ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА
(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерних систем та технологій
(повна назва кафедри)

Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр»
(освітньо-кваліфікаційний рівень)

«Розробка інформаційно-довідкової системи навчальних дисциплін з вибіркової компоненти освітньої програми»

«Development of an information and reference system of educational disciplines from the selective component of the educational program»

Виконав: здобувач денної форми навчання
спеціальності 122 Комп'ютерні науки
(код, назва спеціальності)
Освітня програма Комп'ютерні науки
(назва)
Сєвєрін Станіслав Максимович
(прізвище, ім'я, по-батькові здобувача)

Керівник ст. викладач Шаріпова І.В. _____
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент канд. ф-м. н., доц. Шугайло Ю.Б. _____
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рекомендовано до захисту:
Протокол засідання кафедри
№ ___ від «___»_____2024 р.

Завідувач кафедри

_____ Юрій ГУНЧЕНКО
(підпис) (прізвище, ім'я)

Захищено на засіданні ЕК № _____
Протокол №___від «___»_____2024 р.
Оцінка_____/_____/_____
(за національною шкалою / шкалою ECTS / бали)

Голова ЕК

_____ Микола МАЛАКСІАНО
(підпис) (прізвище, ім'я)

Одеса - 2024

АНОТАЦІЯ

Кваліфікаційна робота розглядає інформаційно-довідкову систему для забезпечення доступу до змісту навчальних дисциплін вибіркової компоненти освітньої програми та орієнтована на покращення процесу навчання шляхом забезпечення здобувачів вищої освіти та викладачів необхідною інформацією про цільові дисципліни, програмні матеріали, методичні рекомендації та оцінювальні критерії. З урахуванням потреб користувачів та сучасних вимог до інформаційно-комунікаційних засобів у сфері освіти у роботі досліджуються методи та технології розробки інформаційно-довідкової системи. Результатом дослідження є створення прототипу системи, яка рекомендується для подальшого впровадження в освітній процес Одеського національного університету імені І.І. Мечникова.

Мета кваліфікаційної роботи полягає в розробці та реалізації інформаційно-довідкової системи, спрямованої на полегшення доступу до змісту навчальних дисциплін вибіркової компоненти освітньої програми. Основними цілями є створення зручного та ефективного інструменту для здобувачів вищої освіти та викладачів, який надасть можливість швидко отримувати інформацію про обрані дисципліни, навчальні плани, методичні матеріали та інші важливі аспекти. Досліджуються сучасні методи та технології для забезпечення зручного доступу до інформації, а також враховуються потреби та вимоги користувачів у сфері освіти. Результатом роботи є розроблений прототип інформаційної системи, який може бути використаний для подальшого впровадження в освітній процес з метою підвищення ефективності навчання та сприяння підвищенню якості освіти.

У результаті проведених у роботі досліджень було розроблено інформаційно-довідкову систему. Ця система включає в себе інтерфейс для швидкого доступу до інформації про навчальні плани, методичні матеріали, рекомендації та оцінювальні критерії обраних дисциплін. Крім того, вона передбачає можливість взаємодії між здобувачами вищої освіти та

викладачами через обмін повідомленнями та обговорення навчальних питань. Розроблений прототип системи відповідає сучасним вимогам до інформаційно-комунікаційних засобів у сфері освіти та може бути використаний для підвищення якості освітнього процесу.

Ключові слова: інформаційно-довідкова система (ІДС), архітектура системи, функціональні можливості, тестування та валідація, вибіркова компонента освітньої програми, взаємодія викладачів і здобувачів вищої освіти.

ANNOTATION

The qualification work examines an information and reference system for providing access to the content of elective courses in the educational program, aimed at improving the learning process by providing higher education students and teachers with the necessary information about target courses, program materials, methodological recommendations, and evaluation criteria. Taking into account the users' needs and modern requirements for information and communication tools in education, the work explores methods and technologies for developing the information and reference system. The result of the research is the creation of a system prototype recommended for further implementation in the educational process of the I.I. Mechnikov Odessa National University.

The aim of the qualification work is to develop and implement an information and reference system designed to facilitate access to the content of elective courses in the educational program. The main objectives are to create a convenient and efficient tool for higher education students and teachers, providing the ability to quickly obtain information about selected courses, syllabi, methodological materials, and other important aspects. The study investigates modern methods and technologies to ensure convenient access to information, while also considering the needs and requirements of users in the field of education. The result of the work is a developed prototype of the information system, which can be used for further implementation in the educational process to enhance learning efficiency and contribute to the improvement of education quality.

As a result of the research conducted in the work, an information and reference system was developed. This system includes an interface for quick access to information about syllabi, methodological materials, recommendations, and evaluation criteria for selected courses. Additionally, it provides the possibility of interaction between higher education students and teachers through message exchanges and discussions on educational matters. The developed system prototype

meets modern requirements for information and communication tools in the field of education and can be used to improve the quality of the educational process.

Keywords: information-reference system (IRS), system architecture, functional capabilities, testing and validation, elective component of the educational program, interaction between teachers and students.

ЗМІСТ

ВСТУП

У сучасному світі, де освіта визначає кращі можливості та успіх в кар'єрі, інформаційні технології відіграють вирішальну роль у поліпшенні якості та доступності освіти. Однією з ключових складових ефективної освіти є інформаційно-довідкові системи, які надають студентам, викладачам та адміністрації університету зручний доступ до інформації про навчальні дисципліни та освітні програми [5].

Актуальність даного дослідження обумовлена необхідністю вдосконалення інформаційно-довідкових систем для забезпечення якісної та комплексної інформації щодо вибіркової компоненти освітньої програми. Вибіркова компонента в освіті грає важливу роль у формуванні індивідуального шляху студента, а отже, розробка ефективної інформаційно-довідкової системи для цього аспекту стає надзвичайно важливою задачею.

Об'єктом та предметом дослідження є інформаційно-довідкова система вибіркової компоненти освітньої програми, а предметом - розробка та впровадження системи на мові програмування C# [19].

Метою даного дослідження є розробка інформаційно-довідкової системи змісту навчальних дисциплін з вибіркової компоненти освітньої програми на мові програмування C#.

Мова програмування C# відмінно підходить для створення інформаційних систем з багатьма причинами [10]:

1. ***Широкі можливості***: C# є мовою програмування загального призначення, що означає, що вона може бути використана для розробки різноманітних програм, включаючи інформаційні системи.

2. ***Платформонезалежність:*** C# може бути використаний для розробки програм для різних платформ, таких як Windows, macOS і Linux. Це забезпечує гнучкість у виборі платформи для розгортання інформаційної системи.

3. ***Широкі можливості для розробки веб-додатків:*** C# використовується разом з платформою ASP.NET для розробки веб-додатків. За допомогою ASP.NET можна створювати потужні веб-сайти та веб-додатки з використанням різних технологій, таких як MVC (Model-View-Controller) або Web API.

4. ***Масштабованість і продуктивність:*** C# є мовою програмування з високим рівнем продуктивності та швидкодії. Вона підтримує механізми для роботи з паралельним програмуванням, що дозволяє покращити ефективність інформаційних систем, особливо в умовах великого обсягу даних та великого навантаження.

5. ***Багатофункціональність:*** C# має велику кількість бібліотек та фреймворків, що робить його відмінним вибором для розробки різноманітних функціональних частин інформаційних систем.

6. ***Інтеграція з іншими технологіями Microsoft:*** C# тісно інтегрується з іншими технологіями Microsoft, такими як Microsoft SQL Server, Azure, і Office. Це дозволяє створювати інформаційні системи, які легко інтегруються з існуючими екосистемами Microsoft [21].

Завдання дослідження:

1. Провести аналіз існуючих інформаційно-довідкових систем у галузі освіти.
2. Розглянути теоретичні аспекти використання мови програмування C# у розробці інформаційних систем.
3. Визначити вимоги до нової системи.
4. Розробити архітектуру системи та реалізувати необхідні функціональні модулі.

У роботі буде проведено аналіз існуючих інформаційно-довідкових систем у галузі освіти, детально розглянуті теоретичні аспекти використання мови програмування C# у розробці інформаційних систем. На основі цього аналізу буде визначено вимоги до нової системи, розроблена її архітектура та реалізовані необхідні функціональні модулі.

Отримана інформаційно-довідкова система [11] буде піддана експериментальному дослідженню для оцінки її ефективності та можливостей.

Висновки дослідження сприятимуть покращенню освітнього процесу та забезпечать студентам та педагогам ефективні інструменти для навчання та викладання.

1 АНАЛІЗ ІСНУЮЧИХ ІНФОРМАЦІЙНО-ДОВІДКОВИХ СИСТЕМ

1.0 Що являють собою експертні системи

Експертна навчальна система (ЕНС) – це комп’ютерна програма, побудована на основі знань експертів предметної області (кваліфікованих викладачів, методистів, психологів), здійснююча та контролююча процес навчання. Призначення такої системи полягає в том, що вона, з одного боку допомагає викладачу навчати та контролювати здобувачів вищої освіти, а з іншого – студенту самостійно навчатися [5].

Навчальні системи дають можливість змоделювати кількість та якість знань студента та його здатність використовувати свої знання в рішенні тієї чи іншої задачі. Окрім того, системи виявляють помилки студента та вказують йому їх, аналізують модель та будують плани виправлення відзначених помилок.

В багатьох вузах вказані системи широко використовуються при навчанні здобувачів вищої освіти, але в той же час їх використання має істотні обмеження. Вони визначаються тим, що в основі таких програм лежать, зазвичай, жорсткі конструкції баз даних з однозначними логічними багатокритеріальними зв’язками. В теперішній час найбільші перспективи використання інтелектуальних навчальних систем лежать в сфері практичного навчання, формуванні умінь та навичок прийняття рішень в тій чи іншій сфері діяльності [5].

Аналіз літературних джерел дозволив визначити, що переваги ЕНС полягають у тому, що вони:

дозволяють на основі накопичувальної бази знань відбивати досвід роботи експертів і вибирати кращі алгоритми навчання для подальшого використання;

накопичують статистичну інформацію за декількома параметрами і дозволяють простежити успішність кожного студенту в динаміці;

стимулюють у здобувачів вищої освіти творче мислення, підсилюють значимість їхньої самостійної роботи;

використовуються не тільки на локальному комп'ютері, але й на вилученому – через комп'ютерну мережу.

До недоліку існуючих ЕНС можна віднести обмежені методи організації діалогу із студентом, а також нерозвинені системи пояснення ходу роботи системи.

Отже, експертні навчальні системи грають значну роль у проблемному навчанні, яке дозволяє активізувати розумову діяльність осіб, що навчаються, заставляє їх знаходити нестандартні рішення різних задач та проблем [5].

1.1 Проблеми розвитку експертних систем

На початку 1980-х років у дослідженнях з штучного інтелекту сформувався самостійний напрямок, який отримав назву «експертні системи» (ЕС). Дослідники для назви дисципліни часто використовують також термін «інженерія знань», запроваджений Е. Фейгенбаумом [8].

Кожна експертна система складається з трьох частин: по-перше, з дуже великої бази сучасних даних, по-друге, підсистеми формування питань і, по-третє, сукупності правил, що дозволяють робити висновки [4]. Деякі експертні системи можуть розповісти про метод, який вони використовують при виробленні свого висновку.

Сучасні дослідження у галузі застосування та розробки експертних систем в освіті, на нашу думку, умовно можна поділити на три групи.

До першої групи можна віднести авторів, що досліджують теоретико-педагогічні аспекти застосування експертних систем в освіті. До другої групи – авторів, які розробили конкретні експертні системи для навчання спільно з викладачами на основі відомих технологій [5]. До третьої групи – авторів, які досліджують нові підходи до створення експертних систем системи освіти.

Розглянемо першу групу публікацій, які аналізують теоретико-педагогічні аспекти застосування експертних систем.

1.2 Аналіз існуючих експертних систем з обраної теми

Узагальнюючи праці, в яких розкриваються питання, можна виділити кілька напрямів проведених наукових досліджень:

– визначення сутності ЕС як філософської категорії (Р.Акоф, У.Росс Ешбі, М.Тод, Е.Шуфорд та ін.) [8];

– використання ЕС у різних галузях виробництва (сільське господарство, бізнес, хімія, комунікації, комп'ютерні системи, освіта, електроніка, управління інформацією, виробництво, медицина, армія, наука, космос, транспорт та ін.) з метою розв'язання складних задач;

– використання ЕС в організації контролю знань (Д.Драг, Б.Едельсон, Р.Лєвін, Ю.Рамський та ін.) [8];

– розробка ЕС з окремих курсів інформатики з метою більш глибокої й ефективної перевірки знань здобувачів вищої освіти, а також надання їм допомоги у вивченні матеріалу (І.Адамович, Л.Бабанін, В.Берестова, П.Брусиловський, В.Даніелян, О.Заволович, Л.Поспєлова, Г.Рибіна, Д.Червік, О.Шудрова та ін.) [8].

Експертні системи здатні поповнювати свої знання в ході взаємодії з експертом. Експерт, знання якого вводяться в систему, може не бути знайомим з деталями програми і обчислювальною машиною, на якій реалізована експертна система. Тому з'являється необхідність зручного та зрозумілого інтерфейсу користувача для заповнення бази знань, щоб уникнути залучення додаткових інженерів для роботи з базою знань [7].

Це все зумовлює актуальність експертних систем, що накопичують знання висококваліфікованих експертів, у даному випадку в області педагогіки, та надають своєчасну рекомендацію здобувачам вищої освіти. У

даній роботі порушено питання використання експертної системи для аналізу навчальних дисциплін та вирішення студентом напрямку вивчення додаткових дисциплін професійної підготовки за спеціальністю.

