommandType = CommandType.Text;
        cmd.Parameters.AddWithValue("SpecializationsName", SpecializationsName);
```

```
          cmd.Parameters.AddWithValue("Description", Description);
          cmd.Parameters.AddWithValue("SpecializationsId", SpecializationsId);
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

    public void DeleteSpecializationsBySpecializationsId(int SpecializationsId) {
      string SqlString = "DELETE FROM Specializations WHERE SpecializationsId=" +
SpecializationsId.ToString();
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }


  }
}


public class Specializations {
  private int _Number;
  private int _SpecializationsId;
  private string _SpecializationsName;
  private string _Description;
  private string _Message;

  public Specializations() {
    _Number = 0;
    _SpecializationsId = 0;
    _SpecializationsName = String.Empty;
    _Description = String.Empty;
    _Message = String.Empty;
  }

  public int Number {
    set { _Number = value; }
    get { return _Number; }
  }
  public int SpecializationsId {
    set { _SpecializationsId = value; }
    get { return _SpecializationsId; }
  }
  public string SpecializationsName {
    set { _SpecializationsName = value; }
    get { return _SpecializationsName; }
```

```
  }
  public string Description {
    set { _Description = value; }
    get { return _Description; }
  }
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```

Лістинг 4. Код класу «SubjectsProvider»

```
using ScheduleGeneratorApp.AppCode;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.OleDb;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ScheduleGeneratorApp.Providers {
  class SubjectsProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];
    GroupsProvider _GroupsProvider = new GroupsProvider();
    List<Groups> _GroupsList = new List<Groups>();
    TeachersProvider _TeachersProvider = new TeachersProvider();
    List<Teachers> _TeachersList = new List<Teachers>();

    public void InsertSubjects(string SubjectsName, int TeacherId, int GroupsId) {
      string SqlString = "INSERT INTO Subjects (SubjectsName, TeacherId, GroupsId" +
        ") Values(?, ?, ?)";

      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("SubjectsName", SubjectsName);
          cmd.Parameters.AddWithValue("TeacherId", TeacherId);
          cmd.Parameters.AddWithValue("GroupsId", GroupsId);
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

    public List<Subjects> GetAllSubjects() {
      int i = 0;
      _GroupsList = _GroupsProvider.GetAllGroups();
      _TeachersList = _TeachersProvider.GetAllTeachers();
```

```
  string SqlString = "SELECT * FROM Subjects";

  List<Subjects> listSubjects = new List<Subjects>();
  using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
    using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
      conn.Open();
      using (OleDbDataReader reader = cmd.ExecuteReader()) {
        while (reader.Read()) {
          Subjects oneSubjects = new Subjects();
          oneSubjects.Number = ++i;
          oneSubjects.SubjectsId = Convert.ToInt32(reader["SubjectsId"]);
          oneSubjects.SubjectsName = reader["SubjectsName"].ToString();
          oneSubjects.TeacherId = Convert.ToInt32(reader["TeacherId"]);
          oneSubjects.GroupsId = Convert.ToInt32(reader["GroupsId"]);
          oneSubjects.Direction = GetDirection(oneSubjects.GroupsId, _GroupsList);
          oneSubjects.FIO = GetFIOName(oneSubjects.TeacherId, _TeachersList);
          listSubjects.Add(oneSubjects);
        }
      }
      conn.Close();
    }
  }

  if (listSubjects.Count == 0) {
    Subjects noSubjects = new Subjects();
    noSubjects.SubjectsId = 0;
    noSubjects.Message = NamesMy.NoDataNames.NoDataInSubjects;
    listSubjects.Add(noSubjects);
  }
  return listSubjects;
}


private string GetDirection(int GroupsId, List<Groups> GroupsList) {
  for (int i = 0; i < GroupsList.Count; i++) {
    if (GroupsId == GroupsList[i].GroupsId) {
      return GroupsList[i].Direction;
    }
  }
  return null;
}

private string GetFIOName(int TeachersId, List<Teachers> TeachersList) {
  for (int i = 0; i < TeachersList.Count; i++) {
    if (TeachersId == TeachersList[i].TeachersId) {
      return TeachersList[i].FIO;
    }
  }
  return null;
}

public Subjects SelectedSubjectsBySubjectsId(int SubjectsId) {
```

```
    string SqlString = "SELECT * FROM Subjects Where SubjectsId=" + SubjectsId.ToString();

    Subjects oneSubjects = new Subjects();
    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        using (OleDbDataReader reader = cmd.ExecuteReader()) {
          while (reader.Read()) {
            oneSubjects.SubjectsId = Convert.ToInt32(reader["SubjectsId"]);
            oneSubjects.SubjectsName = reader["SubjectsName"].ToString();
            oneSubjects.TeacherId = Convert.ToInt32(reader["TeacherId"]);
            oneSubjects.GroupsId = Convert.ToInt32(reader["GroupsId"]);
          }
        }
      }
      conn.Close();
    }
    return oneSubjects;
  }

  public void UpdateSubjects(string SubjectsName, int TeacherId, int GroupsId, int SubjectsId)
{
    string SqlString = "UPDATE Subjects SET SubjectsName=?, TeacherId=?, GroupsId=?  " +
 "WHERE SubjectsId=?";

    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        cmd.CommandType = CommandType.Text;
        cmd.Parameters.AddWithValue("SubjectsName", SubjectsName);
        cmd.Parameters.AddWithValue("TeacherId", TeacherId);
        cmd.Parameters.AddWithValue("GroupsId", GroupsId);
        cmd.Parameters.AddWithValue("SubjectsId", SubjectsId);
        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
      }
    }
  }

  public void DeleteSubjectsBySubjectsId(int SubjectsId) {
    string SqlString = "DELETE FROM Subjects WHERE SubjectsId=" +
SubjectsId.ToString();
    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
      }
    }
  }

 }
```

```
}


public class Subjects {
  private int _Number;
  private int _SubjectsId;
  private string _SubjectsName;
  private int _TeacherId;
  private int _GroupsId;
  private string _Direction;
  private string _FIO;
  private string _Message;

  public Subjects() {
    _Number = 0;
    _SubjectsId = 0;
    _SubjectsName = String.Empty;
    _TeacherId = 0;
    _GroupsId = 0;
    _Direction = String.Empty;
    _FIO = String.Empty;
    _Message = String.Empty;
  }

  public int Number {
    set { _Number = value; }
    get { return _Number; }
  }
  public int SubjectsId {
    set { _SubjectsId = value; }
    get { return _SubjectsId; }
  }
  public string SubjectsName {
    set { _SubjectsName = value; }
    get { return _SubjectsName; }
  }
  public int TeacherId {
    set { _TeacherId = value; }
    get { return _TeacherId; }
  }
  public int GroupsId {
    set { _GroupsId = value; }
    get { return _GroupsId; }
  }
  public string Direction {
    set { _Direction = value; }
    get { return _Direction; }
  }
  public string FIO {
    set { _FIO = value; }
    get { return _FIO; }
  }
```

```
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```

Лістинг 5. Код класу «ScheduleGeneratorMDI»

```
using ScheduleGeneratorApp.AppCode;
using ScheduleGeneratorApp.Forms.Controls;
using ScheduleGeneratorApp.Forms.Dictionary;
using ScheduleGeneratorApp.Forms.Systems;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ScheduleGeneratorApp {
  public partial class ScheduleGeneratorMDI : Form {

    public ScheduleGeneratorMDI() {
      InitializeComponent();
    }
    public void CloseAllWindows() {
      Form[] childArray = this.MdiChildren;
      foreach (Form childForm in childArray) {
        childForm.Close();
      }
    }

    private void викладачіToolStripMenuItem_Click(object sender, EventArgs e) {
      CloseAllWindows();
      TeachersForm clientForm = new TeachersForm();
      clientForm.MdiParent = this;
      clientForm.WindowState = FormWindowState.Maximized;
      clientForm.Show();
    }

    private void групиToolStripMenuItem_Click(object sender, EventArgs e) {
      CloseAllWindows();
      GroupsForm groupsForm = new GroupsForm();
      groupsForm.MdiParent = this;
      groupsForm.WindowState = FormWindowState.Maximized;
      groupsForm.Show();
    }

    private void спеціальностіToolStripMenuItem_Click(object sender, EventArgs e) {
```

```
  CloseAllWindows();
  SpecializationsForm specializationsForm = new SpecializationsForm();
  specializationsForm.MdiParent = this;
  specializationsForm.WindowState = FormWindowState.Maximized;
  specializationsForm.Show();
}

private void предметиToolStripMenuItem_Click(object sender, EventArgs e) {
  CloseAllWindows();
  SubjectsForm subjectsForm = new SubjectsForm();
  subjectsForm.MdiParent = this;
  subjectsForm.WindowState = FormWindowState.Maximized;
  subjectsForm.Show();
}

private void аудиторіїToolStripMenuItem_Click(object sender, EventArgs e) {
  CloseAllWindows();
  ClassroomsForm classroomsForm = new ClassroomsForm();
  classroomsForm.MdiParent = this;
  classroomsForm.WindowState = FormWindowState.Maximized;
  classroomsForm.Show();
}

private void розкладToolStripMenuItem_Click(object sender, EventArgs e) {
  CloseAllWindows();
  SchedulesForm schedulesForm = new SchedulesForm();
  schedulesForm.MdiParent = this;
  schedulesForm.WindowState = FormWindowState.Maximized;
  schedulesForm.Show();
}

private void вихідToolStripMenuItem_Click(object sender, EventArgs e) {
  this.Close();
}

private void користувачіToolStripMenuItem_Click(object sender, EventArgs e) {
  if (LoginForm.CurrentUser.RoleId == 1) {
    CloseAllWindows();
    UsersForm usersForm = new UsersForm();
    usersForm.MdiParent = this;
    usersForm.WindowState = FormWindowState.Maximized;
    usersForm.Show();
  } else {
    MessageBox.Show(NamesMy.MessageBoxExaption.YouDontHavePermission);
  }
}

private void подіїToolStripMenuItem_Click(object sender, EventArgs e) {
  if (LoginForm.CurrentUser.RoleId == 1) {
    CloseAllWindows();
    SystemLogForm systemLogForm = new SystemLogForm();
    systemLogForm.MdiParent = this;
```

```
        systemLogForm.WindowState = FormWindowState.Maximized;
        systemLogForm.Show();
      } else {
        MessageBox.Show(NamesMy.MessageBoxExaption.YouDontHavePermission);
      }
    }

    private void персоналізаціяToolStripMenuItem_Click(object sender, EventArgs e) {
      PersonalizationForm personalizationForm = new
PersonalizationForm(LoginForm.CurrentUser.UsersId);
      personalizationForm.MdiParent = this;
      personalizationForm.WindowState = FormWindowState.Maximized;
      personalizationForm.Show();
    }

    private void ScheduleGeneratorMDI_Resize(object sender, EventArgs e) {
      this.BackgroundImage = Properties.Resources.back;
    }
  }
}
```

Лістинг 6. Код класу «SchedulesForm»