Дані системи надають інтерфейс, що допомагає спілкуватися користувачеві та експерту. Експерт надає всю відому інформацію про навчальну дисципліну користувачеві, а той, в своєю чергою, намагається обрати йому необхідний. Найчастіше такі системи реалізовані у вигляді форуму [6].

Гідність такого підходу полягає в тому, що користувач може обрати дисципліни професійної підготовки по невеликому опису, отже, такі системи іноді мають високу якість обробки інформації.

Недолік – експертні системи не завжди є повністю автоматизованими. Їхнє функціонування може вимагати постійного оновлення експертних знань для врахування нових дисциплін та навчальної програми.

1.3 Висновки за розділом

Моделі представлення інформації (знань) – це один із найважливіших напрямів досліджень у галузі інформаційно-довідкових систем. На сьогоднішній день розроблено вже достатню кількість моделей. Кожна з них має свої плюси та мінуси, і тому для кожного конкретного завдання необхідно вибрати саме свою модель. Від цього залежатиме не так ефективність виконання поставленого завдання, як можливість її вирішення взагалі.

У розглянутій задачі предметна область є безліч причинно-наслідкових зв'язків між зовнішніми факторами та їх впливом на кінцевий результат.

Для вирішення поставленої задачі доцільно використати продукційну модель. Дана модель, заснована на правилах, дозволяє уявити знання у вигляді пропозицій типу: "ЯКЩО – ТО" [11].

2 ПРОЄКТУВАННЯ ЕКСПЕРТНОЇ СИСТЕМИ

2.1 Аналіз варіантів використання

Подання знань - ще одна функція експертної системи. Теорія уявлення знань - це окрема галузь досліджень, тісно пов'язана з філософією формалізму та когнітивною психологією. Предмет дослідження у цій галузі - методи асоціативного зберігання інформації, подібні до тих, які існують у мозку людини. При цьому основна увага, природно, приділяється логічному, а не біологічному боці процесу, опускаючи подробиці фізичних перетворень [11].

При проектуванні експертної системи серйозна увага повинна бути приділена і тому, як здійснюється доступ до знань і як вони використовуються при пошуку рішення [7]. Знання про те, які знання потрібні у тій чи іншій конкретній ситуації, та вміння ними розпорядитися – важлива частина процесу функціонування експертної системи. Такі знання одержали найменування метазнань - тобто знань про знання. Вирішення нетривіальних проблем потребує і певного рівня планування та управління при виборі, яке питання потрібно поставити, який тест виконати, тощо.

В даний час склалася певна технологія розробки ЕС, яка включає наступні шість етапів [11]:

1. Ідентифікація.
2. Концептуалізація.
3. Формалізація.
4. Розробка прототипу.
5. Експериментальна експлуатація.
6. Розробка товару.
7. Промислова експлуатація.

2.2 Діаграма прецедентів (*Use case diagram*)

У ролі системи (*System*) діаграми прецедентів [5] буде наша програма «Інформаційно-довідкова система для змісту навчальних дисциплін у вибірковій компоненті освітньої програми». Будемо позначати її прямокутником.

Прецеденти (*Use case*) [5] – це функції системи, означають всі дії які виконуються над системою. Позначатимемо її овалом.

Актори (*Actors*) [5] – це користувачі системи. Коли одна система є актором іншої системи, позначити систему акторів із стереотипами акторів.

Зв’язки або відношення (Relationships) [5] – це зв’язки між актором і прецедентом представляються стрілками, що помічені як «include» або «extend». Зв’язок або відношення «include» вказує, що один прецедент необхідний іншому для виконання завдання. Зв’язок або відношення «extend» вказує на альтернативні варіанти при певному прецеденті (Рис. 2.1).

Таким чином експертна система змісту навчальних дисциплін має наступний вигляд:

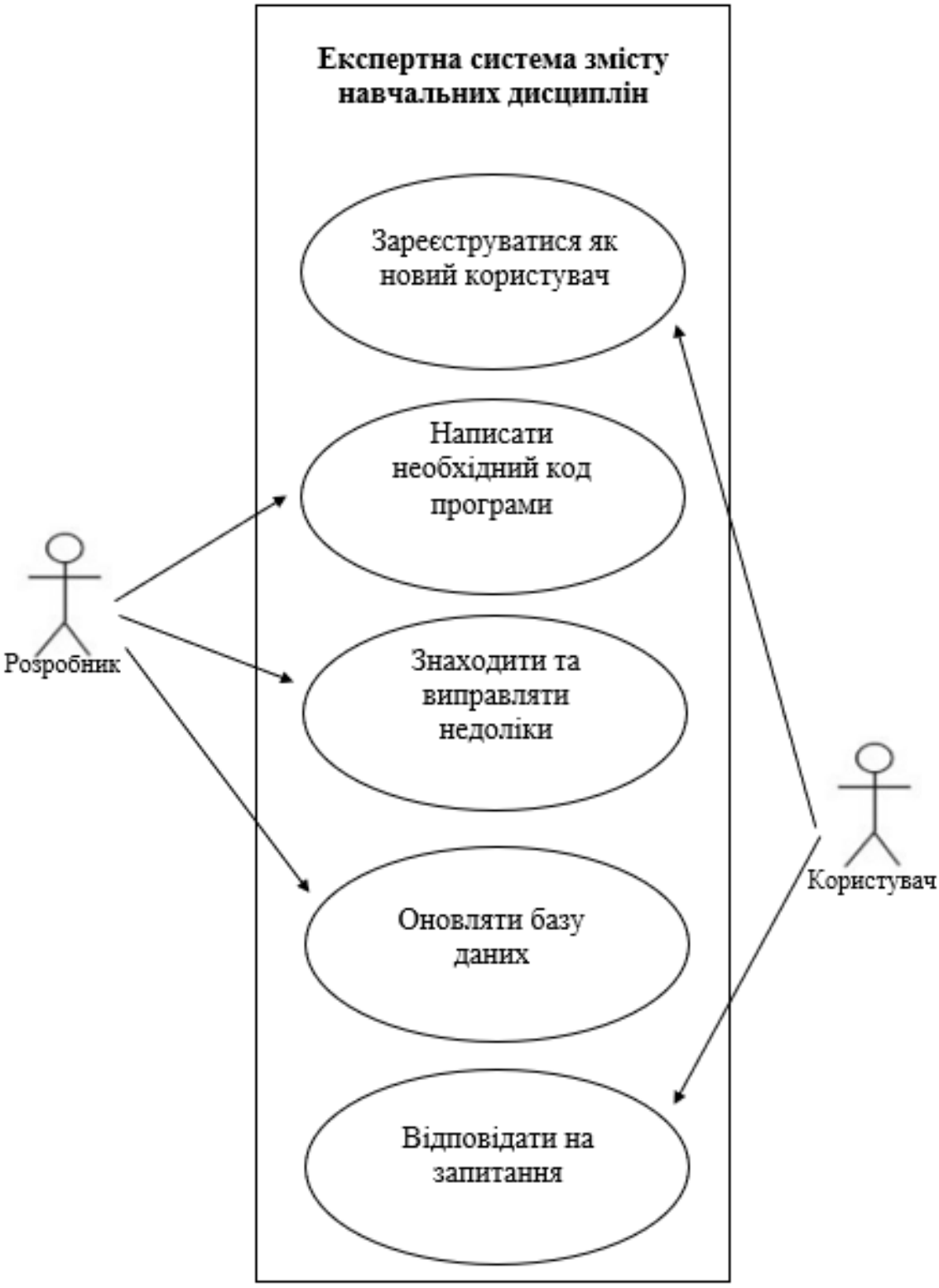

Рис. 2.2.1 – Діаграма прецендентів

2.3 Етап ідентифікації експертної системи

На етапі ідентифікації провадиться:

- неформальне осмислення завдань, які має вирішувати створювана ЕС;
- формування вимог до ЕС;
- визначення ресурсів, необхідних для створення ЕС.

В результаті ідентифікації [7] **функціонально** визначається, що має робити ЕС і що необхідно для її створення.

Ідентифікація завдання полягає у складанні неформального (вербального, тобто словесного) опису, в якому зазначаються:

- загальні характеристики задачі;
- підзавдання, що виділяються всередині даного завдання, розглядаються ключові поняття (об'єкти), їх вхідні та вихідні дані;
- імовірний вид рішення;
- знання, які стосуються вирішуваної задачі.

У процесі ідентифікації завдання інженер зі знань та експерт працюють у тісному контакті [12].

Початковий неформальний опис завдання, даний експертом, використовується інженером знань для уточнення термінів і ключових понять.

Експерт коригує опис завдання, пояснює, як вирішувати її та які міркування лежать в основі того чи іншого рішення.

Після кількох циклів, що уточнюють опис, експерт та інженер за знаннями отримують остаточний неформальний опис завдання.

Під час створення ЕС [7] основними видами ресурсів є:

- джерела знань (експерти);
- інженери знань та програмісти;
- інструментальні програмні засоби (експертні оболонки);
- обчислювальні засоби;

- час розробки;
- обсяг фінансування.

2.4 Етап концептуалізації експертної системи

На даному етапі проводиться змістовний аналіз проблемної галузі, виявляються поняття та їх взаємозв'язки, що використовуються, визначаються методи вирішення завдань.

Цей етап завершується створенням моделі предметної області, що включає основні концепти та відносини між ними. На етапі концептуалізації визначаються такі особливості завдання:

- типи доступних даних;
- вихідні та виведені дані,
- підзавдання загального завдання;
- використовувані стратегії та гіпотези;
- види взаємозв'язків між об'єктами ПЗ [5], типи використовуваних відносин (ієрархія, причина – наслідок, частина – ціле тощо);
- процеси, які використовуються в ході рішення;
- склад знань, що використовуються під час вирішення задачі;
- типи обмежень, що накладаються на процеси, що використовуються під час рішення;
- склад знань, що використовуються для обґрунтування рішень.

Існує два підходи до процесу побудови моделі предметної галузі:

1. Атрибутивний підхід [8] (атрибутами називають суттєві ознаки) передбачає наявність отриманої від експертів інформації у вигляді ланцюжків: "Клас (градація класифікаційної шкали) – об'єкт навчальної вибірки – атрибут (описова шкала) – значення атрибута (градація описової шкали)". Цей підхід розвивається в рамках напряму, який отримав назву формування знань або "машинне навчання" (machine learning).

2. Структурний чи когнітивний підхід [8], заснований на виділенні елементів предметної галузі, їх взаємозв'язків та семантичних (смислових) відносин.

Атрибутивний підхід вимагає повної інформації про предметну область: об'єкти, їх атрибути і значення атрибутів, а також додаткової навчальної інформації про належність конкретних об'єктів до узагальнених класів, що задається експертом [4]. Зазначимо, що атрибутивний підхід у експертних системах має дуже багато спільного з методами, які застосовуються у розпізнаванні образів.

Структурний підхід до побудови моделі предметної області передбачає виділення таких когнітивних елементів знань:

1. Поняття.
2. Взаємозв'язки.
3. Метапоняття.
4. Семантичні відносини.

Поняття предметної області [5], що виділяються, повинні утворювати систему, під якою розуміється сукупність понять, що володіє наступними властивостями:

- мінімальністю (унікальністю, відсутністю надмірності);
- повнотою (досить повним описом різних процесів, фактів, явищ предметної області);
- достовірністю (адекватністю, валідністю)
- відповідністю виділених одиниць смислової інформації їх реальним найменуванням).

Існує ряд методів виявлення ієрархічної системи понять та метапонять (включаючи відносини між ними) [8], що дозволяє адекватно відобразити предметну область:

1. Метод локального уявлення.
2. Метод обчислення коефіцієнта використання.

3. Метод формування переліку понять.
4. Рольовий метод.
5. Методу складання списку елементарних процесів.
6. Методи складання змісту підручника.
7. Текстологічний метод.
8. Метод вільних асоціацій визначення "змістової відстані" між поняттями.
9. Метод "сортування карток".
10. Метод виявлення регулярностей.
11. Методи семантичного диференціалу та репертуарних грат.

Перелічені методи застосовуються етапі концептуалізації при побудові моделі предметної області.

2.5 Етап формалізації експертної системи

Етап формалізації [9] необхідний перетворення декларативних і процедурних знань про предметної області, отриманих на етапі концептуалізації, у форму, придатну їх обробки на комп'ютері.

На цьому етапі:

- вибирається або розробляється формальна мова, що забезпечує подання знань та маніпулювання ними;
- здійснюється формалізація знань, тобто. вони перетворюються на форму, придатну обробки на комп'ютері.

Способи представлення знань: кадри, сценарії, семантичні мережі, продукції.

Методи маніпулювання знаннями: логічний висновок, аналітична модель, статистична модель.

Як середовище реалізації ЕС обрано об'єктно-орієнтоване середовище програмування .Net C# [20] WinForms [25], оскільки воно у порівнянні з іншими має наступні переваги:

1. Зручність введення та виведення інформації за допомогою запитів, що є важливим фактором для написання цієї експертної системи.
2. Легкість написання функцій та легкість налагодження. Це є основним чинником у виборі мови програмування, так як робота системи побудована на циклічності, то легко за допомогою функцій обробляти велику кількість інформації, що вводиться і виводиться.
3. Можливість змінювати розміри та розташування елементів інтегрованого середовища розробки WinForms оскільки воно поєднує різні функції: проектування, редагування, компіляцію і налагодження додатків.
4. Реалізація розробником максимально гнучкого та зручного інтерфейсу для своєї ЕС [5].
5. Створення меню.
6. Обробка подій миші та клавіатури.