```
using ScheduleGeneratorApp.AppCode;
using ScheduleGeneratorApp.Providers;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ScheduleGeneratorApp.Forms.Controls {
  public partial class SchedulesForm : Form {
    private ClassroomsProvider _ClassroomsProvider = new ClassroomsProvider();
    private GroupsProvider _GroupsProvider = new GroupsProvider();
    private TeachersProvider _TeachersProvider = new TeachersProvider();
    private SubjectsProvider _SubjectsProvider = new SubjectsProvider();
    private List<Classrooms> _ClassroomsList = new List<Classrooms>();
    private List<Groups> _GroupsList = new List<Groups>();
    private List<Teachers> _TeachersList = new List<Teachers>();
    private List<Subjects> _SubjectsList = new List<Subjects>();
    private ValidationMy _validation = new ValidationMy();


    public SchedulesForm() {
      InitializeComponent();
      _GroupsList = _GroupsProvider.GetAllGroups();
```

```
            _TeachersList = _TeachersProvider.GetAllTeachers();
            _SubjectsList = _SubjectsProvider.GetAllSubjects();
            _ClassroomsList = _ClassroomsProvider.GetAllClassrooms();
        }


        private void CreateShedule() {
            // Set the lesson duration and number of lessons per week
            int lessonDuration = Convert.ToInt32(LessonDurationTBox.Text); // minutes
            int lessonsPerWeek =  Convert.ToInt32(LessonsPerWeekTBox.Text); //відображає кількість занять, які повинні відбуватись протягом одного тижня
                                                                            //для кожної групи. Ця змінна використовується для задання обмежень щодо розкладу занять
                                                                            //та для створення випадкових розкладів під час генерації початкової популяції в генетичному алгоритмі.

            // Create the Genetic Algorithm instance
            GeneticAlgorithm ga = new GeneticAlgorithm(_GroupsList, _TeachersList, _SubjectsList, _ClassroomsList,
              lessonDuration, lessonsPerWeek);
            // Generate the optimal timetable
            Schedule optimalSchedule = ga.GenerateTimetable();
            string raport = "";
            // Output the optimal schedule
            Console.WriteLine("Оптимальний розклад:");
            RaportTBox.Text += String.Format("{0,-10}|{1, -25}|{2, -30}|{3, -15}|{4, -6}|{5, -4}|{6, -6}|\r\n",
              "Група", "Предмет", "Викладач", "Аудиторія", "Вміст.", "День", "Пара");
            foreach (var lesson in optimalSchedule.Lessons) {
              Groups group = _GroupsList.FirstOrDefault(g => g.GroupsId == lesson.GroupId);
              Subjects subject = _SubjectsList.FirstOrDefault(s => s.SubjectsId == lesson.SubjectId);
              Teachers teacher = _TeachersList.FirstOrDefault(t => t.TeachersId == lesson.TeacherId);
              Classrooms classroom = _ClassroomsList.FirstOrDefault(c => c.ClassroomsId == lesson.ClassroomId);

              raport += String.Format("{0,-10}|{1, -25}|{2, -30}|{3, 15}|{4, 6}|{5, 4}|{6, 6}|\r\n", group.Direction, subject.SubjectsName,
                teacher.FIO, classroom.ClassroomsName, classroom.Capacity, lesson.DayOfWeek, lesson.Period);
            }
            RaportTBox.Text += raport;
        }

        private void FormingBtn_Click(object sender, EventArgs e) {
            if (IsDataEnteringCorrect()) {
              CreateShedule();
```

```
        }
      }


      private bool IsDataEnteringCorrect() {
        bool isCorrect = true;
        if (_validation.IsDataConvertToInt(LessonDurationTBox.Text)) {
          LessonDurationValiadtionLbl.Text =
NamesMy.ProgramButtons.RequiredValidation;
        } else {
          LessonDurationValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
          isCorrect = false;
        }
        if (_validation.IsDataConvertToInt(LessonsPerWeekTBox.Text)) {
          LessonsPerWeekValiadtionLbl.Text =
NamesMy.ProgramButtons.RequiredValidation;
        } else {
          LessonsPerWeekValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
          isCorrect = false;
        }

        return isCorrect;
      }

    }
  }

  public class Lesson {
    public int GroupId { get; set; }
    public int SubjectId { get; set; }
    public int TeacherId { get; set; }
    public int ClassroomId { get; set; }
    public int DayOfWeek { get; set; }
    public int Period { get; set; }
  }

  public class Schedule {
    public List<Lesson> Lessons { get; set; }
    public double Fitness { get; set; }

    public Schedule() {
      Lessons = new List<Lesson>();
    }
  }

  public class GeneticAlgorithm {
    private List<Groups> _groups;
    private List<Teachers> _teachers;
```

```
private List<Subjects> _subjects;
private List<Classrooms> _classrooms;
private int _lessonsPerWeek;

private int _populationSize = 100;
private int _numberOfGenerations = 1000;

public GeneticAlgorithm(
    List<Groups> groups,
    List<Teachers> teachers,
    List<Subjects> subjects,
    List<Classrooms> classrooms,
    int lessonDuration,
    int lessonsPerWeek) {
  _groups = groups;
  _teachers = teachers;
  _subjects = subjects;
  _classrooms = classrooms;
  _lessonsPerWeek = lessonsPerWeek;
}

public Schedule GenerateTimetable() {
  // Initialize population
  List<Schedule> population = new List<Schedule>();
  for (int i = 0; i < _populationSize; i++) {
    population.Add(GenerateRandomSchedule());
  }

  for (int generation = 0; generation < _numberOfGenerations; generation++) {
    // Calculate fitness
    foreach (var schedule in population) {
      schedule.Fitness = CalculateFitness(schedule);
    }

    // Sort by fitness
    population = population.OrderByDescending(s => s.Fitness).ToList();

    // Apply selection
    List<Schedule> newPopulation = new List<Schedule>();
    int eliteSize = (int)(_populationSize * 0.1);
    for (int i = 0; i < eliteSize; i++) {
      newPopulation.Add(population[i]);
    }

    // Apply crossover and mutation
    for (int i = eliteSize; i < _populationSize; i++) {
      Schedule parent1 = SelectParent(population);
      Schedule parent2 = SelectParent(population);
```

```
            Schedule offspring = Crossover(parent1, parent2);
            Mutate(offspring);
            newPopulation.Add(offspring);
          }

          population = newPopulation;

          // Check for early stopping condition
          if (population[0].Fitness == 1.0) {
            break;
          }
        }

        return population[0];
      }

      private Schedule GenerateRandomSchedule() {
        Schedule schedule = new Schedule();
        Random random = new Random();

        foreach (var group in _groups) {
          for (int i = 0; i < _lessonsPerWeek; i++) {
            foreach (var subject in _subjects.Where(s => s.GroupsId == group.GroupsId)) {
              Lesson lesson = new Lesson();
              lesson.GroupId = group.GroupsId;
              lesson.SubjectId = subject.SubjectsId;
              lesson.TeacherId = subject.TeacherId;

              // Assign a random classroom that can accommodate the group
              var suitableClassrooms = _classrooms.Where(c => c.Capacity >=
group.StudentCount).ToList();
              lesson.ClassroomId =
suitableClassrooms[random.Next(suitableClassrooms.Count)].ClassroomsId;

              // Assign a random day of the week (1 to 5 for Monday to Friday)
              lesson.DayOfWeek = random.Next(1, 6);

              // Assign a random period during the day
              // Assuming there are 8 periods per day (you can adjust this as needed)
              lesson.Period = random.Next(1, 9);

              schedule.Lessons.Add(lesson);
            }
          }
        }

        return schedule;
      }
```

```
        //Цей метод обчислює приспособленість розкладу, штраф
        private double CalculateFitness(Schedule schedule) {
          double fitness = 1.0;
          int totalConstraints = 3;

          foreach (var lesson in schedule.Lessons) {
            // Constraint 1: Check if the group has any other lessons at the same time
            if (schedule.Lessons.Any(l => l.GroupId == lesson.GroupId && l.DayOfWeek ==
lesson.DayOfWeek && l.Period == lesson.Period && l.SubjectId != lesson.SubjectId)) {
              fitness -= 1.0 / totalConstraints;
              continue;
            }

            // Constraint 2: Check if the teacher has any other lessons at the same time
            if (schedule.Lessons.Any(l => l.TeacherId == lesson.TeacherId && l.DayOfWeek ==
lesson.DayOfWeek && l.Period == lesson.Period && l.SubjectId != lesson.SubjectId)) {
              fitness -= 1.0 / totalConstraints;
              continue;
            }

            // Constraint 3: Check if the classroom can accommodate the group
            Classrooms classroom = _classrooms.FirstOrDefault(c => c.ClassroomsId ==
lesson.ClassroomId);
            Groups group = _groups.FirstOrDefault(g => g.GroupsId == lesson.GroupId);
            if (classroom.Capacity < group.StudentCount) {
              fitness -= 1.0 / totalConstraints;
            }
          }

          return fitness;
        }


        private Schedule SelectParent(List<Schedule> population) {
          int tournamentSize = 4; // You can adjust the tournament size as needed
          Random random = new Random();

          // Select random schedules for the tournament
          List<Schedule> tournamentSchedules = new List<Schedule>();
          for (int i = 0; i < tournamentSize; i++) {
            int randomIndex = random.Next(population.Count);
            tournamentSchedules.Add(population[randomIndex]);
          }

          // Find the schedule with the highest fitness in the tournament
          Schedule bestSchedule = tournamentSchedules[0];
          double bestFitness = CalculateFitness(bestSchedule);
```

```
  for (int i = 1; i < tournamentSchedules.Count; i++) {
    double currentFitness = CalculateFitness(tournamentSchedules[i]);
    if (currentFitness > bestFitness) {
      bestSchedule = tournamentSchedules[i];
      bestFitness = currentFitness;
    }
  }

  return bestSchedule;
}

private Schedule Crossover(Schedule parent1, Schedule parent2) {
  Schedule offspring = new Schedule();
  Random random = new Random();

  // Choose a random crossover point
  int crossoverPoint = random.Next(parent1.Lessons.Count);

  // Copy the lessons from parent1 up to the crossover point
  for (int i = 0; i < crossoverPoint; i++) {
    offspring.Lessons.Add(parent1.Lessons[i]);
  }

  // Copy the remaining lessons from parent2
  for (int i = crossoverPoint; i < parent2.Lessons.Count; i++) {
    offspring.Lessons.Add(parent2.Lessons[i]);
  }

  return offspring;
}

private void Mutate(Schedule schedule) {
  double mutationRate = 0.1; // You can adjust the mutation rate as needed
  Random random = new Random();