2.6 Висновки за розділом

У експертній системі для аналізу та виведення рішення використовуватиметься пошук у глибину [5]. Є питання, які не залежать від попереднього, і націлені на більш детальне вивчення побажань користувача.

Програма не вимагає установки: для роботи достатньо скопіювати папку на жорсткий диск комп'ютера.

Для початку роботи запустіть файл SpecialCorse.sln. У операційних системах сімейства Windows за замовчуванням це здійснюється подвійним клацанням лівої кнопки миші по відповідній іконці.

Програма працює у діалоговому режимі. Необхідно прочитувати інформацію, що виводиться на екран, а для вибору тієї чи іншої дії або варіанта відповіді слід натиснути відповідну кнопку на формі програми.

3 ПОНЯТТЯ ТА ВИДИ ІНФОРМАЦІЙНИХ СИСТЕМ

3.1 Поняття інформаційна система

Під системою розуміють будь-який об'єкт, який одночасно розглядається як єдине ціле, і як об'єднана на користь досягнення поставленої мети сукупність різнорідних елементів. Системи значно відрізняються між собою як за складом, так і за основними цілями.

Інформаційна система [19] – це система, яка здійснює: отримання вхідних даних; обробку цих даних та/або зміна власного внутрішнього стану (внутрішніх зв'язків/відносин); видачу результату чи зміна свого зовнішнього стану (зовнішніх зв'язків/відносин).

Простою інформаційною системою назвемо систему, елементи якої функціонують відповідно до правил, породжених одним і тим самим взаємонесуперечливим безліччю аксіом.

Складною інформаційною системою назвемо систему, яка містить елементи, що функціонують відповідно до правил, породжених відмінними один від одного множинами аксіом. При цьому допускається, що серед правил функціонування різних елементів можуть бути взаємосуперечливі правила та цілі.

Зважаючи на зазначене, системи можуть бути класифіковані за різними ознаками: за ступенем складності, за типом взаємодії елементів, за метою функціонування тощо [17]. Крім того, важливим аспектом є здатність систем до адаптації та розвитку в умовах змінного середовища. Такі системи називаються динамічними і можуть змінювати свою структуру та функціональні властивості для досягнення нових цілей або для покращення ефективності. Особливо важливою є класифікація систем у контексті інформаційних технологій, де системи відіграють ключову роль у забезпеченні обробки, зберігання та передачі інформації.

Наведемо кілька систем, які з різних елементів і вкладених у реалізацію різних цілей (табл. 3.1.1).

Таблиця 3.1.1 - Приклади різних типів систем за складом і цілями

Система	**Елементи системи**	**Головна мета системи**
Фірма	Люди, обладнання, матеріали, будівлі та ін.	Виробництво товарів
Комп'ютер	Електронні та електромеханічні елементи, лінії зв'язку та ін.	Обробка даних
Телекомунікаційна система	Комп'ютери, модеми, кабелі, мережне програмне забезпечення та ін.	Передача інформації
Інформаційна система	Комп'ютери, комп'ютерні мережі, люди, інформаційне та програмне забезпечення	Виробництво професійної інформації

В інформатиці поняття "система" поширене і має безліч смислових значень. Найчастіше воно використовується стосовно набору технічних засобів і програм. Системою може бути апаратна частина комп'ютера. Системою може вважатися безліч програм на вирішення конкретних прикладних завдань, доповнених процедурами ведення документації та управління розрахунками.

Додавання до поняття "система" слова "інформаційна" відображає мету її створення та функціонування. Інформаційні системи забезпечують збирання, зберігання, обробку, пошук, видачу інформації, необхідної в процесі прийняття рішень задач з будь-якої галузі. Вони допомагають аналізувати проблеми та створювати нові продукти.

Інформаційна система [19] - взаємопов'язана сукупність засобів, методів та персоналу, що використовуються для зберігання, обробки та видачі інформації на користь досягнення поставленої мети.

Сучасне розуміння інформаційної системи передбачає використання як основний технічний засіб переробки інформації персонального комп'ютера. У великих організаціях поруч із персональним комп'ютером у складі технічної бази інформаційної системи може входити мейнфрейм чи суперЭВМ [18]. Крім того, технічне втілення інформаційної системи саме по собі нічого не означатиме, якщо не враховано роль людини, для якої призначена вироблена інформація і без якої неможливе її отримання та подання.

Увага! Під організацією розумітимемо співтовариство людей, об'єднаних спільними цілями та які використовують спільні матеріальні та фінансові засоби для виробництва матеріальних та інформаційних продуктів та послуг. У тексті на рівноправних засадах будуть вживатися два слова: "організація" та "фірма".

Необхідно розуміти різницю між комп'ютерами та інформаційними системами. Комп'ютери, що оснащені спеціалізованими програмними засобами, є технічною базою та інструментом для інформаційних систем. Інформаційна система немислима без персоналу, що взаємодіє з комп'ютерами та телекомунікаціями.

3.2 Етапи розвитку інформаційних систем

Історія розвитку інформаційних систем [9] та цілі їх використання на різних періодах представлені в (табл.3.2.2).

Ця таблиця ілюструє еволюцію інформаційних систем від початкових етапів їх впровадження до сучасного стану. Вона показує, як змінювалися завдання та функції інформаційних систем у відповідь на зростаючі потреби організацій та технологічний прогрес.

Таблиця також відображає ключові етапи використання інформаційних систем для підтримки бізнес-процесів та прийняття управлінських рішень.

Таблиця 3.2.2 - Історія розвитку та цілі використання інформаційних систем

Період часу	**Концепція використання інформації**	**Вид інформаційних систем**	**Ціль використання**
1950-1960 рр.	Паперовий потік розрахункових документів	Інформаційні системи обробки розрахункових документів на електромеханічних бухгалтерських машинах	Підвищення швидкості обробки документів.
1960-1970 рр.	Основна допомога у підготовці звітів	Управлінські інформаційні системи для виробничої інформації	Прискорення процесу підготовки звітності
1970-1980 рр.	Управлінський контроль реалізації (продажів)	Системи підтримки ухвалення рішень. Системи для вищої ланки управління	Вибірка найбільш раціонального рішення
1980-2000 рр.	Інформація - стратегічний ресурс, що забезпечує конкурентну перевагу	Стратегічні інформаційні системи. Автоматизовані офіси	Виживання та процвітання фірми

Перші інформаційні системи з'явилися торік у 50-х гг [7]. У ці роки вони були призначені для обробки рахунків та розрахунку зарплати, а реалізовувалися на електромеханічних бухгалтерських рахункових машинах. Це призводило до деякого скорочення витрат та часу на підготовку паперових документів.

60-ті роки [5]. знаменуються зміною ставлення до інформаційних систем. Інформація, отримана з них, стала застосовуватись для періодичної звітності за багатьма параметрами. Дня цього організаціям було потрібно комп'ютерне обладнання широкого призначення, здатне обслуговувати безліч функцій, а не лише обробляти рахунки та рахувати зарплату, як було раніше.

У 70-х – на початку 80-х рр. XX ст. [7] інформаційні системи починають широко використовуватися як засіб управлінського контролю, що підтримує та прискорює процес прийняття рішень.

До кінця 80-х років [6]. Концепція використання інформаційних систем знову змінюється. Вони стають стратегічним джерелом інформації та використовуються на всіх рівнях організації будь-якого профілю. Інформаційні системи цього періоду, надаючи вчасно потрібну інформацію, допомагають організації досягти успіху у своїй діяльності, створювати нові товари та послуги, знаходити нові ринки збуту, забезпечувати собі гідних партнерів, організовувати випуск продукції за низькою ціною та багато іншого.

3.3 Процеси в інформаційній системі

Процеси, які забезпечують роботу інформаційної системи будь-якого призначення, умовно можна як схеми (рис. 3.3.1), що з блоків [8]:

- введення інформації із зовнішніх чи внутрішніх джерел;
- обробка вхідної інформації та подання її у зручному вигляді;

- виведення інформації для подання споживачам або передачі до іншої системи;
- зворотний - це інформація, перероблена людьми цієї організації для корекції вхідної інформації

Рис. 3.3.1 - Процеси в інформаційній системі

Інформаційна система визначається такими властивостями [8]:

- будь-яка інформаційна система може бути піддана аналізу, побудована та керована на основі загальних принципів побудови систем; \
- інформаційна система є динамічною та розвивається;
- при побудові інформаційної системи необхідно використовувати системний підхід;
- вихідною продукцією інформаційної системи є інформація, на основі якої приймаються рішення;
- інформаційну систему слід сприймати як людино-комп'ютерну систему обробки інформації.

В даний час склалася думка про інформаційну систему як про систему, реалізовану за допомогою комп'ютерної техніки. Хоча у випадку інформаційну систему можна розуміти й у некомп'ютерному варіанті.

Щоб розібратися в роботі інформаційної системи, необхідно зрозуміти суть проблем, які вирішує, а також організаційні процеси, в які вона включена [10]. Так, наприклад, щодо можливості комп'ютерної інформаційної системи для підтримки прийняття рішень слід враховувати:

- структурованість розв'язуваних управлінських завдань;
- рівень ієрархії управління фірмою, у якому рішення має бути прийнято;
- належність розв'язуваного завдання до тієї чи іншої функціональної сфери бізнесу;
- вид використовуваної інформаційної технології.

Технологія роботи в комп'ютерній інформаційній системі доступна для розуміння фахівцем некомп'ютерної галузі та може бути успішно використана для контролю процесів професійної діяльності та управління ними.

Структура інформаційної системи. Структуру інформаційної системи становить сукупність окремих її частин, які називаються підсистемами.

Підсистема - це частина системи, виділена за якоюсь ознакою [11].

Загальну структуру інформаційної системи можна як сукупність підсистем незалежно від сфери застосування. У цьому випадку говорять про *структурну ознаку* класифікації, а підсистеми називають такими, що забезпечують. Таким чином, структура будь-якої інформаційної системи може бути представлена сукупністю підсистем, що забезпечують (рис.3.2).

Серед підсистем [7], що забезпечують, зазвичай виділяють інформаційне, технічне, математичне, програмне, організаційне та правове забезпечення.

Призначення підсистеми інформаційного забезпечення полягає у сучасному формуванні та видачі достовірної інформації для прийняття управлінських рішень.

Інформаційне забезпечення (Рис. 3.3.2) – сукупність єдиної системи класифікації та кодування інформації, уніфікованих систем документації, схем

інформаційних потоків, що циркулюють в організації, а також методологія побудови баз даних [6].

Рис. 3.3.2 - Структура інформаційної системи як сукупність підсистем, що забезпечують цілісність системи

Технічне забезпечення – комплекс технічних засобів, призначених для роботи інформаційної системи, а також відповідна документація на ці засоби та технологічні процеси [7].

Комплекс технічних засобів складають:

- комп'ютери будь-яких моделей;
- пристрої збору, накопичення, обробки, передачі та виведення інформації;
- пристрої передачі даних та ліній зв'язку;
- оргтехніка та пристрої автоматичного знімання інформації;
- експлуатаційні матеріали та ін.

Документацією оформляються попередній вибір технічних засобів, організація їхньої експлуатації, технологічний процес обробки даних, технологічне оснащення.

Документацію можна умовно поділити на три групи:

- загальносистемну, що включає державні та галузеві стандарти з технічного забезпечення;
- спеціалізовану, що містить комплекс методик з усіх етапів розробки технічного забезпечення;
- нормативно-довідкову, використовувану під час виконання розрахунків із технічного забезпечення.

На цей час склалися дві основні форми організації технічного забезпечення (форми використання технічних засобів): централізована і частково чи повністю децентралізована.

Централізоване технічне забезпечення базується на використанні в інформаційній системі великих ЕОМ [5] та обчислювальних центрів.

Децентралізація технічних засобів передбачає реалізацію функціональних підсистем на персональних комп'ютерах безпосередньо робочих місцях.

Перспективним підходом слід вважати, очевидно, частково децентралізований підхід - організацію технічного забезпечення з урахуванням розподілених мереж, які з персональних комп'ютерів і великий ЕОМ зберігання баз даних, загальних будь-яких функціональних підсистем.

Математичне та програмне забезпечення [11]- сукупність математичних методів, моделей, алгоритмів та програм для реалізації цілей та завдань інформаційної системи, а також нормального функціонування комплексу технічних засобів.

До засобів *математичного забезпечення* належать:

- засоби моделювання процесів управління;
- типові завдання керування;
- методи математичного програмування, математичної статистики, теорії масового обслуговування та ін.

До складу *програмного забезпечення* входять загальносистемні та спеціальні програмні продукти, а також *технічна документація*.

До *загальносистемного програмного забезпечення* належать комплекси програм, орієнтованих користувачів і призначених на вирішення типових завдань обробки інформації. Вони служать розширення функціональних можливостей комп'ютерів, контролю та управління процесом обробки даних.

Спеціальне програмне забезпечення є сукупність програм, розроблених під час створення конкретної інформаційної системи. До його складу входять пакети прикладних програм (ППП) [5], що реалізують розроблені моделі різного ступеня адекватності, що відображають функціонування реального об'єкта.

Технічна документація на розробку програмних засобів повинна містити опис завдань, завдання на алгоритмізацію, економіко-математичну модель задачі, контрольні приклади.

Організаційне забезпечення [11]– сукупність методів та засобів, що регламентують взаємодію працівників з технічними засобами та між собою у процесі розробки та експлуатації інформаційної системи.

Організаційне забезпечення реалізує такі функції:

- аналіз існуючої системи управління організацією, де використовуватиметься ІВ [6], та виявлення завдань, що підлягають автоматизації;
- підготовку завдань до вирішення на комп'ютері, включаючи технічне завдання на проектування ІВ та техніко-економічне обґрунтування її ефективності;
- розробку управлінських рішень за складом та структурою організації, методологією вирішення завдань, спрямованих на підвищення ефективності системи управління.