  // Iterate through the lessons in the schedule
  for (int i = 0; i < schedule.Lessons.Count; i++) {
    // Apply mutation with the given mutation rate
    if (random.NextDouble() < mutationRate) {
      // Choose a random lesson to swap with
      int randomIndex = random.Next(schedule.Lessons.Count);

      // Swap the lessons
      Lesson temp = schedule.Lessons[i];
      schedule.Lessons[i] = schedule.Lessons[randomIndex];
      schedule.Lessons[randomIndex] = temp;
    }
  }
```

```
        }

      }
```
Лістинг 7. Код класу «ClassroomsForm»
```
using ScheduleGeneratorApp.AppCode;
using ScheduleGeneratorApp.Providers;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ScheduleGeneratorApp.Forms.Dictionary {
  public partial class ClassroomsForm : Form {
    private int _selectedRowIndex = 0;
    private ValidationMy _validation = new ValidationMy();
    private ClassroomsProvider _ClassroomsProvider = new ClassroomsProvider();
    private List<Classrooms> _ClassroomsList = new List<Classrooms>();

    public ClassroomsForm() {
      InitializeComponent();
      DataLoad();
    }

    private void AddBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _ClassroomsProvider.InsertClassrooms(ClassroomsNameTBox.Text,
Convert.ToInt32(CapacityTBox.Text));
        DataLoad();
        ClearAllControls();
      }
    }

    private void ClearBtn_Click(object sender, EventArgs e) {
      ClearAllControls();
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }


    private void DataLoad() {
      int firstRowIndex = 0;
      if (ClassroomsGridView.FirstDisplayedScrollingRowIndex > 0) {
        firstRowIndex = ClassroomsGridView.FirstDisplayedScrollingRowIndex;
      }
      try {
```

```
        _ClassroomsList = _ClassroomsProvider.GetAllClassrooms();
        LoadDataInClassroomsGridView(_ClassroomsList);
        if (_selectedRowIndex == ClassroomsGridView.Rows.Count) {
          _selectedRowIndex = ClassroomsGridView.Rows.Count - 1;
        }
        if (_selectedRowIndex >= 0) {
          ClassroomsGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
          ClassroomsGridView.Rows[_selectedRowIndex].Selected = true;
        }
      } catch { }
    }

    private void LoadDataInClassroomsGridView(List<Classrooms> ClassroomsList) {
      ClassroomsGridView.DataSource = null;
      ClassroomsGridView.Columns.Clear();
      ClassroomsGridView.AutoGenerateColumns = false;
      ClassroomsGridView.RowHeadersVisible = false;

      ClassroomsGridView.DataSource = ClassroomsList;

      if (ClassroomsList.Count > 0) {
        if (ClassroomsList[0].Message == NamesMy.NoDataNames.NoDataInClassrooms) {
          DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
          messageColumn.DataPropertyName = "Message";
          messageColumn.Width = ClassroomsGridView.Width -
NamesMy.SizeOptins.MinusSizePanel;
          ClassroomsGridView.Columns.Add(messageColumn);
        } else {
          DataGridViewColumn DetailIdColumn = new DataGridViewTextBoxColumn();
          DetailIdColumn.DataPropertyName = "ClassroomsId";
          ClassroomsGridView.Columns.Add(DetailIdColumn);
          ClassroomsGridView.Columns[0].Visible = false;

          DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
          numberColumn.HeaderText = "№ ";
          numberColumn.DataPropertyName = "Number";
          numberColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
          numberColumn.Width = NamesMy.SizeOptins.NumberSize;
          ClassroomsGridView.Columns.Add(numberColumn);

          DataGridViewColumn ClassroomsNameColumn = new DataGridViewTextBoxColumn();
          ClassroomsNameColumn.HeaderText = "Аудиторія";
          ClassroomsNameColumn.DataPropertyName = "ClassroomsName";
          ClassroomsNameColumn.Width = NamesMy.SizeOptins.NameSize;
          ClassroomsGridView.Columns.Add(ClassroomsNameColumn);

          DataGridViewColumn CapacityCount = new DataGridViewTextBoxColumn();
          CapacityCount.HeaderText = "Вмістимість";
          CapacityCount.DataPropertyName = "Capacity";
          CapacityCount.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
```

```
        CapacityCount.Width = 150;
        ClassroomsGridView.Columns.Add(CapacityCount);

      }
      for (int i = 0; i < ClassroomsGridView.Columns.Count; i++) {
        ClassroomsGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
      }
    }
  }

  private void ClearAllControls() {
    ClassroomsNameTBox.Text = String.Empty;
    CapacityTBox.Text = "0";
  }

  private bool IsDataEnteringCorrect() {
    bool isCorrect = true;
    if (_validation.IsDataEntering(ClassroomsNameTBox.Text)) {
      ClassroomsNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      ClassroomsNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_validation.IsDataConvertToInt(CapacityTBox.Text)) {
      CapacityValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      CapacityValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    return isCorrect;
  }

  private void ClassroomsGridView_CellClick(object sender, DataGridViewCellEventArgs e) {
    if (e.RowIndex >= 0 && ClassroomsGridView[0, e.RowIndex].Value.ToString() !=
_ClassroomsList[0].Message) {
      _selectedRowIndex = e.RowIndex;
      UpdateClassroomsForm updateClassroomsForm = new
UpdateClassroomsForm(Convert.ToInt32(ClassroomsGridView[0,
e.RowIndex].Value.ToString()));
      updateClassroomsForm.ShowDialog();
      DataLoad();
    }
  }
 }
}
```

Лістинг 8. Код класу «SpecializationsForm»

```
using ScheduleGeneratorApp.AppCode;
using ScheduleGeneratorApp.Providers;
using System;
using System.Collections.Generic;
using System.ComponentModel;
```

```
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ScheduleGeneratorApp.Forms.Dictionary {
  public partial class SpecializationsForm : Form {
    private int _selectedRowIndex = 0;
    private ValidationMy _validation = new ValidationMy();
    private SpecializationsProvider _SpecializationsProvider = new SpecializationsProvider();
    private List<Specializations> _SpecializationsList = new List<Specializations>();

    public SpecializationsForm() {
      InitializeComponent();
      DataLoad();
    }

    private void AddBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _SpecializationsProvider.InsertSpecializations(SpecializationsNameTBox.Text,
DescriptionTBox.Text);
        DataLoad();
        ClearAllControls();
      }
    }

    private void ClearBtn_Click(object sender, EventArgs e) {
      ClearAllControls();
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }

    private void DataLoad() {
      int firstRowIndex = 0;
      if (SpecializationsGridView.FirstDisplayedScrollingRowIndex > 0) {
        firstRowIndex = SpecializationsGridView.FirstDisplayedScrollingRowIndex;
      }
      try {
        _SpecializationsList = _SpecializationsProvider.GetAllSpecializations();
        LoadDataInSpecializationsGridView(_SpecializationsList);
        if (_selectedRowIndex == SpecializationsGridView.Rows.Count) {
          _selectedRowIndex = SpecializationsGridView.Rows.Count - 1;
        }
        if (_selectedRowIndex >= 0) {
          SpecializationsGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
          SpecializationsGridView.Rows[_selectedRowIndex].Selected = true;
        }
      } catch { }
```

```
    }

    private void LoadDataInSpecializationsGridView(List<Specializations> SpecializationsList) {
      SpecializationsGridView.DataSource = null;
      SpecializationsGridView.Columns.Clear();
      SpecializationsGridView.AutoGenerateColumns = false;
      SpecializationsGridView.RowHeadersVisible = false;

      SpecializationsGridView.DataSource = SpecializationsList;

      if (SpecializationsList.Count > 0) {
        if (SpecializationsList[0].Message == NamesMy.NoDataNames.NoDataInSpecializations) {
          DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
          messageColumn.DataPropertyName = "Message";
          messageColumn.Width = SpecializationsGridView.Width -
NamesMy.SizeOptins.MinusSizePanel;
          SpecializationsGridView.Columns.Add(messageColumn);
        } else {
          DataGridViewColumn DetailIdColumn = new DataGridViewTextBoxColumn();
          DetailIdColumn.DataPropertyName = "SpecializationsId";
          SpecializationsGridView.Columns.Add(DetailIdColumn);
          SpecializationsGridView.Columns[0].Visible = false;

          DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
          numberColumn.HeaderText = "№ ";
          numberColumn.DataPropertyName = "Number";
          numberColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
          numberColumn.Width = NamesMy.SizeOptins.NumberSize;
          SpecializationsGridView.Columns.Add(numberColumn);

          DataGridViewColumn SpecializationsNameColumn = new
DataGridViewTextBoxColumn();
          SpecializationsNameColumn.HeaderText = "Спеціальність";
          SpecializationsNameColumn.DataPropertyName = "SpecializationsName";
          SpecializationsNameColumn.Width = NamesMy.SizeOptins.NameSize;
          SpecializationsGridView.Columns.Add(SpecializationsNameColumn);

        }
        for (int i = 0; i < SpecializationsGridView.Columns.Count; i++) {
          SpecializationsGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
        }
      }
    }

    private void ClearAllControls() {
      SpecializationsNameTBox.Text = String.Empty;
      DescriptionTBox.Text = String.Empty;
    }

    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
```

```
      if (_validation.IsDataEntering(SpecializationsNameTBox.Text)) {
        SpecializationsNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        SpecializationsNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
    }

    private void SpecializationsGridView_CellClick(object sender, DataGridViewCellEventArgs
e) {
      if (e.RowIndex >= 0 && SpecializationsGridView[0, e.RowIndex].Value.ToString() !=
_SpecializationsList[0].Message) {
        _selectedRowIndex = e.RowIndex;
        UpdateSpecializationsForm updateSpecializationsForm = new
UpdateSpecializationsForm(Convert.ToInt32(SpecializationsGridView[0,
e.RowIndex].Value.ToString()));
        updateSpecializationsForm.ShowDialog();
        DataLoad();
      }
    }
  }
}
```

Лістинг 9. Код класу «SubjectsForm»