Організаційне забезпечення створюється за результатами передпроектного обстеження на 1-му етапі побудови баз даних, з метою якого ви познайомилися під час розгляду інформаційного забезпечення.

Правове забезпечення – сукупність правових норм [5], що визначають створення, юридичний статус та функціонування інформаційних систем, що регламентують порядок отримання, перетворення та використання інформації.

Головною метою правового забезпечення є зміцнення законності.

До складу правового забезпечення входять закони, укази, ухвали державних органів влади, накази, інструкції та інші нормативні документи міністерств, відомств, організацій, місцевих органів влади. У правовому забезпеченні можна виділити загальну частину, що регулює функціонування будь-якої інформаційної системи, та локальну частину, що регулює функціонування конкретної системи.

Правове забезпечення [11] етапів розробки інформаційної системи включає нормативні акти, пов'язані з договірними відносинами розробника та замовника та правовим регулюванням відхилень від договору.

Правове забезпечення етапів функціонування інформаційної системи включає:

- статус інформаційної системи;
- права, обов'язки та відповідальність персоналу;
- правові положення окремих видів процесу керування;
- порядок створення та використання інформації та ін.

3.4 Результати впровадження інформаційних систем

Впровадження інформаційних систем може сприяти:

- отримання більш раціональних варіантів вирішення управлінських завдань за рахунок впровадження математичних методів та інтелектуальних систем тощо;

- звільнення працівників від рутинної роботи за рахунок її автоматизації;
- забезпечення достовірності інформації;
- заміні паперових носіїв даних на магнітні диски або стрічки, що призводить до більш раціональної організації переробки інформації на комп'ютері та зниження обсягів документів на папері;
- вдосконаленню структури потоків інформації та системи документообігу у фірмі;
- зменшення витрат на виробництво продуктів та послуг;
- надання споживачам унікальних послуг;
- пошуку нових ринкових ніш;
- прив'язці до фірми покупців і постачальників за рахунок надання їм різних знижок та послуг.

3.5 Види інформаційних систем

За ознакою структурованості завдань

При створенні чи класифікації інформаційних систем [14] неминуче виникають проблеми, пов'язані з формальним - математичним і алгоритмічним описом розв'язуваних завдань. Від ступеня формалізації багато в чому залежать ефективність роботи всієї системи, і навіть рівень автоматизації, який визначається ступенем участі при прийнятті рішення з урахуванням одержуваної інформації.

Чим точніший математичний опис завдання, тим вищі можливості комп'ютерної обробки даних і тим менший ступінь участі людини у її вирішенні. Це визначає ступінь автоматизації завдання.

Розрізняють три типи завдань [5], для яких створюються інформаційні системи: структуровані (формалізовані), неструктуровані (неформалізовані) та частково структуровані. . За ознакою структурованості завдань.

За ознакою структурованості завдань (рис. 3.5.3) можна класифікувати інформаційні системи на три типи. Структуровані (формалізовані) завдання [15] мають чітко визначені правила та алгоритми розв'язання, що дозволяє максимально автоматизувати процеси та зменшити участь людини. Неструктуровані (неформалізовані) завдання, навпаки, не піддаються повній формалізації через свою складність та неоднозначність, тому потребують значної участі людини у прийнятті рішень. Частково структуровані завдання поєднують в собі елементи як формалізованих, так і неформалізованих завдань, що дозволяє автоматизувати лише окремі етапи їх розв'язання [7].

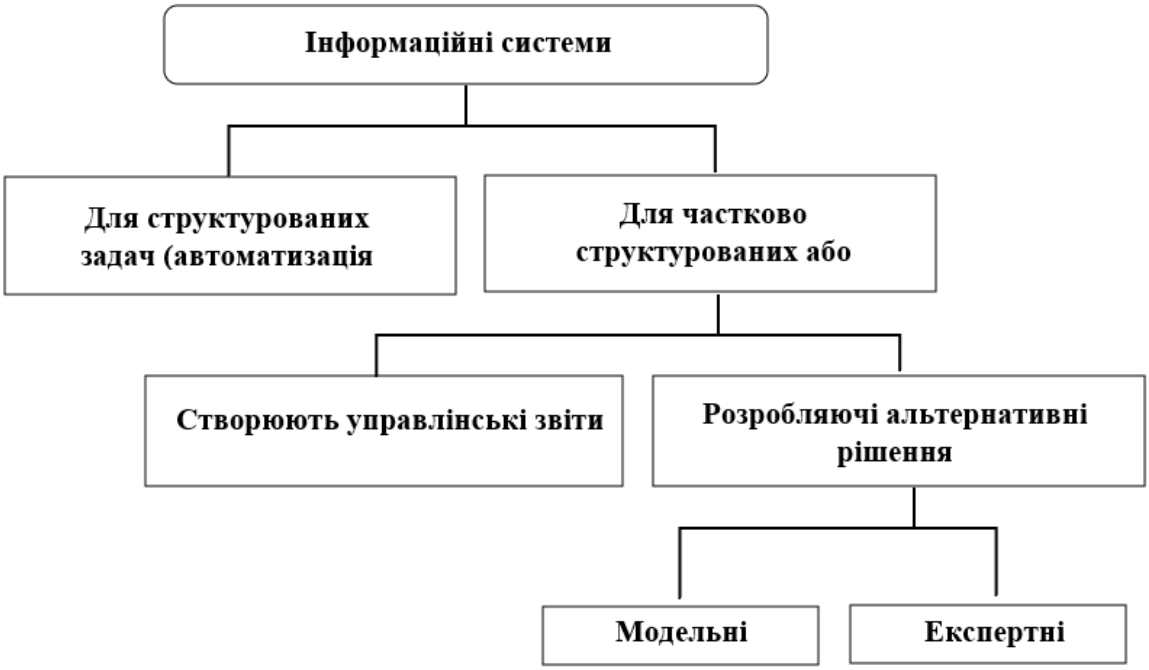

Рис. 3.5.3 - Класифікація інформаційних систем за ознакою структурованості розв'язуваних завдань

Структурована (формалізується) [8] - завдання, де відомі всі її елементи та взаємозв'язки між ними. У *структурованій* задачі вдається виразити її зміст у формі математичної моделі, що має точний алгоритм розв'язання. Подібні завдання зазвичай доводиться вирішувати багаторазово, і вони мають рутинний характер. Метою використання інформаційної системи

на вирішення структурованих завдань є повна автоматизація їх вирішення, тобто. зведення ролі людини нанівець.

Неструктурована (неформалізована) [8] - завдання, у якій неможливо виділити елементи та встановити між ними зв'язки. Вирішення *неструктурованих* завдань через неможливість створення математичного опису та розробки алгоритму пов'язане з великими труднощами. Можливості використання інформаційної системи невеликі. Рішення в таких випадках приймається людиною з евристичних міркувань на основі свого досвіду та, можливо, непрямої інформації з різних джерел.

Зауважимо, що у практиці роботи будь-якої організації існує порівняно трохи повністю структурованих чи зовсім неструктурованих завдань. Про більшість завдань можна сказати, що відома лише частина їх елементів та зв'язків між ними. Такі завдання називаються **частково структурованими** [8]. У умовах можна створити інформаційну систему. Отримувана в ній інформація аналізується людиною, яка відіграватиме визначальну роль. Такі інформаційні системи є автоматизованими, оскільки у їх функціонуванні бере участь людина.

За рівнем автоматизації.

Залежно від рівня автоматизації інформаційних процесів у системі управління фірмою інформаційні системи визначаються як ручні, автоматичні, автоматизовані (рис. 3.5.4)

Ручні інформаційні системи передбачають виконання всіх операцій вручну, без використання технічних засобів. У автоматизованих системах значну частину інформаційних процесів виконує комп'ютерна техніка, тоді як людина контролює та коригує результати роботи системи, забезпечуючи високу ефективність та точність управлінських рішень.

Автоматичні інформаційні системи [7] виконують всі інформаційні процеси без участі людини, що дозволяє досягти високої продуктивності та мінімізувати ризик помилок.

Таким чином, вибір рівня автоматизації залежить від складності завдань та вимог до точності і швидкості обробки інформації.

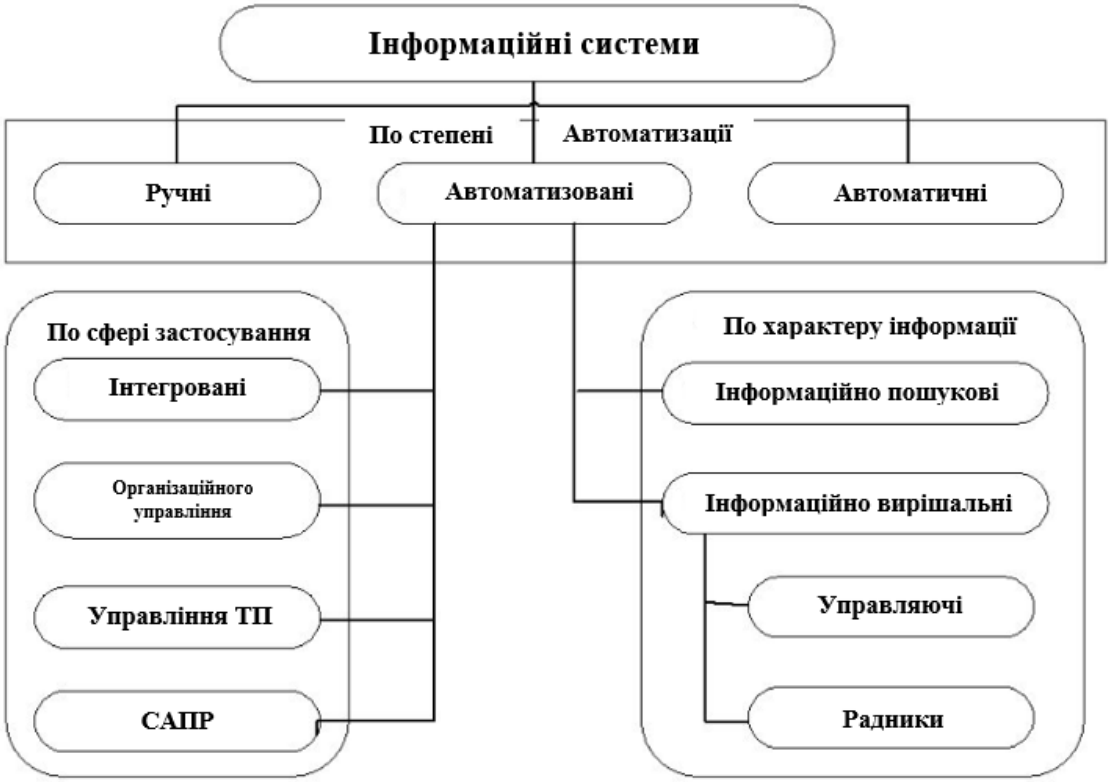

Рис. 3.5.4 - Класифікація інформаційних систем за різними ознаками

Ручні ІВ характеризуються відсутністю сучасних технічних засобів переробки інформації та виконання всіх операцій людиною [9]. Наприклад, про діяльність менеджера у фірмі, де відсутні комп'ютери, можна говорити, що він працює з ручною ІВ.

Автоматичні ІВ виконують усі операції з переробки інформації без участі людини [9].

Автоматизовані ІС припускають участь у процесі обробки інформації та людини, і технічних засобів, причому головна роль відводиться комп'ютеру. [9] У сучасному тлумаченні термін " інформаційна система " вкладається обов'язково поняття автоматизованої системи.

Автоматизовані ІС, враховуючи їх широке використання в організації процесів управління, мають різні модифікації і можуть бути класифіковані, наприклад, характер використання інформації та у сфері застосування [7].

За характером використання інформації.

Інформаційно-пошукові системи (див. рис. 3.5.4) проводять введення, систематизацію, зберігання, видачу інформації на запит користувача без складних перетворень даних [15]. Наприклад, інформаційно-пошукова система у бібліотеці, у залізничних та авіакасах продажу квитків.

Інформаційно-вирішальні системи здійснюють усі операції переробки інформації за певним алгоритмом [15]. Серед них можна провести класифікацію за ступенем впливу виробленої результатної інформації на процес прийняття рішень та виділити два класи: керуючі та радники.

Управляючі ІС виробляють інформацію, на підставі якої людина приймає рішення. Для цих систем характерні тип завдань розрахункового характеру та обробка великих обсягів даних. Прикладом може бути система оперативного планування випуску продукції, система бухгалтерського обліку.

Радячі ІС виробляють інформацію, яка приймається людиною до відома і не перетворюється негайно на серію конкретних дій. Ці системи мають більш високий рівень інтелекту, оскільки їм характерна обробка знань, а чи не даних.

За сферою застосування.

Інформаційні системи **організаційного управління** (див. рис.3.5.4) призначені для автоматизації функцій управлінського персоналу [16]. З огляду на найбільш широке застосування та різноманітність цього класу систем часто будь-які інформаційні системи розуміють саме в даному тлумаченні. До цього класу належать інформаційні системи управління як промисловими фірмами, і непромисловими об'єктами: готелями, банками, торговими фірмами та інших.

Основними функціями подібних систем є: оперативний контроль та регулювання, оперативний облік та аналіз, перспективне та оперативне

планування, бухгалтерський облік, управління збутом та постачанням та інші економічні та організаційні завдання.

ІВ управління технологічними процесами (ТП) служать для автоматизації функцій виробничого персоналу [17]. Вони широко використовуються при організації для підтримки технологічного процесу в металургійній та машинобудівній промисловості.