```
using ScheduleGeneratorApp.AppCode;
using ScheduleGeneratorApp.Providers;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ScheduleGeneratorApp.Forms.Dictionary {
  public partial class SubjectsForm : Form {
    private int _selectedRowIndex = 0;
    private ValidationMy _validation = new ValidationMy();
    private SubjectsProvider _SubjectsPrivider = new SubjectsProvider();
    private List<Subjects> _SubjectsList = new List<Subjects>();
    private GroupsProvider _GroupsProvider = new GroupsProvider();
    private List<Groups> _GroupsList = new List<Groups>();
    private TeachersProvider _TeachersProvider = new TeachersProvider();
    private List<Teachers> _TeachersList = new List<Teachers>();


    public SubjectsForm() {
      InitializeComponent();
      LoadAllDate();
```

```
    DataLoad();
  }

  private void AddBtn_Click(object sender, EventArgs e) {
    if (IsDataEnteringCorrect()) {
      _SubjectsPrivider.InsertSubjects(SubjectsNameTBox.Text,
Convert.ToInt32(TeacherCBox.SelectedValue), Convert.ToInt32(GroupsCBox.SelectedValue));
      DataLoad();
      ClearAllControls();
    }
  }

  private void ClearBtn_Click(object sender, EventArgs e) {
    ClearAllControls();
  }

  private void ExitBtn_Click(object sender, EventArgs e) {
    this.Close();
  }


  private void LoadAllDate() {
    _GroupsList = _GroupsProvider.GetAllGroups();
    GroupsCBox.DataSource = _GroupsList;
    GroupsCBox.ValueMember = "GroupsId";
    GroupsCBox.DisplayMember = "Direction";

    _TeachersList = _TeachersProvider.GetAllTeachers();
    TeacherCBox.DataSource = _TeachersList;
    TeacherCBox.ValueMember = "TeachersId";
    TeacherCBox.DisplayMember = "FIO";
  }

  private void DataLoad() {
    int firstRowIndex = 0;
    if (SubjectsGridView.FirstDisplayedScrollingRowIndex > 0) {
      firstRowIndex = SubjectsGridView.FirstDisplayedScrollingRowIndex;
    }
    try {
      _SubjectsList = _SubjectsPrivider.GetAllSubjects();
      LoadDataInSubjectsGridView(_SubjectsList);
      if (_selectedRowIndex == SubjectsGridView.Rows.Count) {
        _selectedRowIndex = SubjectsGridView.Rows.Count - 1;
      }
      if (_selectedRowIndex >= 0) {
        SubjectsGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
        SubjectsGridView.Rows[_selectedRowIndex].Selected = true;
      }
    } catch { }
  }

  private void LoadDataInSubjectsGridView(List<Subjects> SubjectsList) {
```

```
    SubjectsGridView.DataSource = null;
    SubjectsGridView.Columns.Clear();
    SubjectsGridView.AutoGenerateColumns = false;
    SubjectsGridView.RowHeadersVisible = false;

    SubjectsGridView.DataSource = SubjectsList;

    if (SubjectsList.Count > 0) {
      if (SubjectsList[0].Message == NamesMy.NoDataNames.NoDataInSubjects) {
        DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
        messageColumn.DataPropertyName = "Message";
        messageColumn.Width = SubjectsGridView.Width -
NamesMy.SizeOptins.MinusSizePanel;
        SubjectsGridView.Columns.Add(messageColumn);
      } else {
        DataGridViewColumn DetailIdColumn = new DataGridViewTextBoxColumn();
        DetailIdColumn.DataPropertyName = "SubjectsId";
        SubjectsGridView.Columns.Add(DetailIdColumn);
        SubjectsGridView.Columns[0].Visible = false;

        DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
        numberColumn.HeaderText = "№ ";
        numberColumn.DataPropertyName = "Number";
        numberColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
        numberColumn.Width = NamesMy.SizeOptins.NumberSize;
        SubjectsGridView.Columns.Add(numberColumn);

        DataGridViewColumn SubjectsNameColumn = new DataGridViewTextBoxColumn();
        SubjectsNameColumn.HeaderText = "Предмети";
        SubjectsNameColumn.DataPropertyName = "SubjectsName";
        SubjectsNameColumn.Width = NamesMy.SizeOptins.NameSize;
        SubjectsGridView.Columns.Add(SubjectsNameColumn);

        DataGridViewColumn DirectionColumn = new DataGridViewTextBoxColumn();
        DirectionColumn.HeaderText = "Група";
        DirectionColumn.DataPropertyName = "Direction";
        DirectionColumn.Width = 150;
        SubjectsGridView.Columns.Add(DirectionColumn);


        DataGridViewColumn FIOColumn = new DataGridViewTextBoxColumn();
        FIOColumn.HeaderText = "Викладач";
        FIOColumn.DataPropertyName = "FIO";
        FIOColumn.Width = 180;
        SubjectsGridView.Columns.Add(FIOColumn);
      }
      for (int i = 0; i < SubjectsGridView.Columns.Count; i++) {
        SubjectsGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
      }
    }
  }
```

```
    private void ClearAllControls() {
      SubjectsNameTBox.Text = String.Empty;
    }

    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
      if (_validation.IsDataEntering(SubjectsNameTBox.Text)) {
        SubjectsNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        SubjectsNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      if (Convert.ToInt32(TeacherCBox.SelectedValue) > 0) {
        TeacherValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        TeacherValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      if (Convert.ToInt32(GroupsCBox.SelectedValue) > 0) {
        GroupValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        GroupValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
    }

    private void SubjectsGridView_CellClick(object sender, DataGridViewCellEventArgs e) {
      if (e.RowIndex >= 0 && SubjectsGridView[0, e.RowIndex].Value.ToString() !=
_SubjectsList[0].Message) {
        _selectedRowIndex = e.RowIndex;
        UpdateSubjectsForm updateSubjectsForm = new
UpdateSubjectsForm(Convert.ToInt32(SubjectsGridView[0, e.RowIndex].Value.ToString()));
        updateSubjectsForm.ShowDialog();
        DataLoad();
      }
    }
  }
}
```

Лістинг 10. Код класу «TeachersForm»

```
using ScheduleGeneratorApp.AppCode;
using ScheduleGeneratorApp.Providers;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
using System.Windows.Forms;

namespace ScheduleGeneratorApp.Forms.Dictionary {
  public partial class TeachersForm : Form {
    private int _selectedRowIndex = 0;
    private ValidationMy _validation = new ValidationMy();
    TeachersProvider _TeachersProvider = new TeachersProvider();
    List<Teachers> _TeachersList = new List<Teachers>();

    public TeachersForm() {
      InitializeComponent();
      DataLoad();
    }

    private void AddBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _TeachersProvider.InsertTeachers(LastNameTBox.Text, FirstNameTBox.Text,
PhoneTBox.Text, AddressTBox.Text, EmailTBox.Text,
          Convert.ToInt32(WorkloadTBox.Text));
        DataLoad();
        ClearAllControls();
      }
    }

    private void ClearBtn_Click(object sender, EventArgs e) {
      ClearAllControls();
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }


    private void DataLoad() {
      int firstRowIndex = 0;
      if (TeachersGridView.FirstDisplayedScrollingRowIndex > 0) {
        firstRowIndex = TeachersGridView.FirstDisplayedScrollingRowIndex;
      }
      try {
        _TeachersList = _TeachersProvider.GetAllTeachers();
        LoadDataInTeachersGridView(_TeachersList);
        if (_selectedRowIndex == TeachersGridView.Rows.Count) {
          _selectedRowIndex = TeachersGridView.Rows.Count - 1;
        }
        if (_selectedRowIndex >= 0) {
          TeachersGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
          TeachersGridView.Rows[_selectedRowIndex].Selected = true;
        }
      } catch { }
    }

    private void LoadDataInTeachersGridView(List<Teachers> TeachersList) {
```

```
    TeachersGridView.DataSource = null;
    TeachersGridView.Columns.Clear();
    TeachersGridView.AutoGenerateColumns = false;
    TeachersGridView.RowHeadersVisible = false;

    TeachersGridView.DataSource = TeachersList;

    if (TeachersList.Count > 0) {
      if (TeachersList[0].Message == NamesMy.NoDataNames.NoDataInTeachers) {
        DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
        messageColumn.DataPropertyName = "Message";
        messageColumn.Width = TeachersGridView.Width -
NamesMy.SizeOptins.MinusSizePanel;
        TeachersGridView.Columns.Add(messageColumn);
      } else {
        DataGridViewColumn DetailIdColumn = new DataGridViewTextBoxColumn();
        DetailIdColumn.DataPropertyName = "TeachersId";
        TeachersGridView.Columns.Add(DetailIdColumn);
        TeachersGridView.Columns[0].Visible = false;

        DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
        numberColumn.HeaderText = "№ ";
        numberColumn.DataPropertyName = "Number";
        numberColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
        numberColumn.Width = NamesMy.SizeOptins.NumberSize;
        TeachersGridView.Columns.Add(numberColumn);

        DataGridViewColumn LastNameColumn = new DataGridViewTextBoxColumn();
        LastNameColumn.HeaderText = "Прізвище";
        LastNameColumn.DataPropertyName = "LastName";
        LastNameColumn.Width = 200;
        TeachersGridView.Columns.Add(LastNameColumn);

        DataGridViewColumn FirstNameColumn = new DataGridViewTextBoxColumn();
        FirstNameColumn.HeaderText = "Ім'я";
        FirstNameColumn.DataPropertyName = "FirstName";
        FirstNameColumn.Width = 200;
        TeachersGridView.Columns.Add(FirstNameColumn);

        DataGridViewColumn PhoneColumn = new DataGridViewTextBoxColumn();
        PhoneColumn.HeaderText = "№ телефону";
        PhoneColumn.DataPropertyName = "Phone";
        PhoneColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
        PhoneColumn.Width = 150;
        TeachersGridView.Columns.Add(PhoneColumn);

      }
      for (int i = 0; i < TeachersGridView.Columns.Count; i++) {
        TeachersGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
      }
```

```
    }
   }

   private void ClearAllControls() {
    LastNameTBox.Text = String.Empty;
    FirstNameTBox.Text = String.Empty;
    PhoneTBox.Text = String.Empty;
   }

   private bool IsDataEnteringCorrect() {
    bool isCorrect = true;
    if (_validation.IsDataEntering(LastNameTBox.Text)) {
     LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
     LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
     isCorrect = false;
    }
    if (_validation.IsDataEntering(FirstNameTBox.Text)) {
     FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
     FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
     isCorrect = false;
    }
    if (_validation.IsDataEntering(PhoneTBox.Text)) {
     PhoneValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
     PhoneValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
     isCorrect = false;
    }
    if (_validation.IsDataConvertToInt(WorkloadTBox.Text)) {
     WorkloadValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
     WorkloadValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
     isCorrect = false;
    }
    return isCorrect;
   }

   private void TeachersGridView_CellClick(object sender, DataGridViewCellEventArgs e) {
    if (e.RowIndex >= 0 && TeachersGridView[0, e.RowIndex].Value.ToString() !=
_TeachersList[0].Message) {
     _selectedRowIndex = e.RowIndex;
     UpdateTeachersForm updateTeachersForm = new
UpdateTeachersForm(Convert.ToInt32(TeachersGridView[0, e.RowIndex].Value.ToString()));
     updateTeachersForm.ShowDialog();
     DataLoad();
    }
   }
  }
}
```