ІС автоматизованого проектування (САПР) [17] призначені для автоматизації функцій інженерів-проектувальників, конструкторів, архітекторів, дизайнерів під час створення нової техніки чи технології. Основними функціями таких систем є: інженерні розрахунки, створення графічної документації (креслень, схем, планів), створення проектної документації, моделювання об'єктів, що проектуються.

Інтегровані (корпоративні) ІВ використовуються для автоматизації всіх функцій фірми та охоплюють весь цикл робіт від проектування до збуту продукції [17]. Створення таких систем дуже важко, оскільки вимагає системного підходу з позицій головної мети, наприклад, отримання прибутку, завоювання ринку збуту і т.д. Такий підхід може призвести до істотних змін у самій структурі фірми, на що може вирішитись не кожен керуючий.

4 ВВЕДЕННЯ В C#

4.1 Мова C# та платформа .NET

На сьогоднішній момент мова програмування C# [20] одна з найпотужніших мов, що швидко розвиваються і затребуваних в IT-галузі. Зараз на ньому пишуться різні програми: від невеликих десктопних програм до великих веб-порталів і веб-сервісів, що обслуговують щодня мільйони користувачів.

C# вже не молода мова і, як і вся платформа .NET, вже пройшов великий шлях. Перша версія мови вийшла разом із релізом Microsoft Visual Studio .NET у лютому 2002 року. Поточною версією мови є версія C# 12, яка вийшла 14 листопада 2023 разом із релізом .NET 8.

C# є мовою із Сі-подібним синтаксисом і близький у цьому відношенні до C++ та Java. Тому, якщо ви знайомі з однією з цих мов, то опанувати C# буде легше.

C# є об'єктно-орієнтованим і в цьому плані багато перейняв у Java та C++. Наприклад, C# підтримує поліморфізм, успадкування, навантаження операторів, статичну типізацію. Об'єктно-орієнтований підхід дозволяє вирішити завдання по побудові великих, але в той же час гнучких, масштабованих і додатків, що розширюються. І C# продовжує активно розвиватися і з кожною новою версією з'являється все більше цікавих функціональностей (див. ДОДАТОК А).

4.2 Роль платформи .Net

Коли говорять C#, часто мають на увазі технології платформи .NET (Windows Forms, WPF, ASP.NET, .NET MAUI) [27]. І навпаки, коли говорять .NET, нерідко мають на увазі C#. Однак, хоча ці поняття пов'язані,

ототожнювати їх не так. Мова C# була створена спеціально для роботи з фреймворком .NET, проте саме поняття .NET дещо ширше.

Якось Білл Гейтс сказав, що платформа .NET – це найкраще, що створила компанія Microsoft. Можливо, він мав рацію. Фреймворк .NET представляє потужну платформу створення додатків. Можна виділити такі основні риси:

- Підтримка кількох мов. Основою платформи є загальномовне середовище виконання Common Language Runtime (CLR), завдяки чому. NET. При компіляції код будь-якою з цих мов компілюється у складання загальною мовою CIL (Common Intermediate Language) - свого роду асемблер платформи .NET. Тому за певних умов ми можемо зробити окремі модулі однієї програми окремими мовами.

- Кросплатформність. .NET є платформою, що переноситься (з деякими обмеженнями). Наприклад, остання версія платформи на даний момент – .NET 8 підтримується на більшості сучасних ОС Windows, MacOS, Linux. Використовуючи різні технології на платформі .NET, можна розробляти програми на мові C# для різних платформ - Windows, MacOS, Linux, Android, iOS, Tizen.

- Потужна бібліотека класів. .NET представляє єдину всім підтримуваних мов бібліотеку класів. І який би додаток ми не збиралися писати на C# - текстовий редактор, чат або складний веб-сайт - так чи інакше ми використовуємо бібліотеку класів .NET.

- Різноманітність технологій. Загальномовне середовище виконання CLR та базова бібліотека класів є основою цілого стеку технологій, які розробники можуть задіяти при побудові тих чи інших додатків. Наприклад, для роботи з базами даних у цьому стеку технологій призначено технологію ADO.NET та Entity Framework Core. Для побудови графічних додатків з багатим насиченим інтерфейсом – технологія WPF та WinUI, для створення більш простих графічних додатків – Windows Forms. Для розробки

кросплатформових мобільних та десктопних програм - Xamarin/MAUI. Для створення веб-сайтів та веб-додатків - ASP.NET і т.д. До цього варто додати активний Blazor - фреймворк, що розвивається і набирає популяність, який працює поверх .NET і який дозволяє створювати веб-додатки як на стороні сервера, так і на стороні клієнта. А в майбутньому підтримуватиме створення мобільних додатків та, можливо, десктоп-додатків.

- Продуктивність. Згідно з рядом тестів веб-програми на .NET у ряді категорій сильно випереджають веб-програми, побудовані за допомогою інших технологій. Програми на .NET у принципі відрізняються високою продуктивністю.

Також слід відзначити таку особливість мови C# і фреймворку .NET, як автоматичне складання сміття. А це означає, що нам здебільшого не доведеться, на відміну від C++ [28], дбати про звільнення пам'яті. Вищезгадане загальномовне середовище CLR сама викличе збирач сміття та очистить пам'ять.

4.3 .NET Framework и .NET 8

Варто зазначити, що .NET тривалий час розвивався переважно як платформа для Windows під назвою .NET Framework. У 2019 році вийшла остання версія цієї платформи - .NET Framework 4.8. Вона більше не розвивається

З 2014 року Microsoft став розвивати альтернативну платформу - .NET Core, яка вже призначалася для різних платформ і повинна була увібрати в себе всі можливості застарілого .NET Framework і додати нову функціональність. Потім Microsoft послідовно випустив ряд версій цієї платформи: .NET Core 1, .NET Core 2, .NET Core 3, .NET 5. І поточною версією є платформа .NET 8, що розглядається в цьому посібнику. Тому слід розрізняти. переважно для Windows, і кроссплатформенний .NET 8. У цьому посібнику йтиметься про C# 12 у зв'язці з .NET 8, оскільки це актуальна платформа [28].

4.4 Керований та некерований код

Нерідко програму, створену на C#, називають керованим кодом (managed code). Що це означає? А це означає, що ця програма створена на основі платформи .NET і тому управляється загальномовним середовищем CLR, яке завантажує програму і при необхідності очищує пам'ять. Але є також програми, наприклад, створені мовою C++, які компілюються над загальну мову CIL, як C#, VB.NET чи F#, а звичайний машинний код. У цьому випадку .NET не керує програмою [27].

У той же час платформа .NET надає можливості для взаємодії з некерованим кодом.

4.5 JIT-компіляція

Як вище писалося, код C# компілюється в додатки або збірки з розширеннями exe або dll мовою CIL. Далі при запуску на виконання подібної програми відбувається JIT-компіляція (Just-In-Time) [20] у машинний код, який потім виконується. При цьому, оскільки наша програма може бути великою і містити купу інструкцій, в даний момент компілюватиметься лише та частина програми, до якої безпосередньо йде звернення. Якщо ми звернемося до іншої частини коду, то вона буде скомпільована з CIL до машинного коду. При цьому вже скомпільована частина програми зберігається до завершення роботи програми. У результаті це підвищує продуктивність.

4.6 Що розробляють за допомогою C#

Мова C# практично універсальна [25]. Можна використовувати його для створення будь-якого програмного забезпечення: просунутих бізнес-додатків,

відеоігор, функціональних веб-додатків, програм для Windows, macOS, мобільних програм для iOS та Android.

4.7 Відеоігри

C# без перебільшення дуже популярний серед творців відеоігор. Мова використовується для розробки ігор під Windows, MacOS, Android та iOS. Вся справа в Unity – платформа для роботи з 3D-графікою. C# краще інших мов адаптований під роботу з цим двигуном. Тому програмісти зазвичай не вибирають, а одразу використовують зв'язку Unity + C# [27]. (рис. 4.7.1)

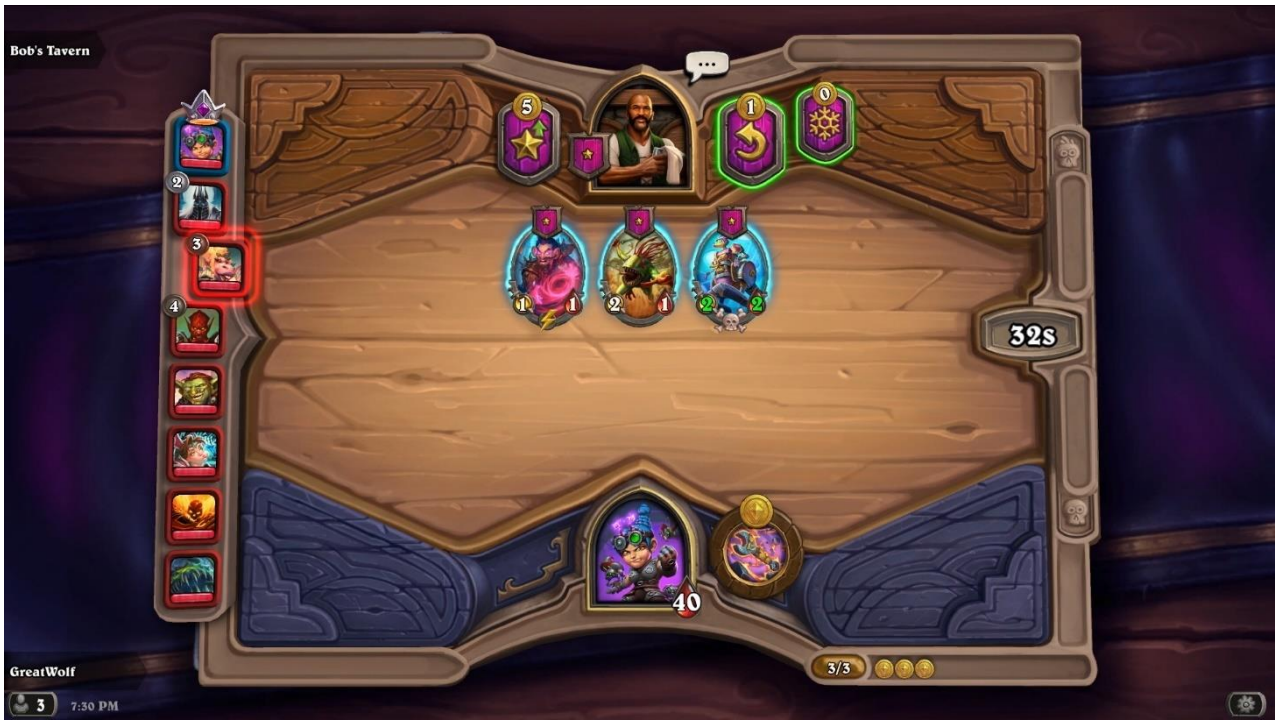

Рис. 4.7.1 – Відеогра створена на мові C#

З популярних проектів варто виділити такі хіти ігрової індустрії, як Bastion (кросплатформна РПГ-адвенчура з ізометричним виглядом), Wasteland (популярний шутер у пост-апокаліптичному всесвіті), знаменитий Doom 3 і Hearthstone (карткова гра у всесвіті World of Warcraft, створений) [28].

4.8 ПЗ для захисту систем

Безпека ваших програм та операційних систем забезпечується завдяки потужним утилітам на базі C# [28]. Колосальна кількість вірусів, що на щоденній основі атакують комп'ютери користувачів, блокується інструментами, створеними за допомогою мови Microsoft. Аналогічна ситуація спостерігається у великому бізнесі – світові корпорації захищаються від атак хакерів за допомогою ПЗ, написаного на C# [20].

4.9 Програми для Windows

Практично вся операційна система Microsoft існує завдяки C#. Звичні утиліти та програми створені з використанням цієї мови та фреймворків, розроблених для неї [20].

До цієї категорії потрапляє месенджер Skype, браузер Internet Explorer, середовище для розробки Visual Studio 2012, Microsoft Office (всі його складові, включаючи Word, PowerPoint, Excel, Outlook тощо) [27].

Сюди можна віднести продукти компанії Adobe (Photoshop, Lightroom), браузер Mozilla Firefox і Winamp.

4.10 Мобільні додатки

У деяких колах програмістів C# вважається чи не найкращою мовою для проектування мобільних додатків. Все завдяки можливості створювати за допомогою цієї мови нативні програми для будь-яких платформ (iOS, Android). Для створення програм, які ідеально працюють на Айфоні та на Андроїд-смартфонах, використовується IDE Xamarin [28].

З відомих програм, написаних C#, варто відзначити Slack, Pinterest, Tableau, The World Bank та інші. "Плиткові" програми, що з'явилися в Windows 8, практично всі побудовані на базі C # і XAML [28].

5 WINDOWS FORMS

5.1 Сучасна модель програмування для створення додатків GUI

Щоб створити GUI програми [25] в Microsoft .NET, потрібно використовувати Windows Forms. Windows Forms – новий стиль побудови програми на базі класів .NET Framework class library. Вони мають власну модель програмування, яка більш досконала, ніж моделі, що базуються на Win32 API або MFC, і вони виконуються в керованому середовищі .NET Common Language Runtime (CLR). Ця стаття дає уявлення про те, що таке Windows Forms, розглядаючи її від моделі програмування до Microsoft Intermediate Language та JIT-транслятора.

Ви вже багато чули, що Microsoft .NET [27] – нова платформа, яка базується на Windows. Це ціла нова парадигма програмування, яка змінить шлях, яким ви думаєте про написання програм для Windows. Вона реалізована на бібліотеці класів .NET Framework class library та містить більш єдину модель програмування, покращений захист та багатші можливості для написання повнофункціональних веб-додатків. І це лише початок.

Windows Forms [25] – одна з найцікавіших можливостей Microsoft .NET. Якщо ви знайомі з MFC (або Windows API), то Windows Forms гарний початок для роботи з .NET Framework class library, тому що вона дозволяє писати традиційні GUI-додатки з вікнами, формами і т.п. речами. Одного разу, почавши працювати з Windows Forms, ви зможете швидко зрозуміти .NET Framework.

Головна вигода від написання Windows-додатків з використанням Windows Forms - це те, що Windows Forms гомогенізують (створюють однорідніщу (гомогеннішу) структуру) програмну модель і усувають багато помилок і протиріч від використання Windows API. Наприклад, кожен досвідчений програміст під Windows знає, що деякі стилі вікна можуть застосовуватися лише до вікна, коли вже створено. Windows Forms значною

мірою усувають таку суперечність. Якщо ви хочете існуючому вікну встановити стиль, який може бути присвоєний тільки в момент створення вікна, то Windows Forms спокійно знищить вікно і знову створить його із зазначеним стилем. Крім того, .NET Framework class library набагато багатший, ніж Windows API [20], і коли ви писатимете програми, використовуючи Windows Forms, ви отримаєте в розпорядження більше можливостей. Написання програми з використанням Windows Forms потребує менше коду, ніж програми, які використовують Windows API або MFC.

Інша вигода від Windows Forms - ви використовуєте той же API, незалежно від мови програмування, яку ви вибрали. У минулому вибір мови програмування керував вибором API. Якщо ви програмували Visual Basic, ви використовували один API (реалізований мовою Visual Basic), тоді як програмісти C використовували Win32 API, а програмісти C++, взагалі кажучи, використовували MFC. MFC-програмісту було важко перейти на Visual Basic і навпаки. Але наразі такого більше немає. Всі програми, які використовують

Windows Forms [25], використовують один API із .NET Framework class library. Знання одного API достатньо дозволить програмісту писати програми фактично будь-якою мовою, який він вибере. Windows Forms не менше, ніж сучасна модель програмування для GUI додатків [27]. На відміну від моделі програмування Win32, в якій багато чого йде ще від Windows 1.0, нова модель була розроблена з урахуванням усіх сучасних вимог. Ціль цієї статті полягає в тому, щоб познайомити читача з моделлю програмування Windows Forms. Щоб компілювати та виконувати приклади коду, наведеного далі, на вашому комп'ютері має бути встановлений пакет Microsoft .NET Framework SDK (.NET Framework SDK Beta 1 доступний на сайті Microsoft).

5.2 Модель програмування Windows Forms

У Windows Forms термін "форма" – синонім вікна верхнього рівня [21]. Головне вікно програми – форма. Будь-які інші вікна верхнього рівня, які має додаток, - також форми. Вікна діалогу також вважаються формами. Незважаючи на назву, програми, які використовують Windows Forms, не виглядають як форми. Подібно до традиційних Windows-додатків програми здійснюють повний контроль над подіями у власних вікнах.

Програмісти бачать Microsoft .NET через лінзу .NET Framework class library. Уявіть MFC набагато більше і ви отримаєте точну картину про ширину і глибину .NET Framework class library [20]. Щоб полегшити протиріччя в позначеннях і надати організацію багатьом сотням класів, .NET Framework class library розбита на ієрархічні розділи на ім'я. Кореневий розділ, System, визначає фундаментальні типи даних, що використовуються всіма програмами .NET.

Програми, які використовують Windows Forms, використовують класи System.WinForms. Цей розділ включає такі класи, як Form, який моделює поведінку вікон чи форм; Menu, що представляє меню; Clipboard, який дає можливість програмам Windows Forms використовувати буфер обміну. Він також містить численні класи, що надають засоби керування, наприклад: Button, TextBox, ListView, MonthCalendar і т.д. [20]. Ці класи можуть бути включені в додаток з використанням тільки імені класу, або з використанням повного імені, наприклад: System.WinForms.Button.

В основі майже кожної програми, написаної із застосуванням Windows Forms, - похідний клас від System.WinForms.Form. Приклад цього класу представляє головне вікно програми. System.WinForms.Form має безліч властивостей та методів, які мають багатий програмний інтерфейс до форм. Бажаєте знати розміри клієнтської області форми? У Windows ви б викликали функцію API GetClientRect. У Windows Forms потрібно використовувати властивості ClientRectangle або ClientSize.

Програми, засновані на Windows Forms, які використовують кнопки, списки та інші типи компонентів Windows [28], використовують класи керування System.WinForms, які значно спрощують програмування керування. Бажаєте створити стилізовану кнопку із зображенням у вигляді фону? Немає проблем. Увімкніть потрібне зображення до об'єкта System.Drawing.Bitmap і призначте його властивості кнопки BackgroundImage. Як щодо керування кольором? Ви коли-небудь намагалися налаштувати колір тла текстового поля? У Windows Forms це просто: потрібно просто привласнити колір властивості BackColor, все відстальне система зробить сама.

Інший важливий "будівельний" [25] блок програми, який використовує Windows Forms - клас System.WinForms на ім'я Application. Цей клас містить статичний метод Run, який завантажує програму та відображає вікно.

Ви скажете: якщо програми, які є Windows Forms, не обробляють повідомлення, як вони відповідають на введення користувача або знають коли малювати? Багато класів мають віртуальні методи, які можна перевизначити... Наприклад, System.WinForms.Form містить віртуальний метод OnPaint, який викликається, коли клієнтська область форми потребує оновлення. OnPaint - один із багатьох віртуальних методів, який можна перевизначити у похідному класі для формування інтерактивних форм.

Інша важлива грань моделі програмування Windows Forms - механізм, який форми використовують для відповіді на введення в меню, засобів керування та інших елементів GUI програми. Традиційні програми Windows обробляють повідомлення WM_COMMAND і WM_NOTIFY [25] використовуючи події процесу Windows Forms. У C# та іншими мовами, які підтримують .NET Common Language Runtime (CLR), події - члени типу першого класу нарівні з методами, полями та властивостями. Фактично всі класи, що управляють (control classes) Windows Forms (а також і багато некеруючих класів) створюють події. Наприклад, кнопка (примірник System.WinForms.Button) після натискання створює подію Click. Форма, яка

повинна відповісти на натискання кнопки, може використовувати наступний код, щоб з'єднати кнопку на обробником події Click:

```
MyButton.Click += new EventHandler (OnButtonClicked);

...

private void OnButtonClicked (object sender, EventArgs e)
{
    MessageBox.Show ("Click!");
}
```

Лістинг 5.2.1 – Приклад підписки на подію та обробки натискання кнопки у C#

EventHandler [25] - спеціальний обробник подій, який виконує метод OnButtonClicked, коли MyButton створює подію Click. Перший параметр OnButtonClicked ідентифікує об'єкт, який викликав подію. Другий параметр переважно безглуздий для події Click, але використовується деякими іншими типами подій, щоб передати додаткову інформацію.

6 MS SQL SERVER ТА T-SQL

6.1 Що таке SQL Server та T-SQL

SQL Server [22] є однією з найпопулярніших систем управління базами даних (СУБД) у світі. Ця СУБД(Система управління базами даних) підходить для різних проектів: від невеликих додатків до великих високонавантажених проектів.

SQL Server було створено компанією Microsoft. Перша версія вийшла 1987 року [22]. А поточною версією є версія 2022, яка вийшла у листопаді 2022 року і яка використовуватиметься у поточному посібнику.

SQL Server довгий час був виключно системою керування базами даних для Windows, проте починаючи з версії 16 ця система доступна і на Linux.

SQL Server характеризується такими особливостями як:

- Продуктивність. SQL Server працює дуже швидко.
- Надійність та безпека. SQL Server надає шифрування даних.
- Простота. З цієї СУБД щодо легко працювати та вести адміністрування.

Центральним аспектом у MS SQL Server, як і будь-який СУБД [23], є база даних. База даних представляє сховище даних, організованих певним способом. Нерідко фізично база даних представляє файл на жорсткому диску, хоча така відповідність необов'язково. Для зберігання та адміністрування баз даних застосовуються системи управління базами даних (database management system) або СУБД (DBMS). І саме MS SQL Server є однією з таких СУБД.

Для організації баз даних MS SQL Server використовує реляційну модель. Ця модель баз даних була розроблена ще в 1970 Едгаром Коддом [23]. А на сьогодні вона фактично є стандартом для організації баз даних.

Реляційна модель передбачає зберігання даних у вигляді таблиць, кожна з яких складається з рядків та стовпців. Кожен рядок зберігає окремий об'єкт, а стовпці розміщуються атрибути цього об'єкта.

Для ідентифікації кожного рядка у межах таблиці застосовується первинний ключ (primary key). Як первинний ключ може виступати один або кілька стовпців. Використовуючи первинний ключ, ми можемо посилатися на певний рядок у таблиці. Відповідно два рядки не можуть мати один і той же первинний ключ.

Через ключі одна таблиця може бути пов'язана з іншою, тобто між двома таблицями можуть бути організовані зв'язки. А сама таблиця може бути представлена у вигляді відношення ("relation").

Для взаємодії з базою даних використовується мова SQL (Structured Query Language). Клієнт (наприклад, зовнішня програма) надсилає запит мовою SQL за допомогою спеціального API. СУБД [23] належним чином інтерпретує та виконує запит, а потім надсилає клієнту результат виконання.

Спочатку мова SQL була розроблена в компанії IBM для системи баз даних, яка називалася System/R. При цьому сама мова називалася SEQUEL (Structured English Query Language). Хоча в результаті ні база даних, ні сама мова не згодом були офіційно опубліковані, за традицією сам термін SQL нерідко вимовляють як "сіквел".

1979 року компанія Relational Software Inc. розробила першу систему управління баз даних, яка називалася Oracle і використовувала мову SQL. У зв'язку з успіхом цього продукту компанія була перейменована на Oracle.

Згодом почали з'являтися інші системи баз даних, які використовували SQL. У результаті в 1989 році Американський Національний Інститут Стандартів (ANSI) кодифікував мову та опублікував її перший стандарт. Після цього стандарт періодично оновлювався та доповнювався. Останнє його оновлення відбулося у 2011 році. Але незважаючи на наявність стандарту нерідко виробники СУБД використовують власні реалізації мови SQL, які трохи відрізняються один від одного.

Виділяються два різновиди мови SQL: PL-SQL та T-SQL. PL-SQL використовується у таких СУБД як Oracle та MySQL. T-SQL (Transact-SQL) застосовується у SQL Server. Саме тому в рамках поточного керівництва розглядатиметься саме T-SQL.

Залежно від завдання, яке виконує команда T-SQL, він може належати до одного з таких типів:

- DDL (Data Definition Language/Мова визначення даних) [22]. До цього типу належать різні команди, які створюють базу даних, таблиці, індекси, процедури, що зберігаються і т.д. Загалом визначають дані.

  Зокрема, до цього типу ми можемо віднести такі команди:

  - CREATE: створює об'єкти бази даних (саму базу даних, таблиці, індекси тощо)
  - ALTER: змінює об'єкти бази даних
  - DROP: видаляє об'єкти бази даних
  - TRUNCATE: видаляє всі дані з таблиць
- DML (Data Manipulation Language/Мова маніпуляції даними) [22]. До цього типу відносять команди з вибору даних, їх оновлення, додавання, видалення - загалом усі команди, з допомогою якими ми можемо управляти даними.

  До цього типу належать такі команди:

  - SELECT: витягує дані з бази даних
  - UPDATE: оновлює дані
  - INSERT: додає нові дані
  - DELETE: видаляє дані
- DCL (Data Control Language/Мова керування доступу до даних) [22]. До цього типу відносять команди, які керують правами доступу до даних. Зокрема, це такі команди:

  - GRANT: надає права на доступ до даних
  - REVOKE: відкликає права на доступ до даних

7 АЛЬТЕРНАТИВНІ СУБД ДО MSSQL SERVER

7.1 Альтернативні СУБД

Альтернативні системи управління базами даних (СУБД) [23] до Microsoft SQL Server, які можуть бути використані для різних типів додатків та потреб:

7.1.1 PostgreSQL

PostgreSQL — це потужна відкрита реляційна система управління базами даних, яка відома своєю стабільністю, надійністю та відповідністю стандартам SQL. Розробка PostgreSQL почалася в 1986 році в Каліфорнійському університеті Берклі [23]. Вона підтримує широкий спектр типів даних, включаючи як вбудовані типи (INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, TIMESTAMP), так і розширювані типи (hstore, JSON, XML). PostgreSQL підтримує складні запити, транзакції, реплікацію та безліч різноманітних індексів. Це СУБД часто вибирають для корпоративних систем, аналітики даних та веб-додатків [23].

Переваги:

- Відкрите програмне забезпечення.
- Підтримка розширених типів даних та індексів.
- Потужна підтримка транзакцій та ACID.
- Широка спільнота та документація.

Недоліки:

- Вища складність налаштування та адміністрування порівняно з деякими іншими СУБД.

Використання:

- Веб-додатки, корпоративні системи, аналітика даних.

7.1.2 MySQL

MySQL — одна з найпопулярніших відкритих реляційних СУБД. Вона часто використовується для веб-додатків та служб, завдяки своїй високій продуктивності, масштабованості та легкості у використанні. MySQL має широку підтримку в спільноті та велику документацію. Однак, вона має деякі обмеження у складних транзакціях порівняно з PostgreSQL. MySQL підходить для таких додатків, як блоги, форуми, електронна комерція та інші веб-додатки [23].

Переваги:

- Відкрите програмне забезпечення.
- Висока продуктивність та масштабованість.
- Широка підтримка та документація.
- Легкість у використанні та налаштуванні.

Недоліки:

- Деякі обмеження в складних транзакціях та функціях порівняно з PostgreSQL.

Використання:

- Веб-додатки, блоги, форуми, електронна комерція.

7.1.3 SQLite

SQLite — це вбудована реляційна СУБД [23], яка зберігає всі дані в одному файлі та не потребує серверної частини. Вона відома своєю легкістю використання та інтеграції, високою продуктивністю для невеликих додатків. SQLite не підтримує багатокористувацький доступ на рівні сервера, що робить її менш придатною для великих або масштабованих додатків. Проте, вона ідеальна для мобільних додатків, невеликих десктопних додатків і прототипів.

Переваги:

- Легкість використання та інтеграції.
- Відсутність потреби в сервері.

- Висока продуктивність для невеликих додатків.

Недоліки:

- Обмежена масштабованість.
- Підтримка одночасного доступу не така потужна, як у серверних СУБД.

Використання:

- Мобільні додатки, невеликі десктопні додатки, прототипи.

7.1.4 Oracle Database

Oracle Database — це потужна комерційна реляційна СУБД [23], відома своєю високою продуктивністю, масштабованістю та розширеними функціональними можливостями. Вона підтримує складні транзакції, аналітику даних і має потужні інструменти для адміністрування та розробки. Однак, Oracle Database є досить дорогою та складною в налаштуванні і адмініструванні. Ця СУБД зазвичай використовується у великих корпоративних системах, фінансових додатках та для аналітики даних.

Переваги:

- Висока продуктивність та масштабованість.
- Потужні інструменти для адміністрування та розробки.
- Підтримка складних транзакцій та аналітики.

Недоліки:

- Висока вартість ліцензій.
- Складність налаштування та адміністрування.

Використання:

- Великі корпоративні системи, аналітика даних, фінансові додатки.

7.1.5 MariaDB

MariaDB — це форк MySQL [23], створений розробниками оригінальної MySQL після її придбання Oracle. Вона забезпечує сумісність з MySQL та додає нові функції і покращення продуктивності. MariaDB є відкритим програмним забезпеченням і має широку підтримку спільноти. Вона підходить для тих же сфер застосування, що й MySQL, включаючи веб-додатки, корпоративні системи та аналітику даних.

Переваги:

- Відкрите програмне забезпечення.
- Сумісність з MySQL.
- Покращена продуктивність та нові функції.

Недоліки:

- Менша спільнота порівняно з MySQL.

Використання:

- Веб-додатки, корпоративні системи, аналітика даних.

7.1.6 MongoDB

MongoDB — це NoSQL СУБД, яка використовує документи у форматі BSON (подібний до JSON) для зберігання даних [23]. Вона добре підходить для роботи з великими обсягами даних та гнучкими структурами. MongoDB забезпечує високу масштабованість і підтримку реплікації та шардінгу. Вона менш придатна для додатків, які потребують складних транзакцій. MongoDB зазвичай використовується для великих даних, веб-додатків, IoT та аналітики.

Переваги:

- Гнучка схема даних.
- Висока масштабованість.
- Підтримка реплікації та шардінгу.

Недоліки:

- Відсутність підтримки транзакцій на рівні SQL.

- Може бути складніше для розробників, які звикли до реляційних баз даних.

Використання:

- Великі дані, веб-додатки, IoT, аналітика.

7.1.7 Microsoft Azure SQL Database

Azure SQL Database — це хмарна реляційна СУБД від Microsoft, побудована на основі SQL Server [23]. Вона забезпечує високу доступність, масштабованість та безпеку для хмарних додатків. Azure SQL Database інтегрується з іншими сервісами Microsoft Azure і забезпечує потужні інструменти для управління та безпеки. Вартість цієї СУБД може бути високою для великих обсягів даних та високої продуктивності. Azure SQL Database часто використовується для хмарних додатків, веб-додатків та аналітики даних.

Переваги:

- Повна інтеграція з Microsoft Azure.
- Висока доступність та масштабованість.
- Потужні інструменти для управління та безпеки.

Недоліки:

- Вартість може бути високою для великих обсягів даних та високої продуктивності.

Використання:

- Хмарні додатки, веб-додатки, аналітика даних.

Висновок: Вибір СУБД залежить від ваших конкретних потреб, включаючи вимоги до продуктивності, масштабованості, гнучкості даних, бюджету та навичок вашої команди. Кожна з вищезазначених СУБД має свої сильні та слабкі сторони, тому важливо оцінити їх відповідно до ваших потреб перед прийняттям рішення.

8 ENTITY FRAMEWORK CORE

8.1 Що таке Entity Framework Core

Entity Framework представляє ORM-технологію (object-relational mapping – відображення даних на реальні об'єкти) від компанії Microsoft для доступу до даних [21]. Entity Framework Core дозволяє абстрагуватися від самої бази даних та її таблиць та працювати з даними як з об'єктами класом незалежно від типу сховища. Якщо фізично ми оперуємо таблицями, індексами, первинними та зовнішніми ключами, але на концептуальному рівні, який нам пропонує Entity Framework, ми вже працюємо з об'єктами.

Як технологія доступу до даних Entity Framework Core працює поверх платформи .NET і тому може використовуватися на різних платформах стека .NET. Це і стандартні платформи типу Windows Forms, консольні програми, WPF, UWP та ASP.NET Core. При цьому кросплатформова природа EF Core дозволяє задіяти її не тільки на Windows, але і на Linux і Mac OS X.

Оскільки Entity Framework Core працює на основі платформи .NET, він розвивається разом з даною платформою. Поточну версію EF Core - 8.0 було випущено в листопаді 2023 року разом з .NET 8. І технологія продовжує розвиватися.

Entity Framework Core підтримує багато різних систем баз даних. Таким чином, ми можемо через EF Core працювати з будь-яким СУБД, якщо для неї є потрібний провайдер. За замовчуванням на даний момент Microsoft надає ряд вбудованих провайдерів: для роботи з MS SQL Server, SQLite, PostgreSQL. Також є провайдери від сторонніх постачальників, наприклад MySQL.

Варто зазначити, що Entity Framework Core розвивається як opensourse-проект, усі файли якого можна знайти в репозиторії на github за адресою https://github.com/dotnet/efcore.

Також варто відзначити, що EF Core надає універсальний API для роботи з даними. І якщо, наприклад, ми вирішимо змінити цільову СУБД, то основні

зміни в проекті стосуватимуться насамперед конфігурації та налаштування підключення до відповідних провайдерів. А код, який безпосередньо працює з даними, отримує дані, додає їх у базу даних тощо, залишиться тим самим.

Центральною концепцією Entity Framework є поняття сутності чи entity. Сутність визначає набір даних, які пов'язані з певним об'єктом. Тому дана технологія передбачає роботу не з таблицями, а з об'єктами та їх колекціями.

Будь-яка сутність, як і будь-який об'єкт із реального світу, має ряд властивостей. Наприклад, якщо сутність описує людину, ми можемо виділити такі властивості, як ім'я, прізвище, зростання, вік. Властивості необов'язково являють собою прості дані типу int або string, але можуть також представляти і більш комплексні типи даних. І в кожної сутності може бути одна або кілька властивостей, які відрізнятимуть цю сутність від інших і унікально визначатимуть цю сутність. Такі властивості називають ключами.

При цьому сутності можуть бути пов'язані асоціативним зв'язком одним, багатьом, подібно до того, як у реальній базі даних відбувається зв'язок через зовнішні ключі.

Відмінною рисою Entity Framework Core як технології ORM є використання запитів LINQ для вибірки даних з БД. За допомогою LINQ ми можемо створювати різні запити на вибір об'єктів, у тому числі пов'язаних різними асоціативними зв'язками. А Entity Framework при виконанні запиту транслює вирази LINQ у вирази, зрозумілі для конкретної СУБД (зазвичай у вирази SQL).

Основна функціональність Entity Framework Core зосереджена у таких пакетах [21]:

- Microsoft.EntityFrameworkCore:основний пакет EF Core
- Microsoft.EntityFrameworkCore.SqlServer: представляє функціональність провайдера для Microsoft SQL Server та SQL Azure
- Microsoft.EntityFrameworkCore.SqlServer.NetTopologySuite: надає підтримку географічних типів (spatial types) для SQL Server

- Microsoft.EntityFrameworkCore.Sqlite: представляє функціональність провайдера для SQLite і включає нативні бінарні файли для движка бази даних
- Microsoft.EntityFrameworkCore.Sqlite.Core: представляє функціональність провайдера для SQLite, але на відміну від попереднього пакета не містить нативні бінарні файли для движка бази даних
- Microsoft.EntityFrameworkCore.Sqlite.NetTopologySuite: надає підтримку географічних типів (spatial types) для SQLite
- Microsoft.EntityFrameworkCore.Cosmos: представляє функціональність провайдера для Azure Cosmos DB
- Microsoft.EntityFrameworkCore.InMemory: представляє функціональність провайдера бази даних у пам'яті
- Microsoft.EntityFrameworkCore.Tools: містить команди EF Core PowerShell для Visual Studio Package Manager Console; застосовується в Visual Studio для міграцій та генерації класів за готовою базою даних
- Microsoft.EntityFrameworkCore.Design: містить допоміжні компоненти EF Core, які застосовуються в процесі розробки
- Microsoft.EntityFrameworkCore.Proxies: зберігає функціональність для так званого "лінивого завантаження" (lazy-loading) та проксі охолодження змін
- Microsoft.EntityFrameworkCore.Abstractions: містить набір абстракцій EF Core, які не залежать від конкретної СУБД
- Microsoft.EntityFrameworkCore.Relational: зберігає компоненти EF Core для провайдерів реляційних СУБД
- Microsoft.EntityFrameworkCore.Analyzers: містить функціонал аналізаторів C# для EF Core.

Платформу Entity Framework Core можна застосовувати в різних технологіях стека .NET - консольних програмах, програмах на WinForms, WPF, UWP, веб-додатки ASP.NET і так далі. В даному випадку ми

розглядатимемо базові моменти платформи на прикладі консольних додатків, як найпростіших і не містять жодного зайвого коду.

Крім того, EF Core може працювати з різними системами баз даних [24]. Тут ми будемо розглядати загальні можливості на прикладі бази даних SQLite як найпростішої і зручнішої СУБД. І оскільки Entity Framework дозволяє писати універсальний код для підключення до різних СУБД, цей код можна буде застосовувати і до інших СУБД. Однак, згодом також торкнемося роботи з іншими СУБД у тих аспектах, де є відмінності.

9 АЛЬТЕРНАТИВНІ ORM ТЕХНОЛОГІЇ ENTITY FRAMEWORK

9.1 Альтернативні ORM технології

У .NET існує декілька альтернатив Entity Framework для доступу до даних [24]:

9.1.1 Dapper

Dapper — це мікро ORM, розроблений командою з Stack Overflow. Він працює поверх ADO.NET і забезпечує простий інтерфейс для виконання SQL-запитів та мапінгу результатів на об'єкти C# [24]. Dapper є надзвичайно швидким і легким, тому що він не додає значної обробки поверх ADO.NET.

Переваги:

- Висока продуктивність.
- Простота використання.
- Легкий для розгортання та навчання.
- Повний контроль над SQL-запитами.

Недоліки:

- Відсутність автоматичного мапінгу об'єктів на таблиці.
- Не підтримує складні об'єктні відносини, такі як колекції чи спадкування.
- Приклад використання:

```
using (var connection = new SqlConnection(connectionString))
{
    var users = connection.Query<User>("SELECT * FROM
Users").ToList();
}
```

Лістинг 9.1.1 – Приклад використання Dapper для виконання SQL-запиту у C#

9.1.2 NHibernate

NHibernate — це повнофункціональний ORM для .NET, який надає потужні можливості мапінгу об'єктів на реляційні бази даних [24]. NHibernate підтримує складні об'єктні відносини та бізнес-логіку, забезпечуючи багаті можливості для роботи з базами даних.

Переваги:

- Підтримка складних об'єктних відносин.
- Можливість конфігурації через XML або Fluent API.
- Підтримка кешування другого рівня.

Недоліки:

- Висока складність налаштування.
- Може бути менш продуктивним в порівнянні з Dapper для простих запитів.
- Приклад використання:

```
var sessionFactory = new
NHibernate.Cfg.Configuration().Configure().BuildSessionFactory()
;
using (var session = sessionFactory.OpenSession())
{
    using (var transaction = session.BeginTransaction())
    {
        var users = session.Query<User>().ToList();
        transaction.Commit();
    }
}
```

Лістинг 9.1.2 – Приклад використання NHibernate для отримання даних з бази даних у C#

9.1.3 ServiceStack OrmLite

OrmLite від ServiceStack — це легковажний ORM, який забезпечує простий спосіб роботи з базою даних без складної конфігурації [24]. OrmLite використовує чисті SQL-запити і забезпечує мапінг результатів на об'єкти C#.

Переваги:

- Простота використання.
- Підтримка чистого SQL.
- Легка інтеграція з іншими компонентами ServiceStack.

Недоліки:

- Менше можливостей порівняно з повнофункціональними ORM.
- Приклад використання:

```
using (var db = new OrmLiteConnectionFactory(connectionString,
SqlServerDialect.Provider).Open())
{
    var users = db.Select<User>();
}
```

Лістинг 9.1.3 – Приклад використання ServiceStack OrmLite для отримання даних з бази даних у C#

9.1.4 LINQ to SQL

LINQ to SQL — це вбудована в .NET бібліотека, яка дозволяє використовувати LINQ-запити для доступу до SQL Server [24]. Вона є легковажною альтернативою Entity Framework і забезпечує простий спосіб мапінгу об'єктів на базу даних.

Переваги:

- Простота використання.
- Підтримка LINQ-запитів.
- Безпосередня інтеграція з .NET.

Недоліки:

- Обмежена підтримка складних об'єктних відносин.
- Працює тільки з SQL Server.
- Приклад використання:

```
using (var db = new DataContext(connectionString))
{
    var users = db.GetTable<User>().ToList();
}
```

Лістинг 9.1.4 – Приклад використання LINQ to SQL для отримання даних з бази даних у C#

9.1.5 Мікро ORM бібліотеки

Існують інші **мікро ORM бібліотеки**, такі як PetaPoco та Massive, які пропонують різні рівні функціональності та продуктивності [24]. Вони орієнтовані на простоту використання та високу швидкість виконання операцій.

Переваги:

- Висока продуктивність.
- Простота та легкість.
- Підтримка різних баз даних.

Недоліки:

- Обмежені можливості порівняно з повнофункціональними ORM.

Як вибрати альтернативу?

- Вибір відповідної альтернативи залежить від конкретних вимог:
- Продуктивність: Якщо потрібна висока продуктивність і контроль над SQL-запитами, Dapper може бути кращим вибором.
- Функціональність: Якщо потрібні складні об'єктні відносини та потужні можливості мапінгу, NHibernate може бути відповідним варіантом.
- Простота: Якщо потрібна простота та зручність, OrmLite або LINQ to SQL можуть бути хорошими варіантами.

10 РЕАЛІЗАЦІЯ ЕКСПЕРТНОЇ СИСТЕМИ

10.1 Реєстрація користувача:

При першому запуску програми користувач бачить форму реєстрації, де необхідно ввести свої дані: ім'я (до 20 символів), прізвище (до 30 символів), по-батькові (до 30 символів), вік (від 16 до 120 років), та електронну пошту (до 100 символів) (рис. 10.1.1). Усі поля обов'язкові для заповнення (див. ДОДАТОК В).

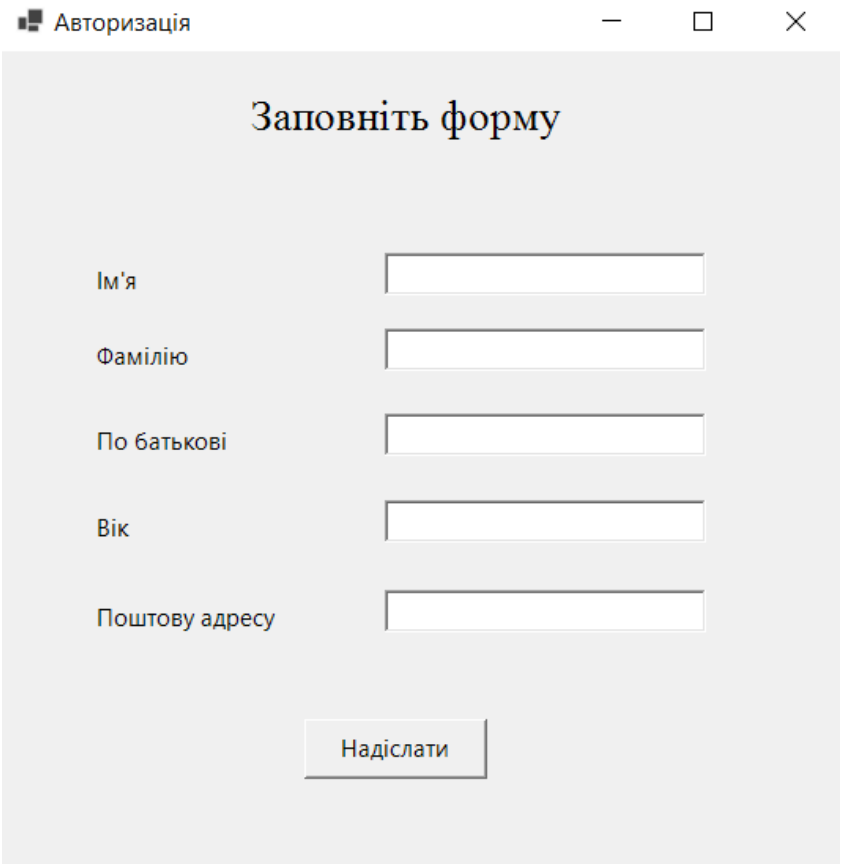

Рис. 10.1.1 – Реєстрація

При натисканні кнопки "Надіслати" дані (якщо Вони введені правильно) через механізм Entity Framework Core заповнюються в колонки таблиці Users (рис. 10.1.2) в базу даних, через властивості у відповідному класі (моделі) у коді C# (див. рис. 10.1.3).

Цей процес забезпечує цілісність та узгодженість даних.

Рис. 10.1.2 – Таблиця Users у MSSql

Цей процес включає в себе створення нового об’єкта користувача з введеними даними, додавання його до контексту та збереження змін. Entity Framework Core автоматично генерує SQL-запити для вставки даних у таблицю Users.

Крім того, при збереженні властивостей у базу даних здійснюється додаткова валідація на стороні бази, що забезпечує цілісність та узгодженість даних. Це включає перевірку унікальності електронної пошти, коректності формату введених значень та дотримання встановлених обмежень.

Таким чином, інтеграція валідації на стороні клієнта та бази даних гарантує, що всі введені дані є коректними та відповідають вимогам системи, що покращує загальну надійність та стабільність програми.

Усі текстбокси, що знаходяться на формі – обов’язкові Для заповнення, про що свідчать атрибути “[Required]” на властивостях моделі у коді C#.

Процес валідації на стороні клієнта здійснюється за допомогою відповідних атрибутів і методів, що дозволяють перевірити правильність введених даних ще до їх відправлення на сервер. Наприклад, атрибут «[EmailAddress]» використовується для перевірки правильності формату електронної пошти, а «[Range(16, 120)]» – для перевірки вікового діапазону. Після проходження всіх перевірок, дані надсилаються на сервер, де відбувається їх обробка і збереження у базі.

Крім цього, система включає механізми обробки помилок та зворотного зв’язку для користувача. Якщо під час збереження даних виникає помилка, користувач отримає відповідне повідомлення з поясненням причини помилки

та порадами щодо її виправлення. Це забезпечує більш зручний та інтуїтивно зрозумілий процес взаємодії з програмою.

Отже, комплексний підхід до валідації та обробки даних, що включає перевірки на стороні клієнта і сервера, значно підвищує ефективність і надійність інформаційної системи, зменшуючи кількість помилок та покращуючи користувацький досвід (див. ДОДАТОК К).

```
7 references
public class User
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    2 references
    public int Id { get; set; }

    [Required]
    [MaxLength(20)]
    [Column("Ім'я")]
    4 references
    public string Name { get; set; } = null!;

    [Required]
    [MaxLength(30)]
    [Column("Фамілія")]
    4 references
    public string Surname { get; set; } = null!;

    [Required]
    [MaxLength(30)]
    [Column("По батькові")]
    4 references
    public string MiddleName { get; set; } = null!;

    [Required]
    [Range(16, 120)]
    [Column("Вік")]
    4 references
    public int Age { get; set; }

    [Required]
    [MaxLength(100)]
    4 references
    public string Email { get; set; } = null!;

    [Column("Обрана дисципліна")]
    4 references
    public string? CurrentDiscipline { get; set; }
}
```

Рис. 10.1.3 – Модель у коді C#, яка буде представлена у якості таблиці Users

Для валідації параметрів, що надходять від клієнта, застосовується підхід Data Annotation, який забезпечує перевірку правильності введених даних безпосередньо в моделі.

Додатково здійснюється валідація на стороні бази даних для забезпечення цілісності та узгодженості даних. Це включає використання обмежень на рівні бази даних, таких як первинні та зовнішні ключі, унікальні індекси та перевірки, що гарантують правильність і взаємозв'язок наданих параметрів. Такий підхід забезпечує подвійний рівень валідації: на рівні програми, що дозволяє зменшити навантаження на базу даних, і на рівні самої бази даних, що гарантує цілісність даних навіть при прямому доступі до неї.

10.2 Головна форма програми

Після реєстрації нас зустрічають усі існуючі дисципліни (рис. 10.2.4), натиснувши на які можна подивитися на усі "вміння" які матиме студент після закінчення навчання (див. рис. 10.4.7). Для кожної дисципліни надається детальний опис навичок і знань, які студент зможе опанувати. Це допомагає студентам усвідомлено обирати дисципліни, які найбільше відповідають їхнім інтересам і майбутнім професійним планам (див. ДОДАТОК Б).

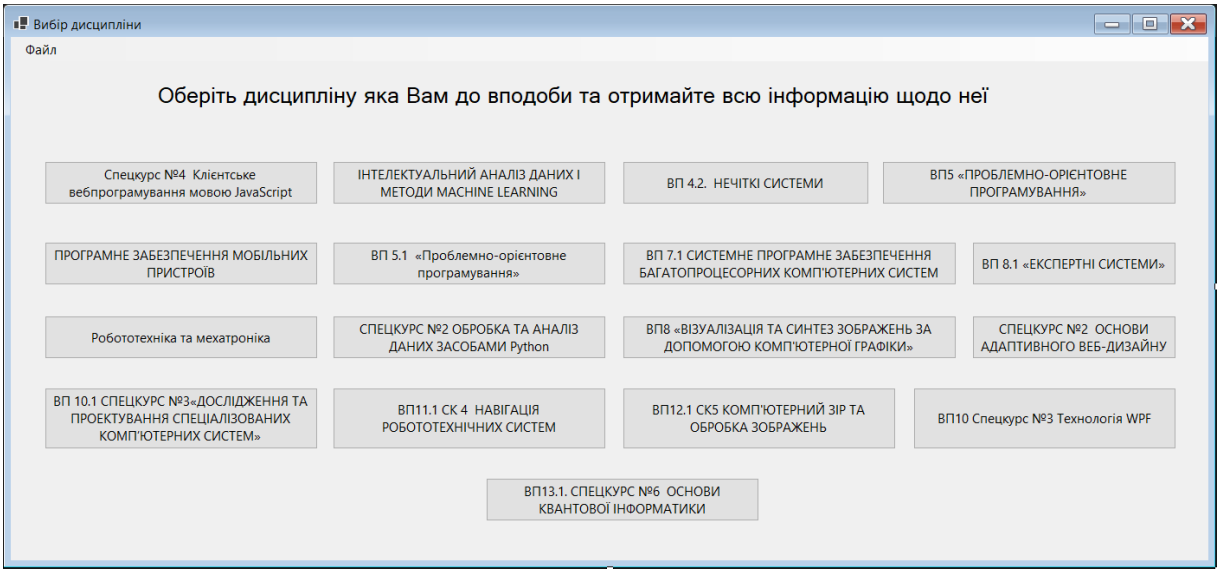

Рис. 10.2.4 – Форма з відображенням усіх дисциплін

Інтерфейс системи розроблений таким чином, щоб забезпечити зручну навігацію та доступ до необхідної інформації. Користувачі можуть сортувати дисципліни за різними критеріями, наприклад, за популярністю, рівнем складності або кількістю отриманих умінь. Це сприяє створенню персоналізованого навчального плану, який відповідає індивідуальним потребам і цілям кожного студента.

10.3 Робота з базою даних

Кожній дисципліні відповідає своя таблиця у базі даних. Дані з комірок якої розміщуються у labels на формі для кожного спецкурсу (рис. 10.3.5).

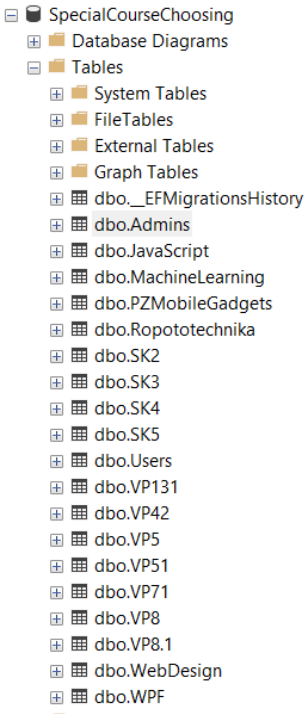

Рис. 10.3.5 – Усі таблиці існуючої бази даних

У цьому проекті я використав підхід Entity Framework – Data Base First. Цей підхід дозволяє створювати класи моделей на основі таблиць бази даних, що створюються після застосування міграцій (див. рис. 10.3.6). Після

виконання міграцій у проекті створюється папка «Migrations», у якій міститься інформація про зміни, що відбулися в базі даних.

Підхід Data Base First є особливо корисним, коли вже існує наявна база даних, яку необхідно інтегрувати в новий або існуючий проект. Використовуючи цей підхід, ми автоматично генеруємо класи моделей, що відображають структуру таблиць бази даних, зберігаючи всі зв’язки та обмеження між таблицями. Це значно зменшує кількість ручної роботи та забезпечує точну відповідність між базою даних і моделями програми.

Процес створення та застосування міграцій забезпечує версійний контроль бази даних, що дозволяє легко відслідковувати історію змін, повертатися до попередніх версій або інтегрувати нові зміни в командній розробці. Це особливо важливо у великих проектах, де база даних постійно розвивається та змінюється.

Крім того, використання міграцій дозволяє зберігати структуру бази даних у контрольованому середовищі, що підвищує надійність та стабільність системи. Всі зміни, внесені через міграції, зберігаються в історії проекту, що полегшує аудит та обслуговування бази даних.

Таким чином, підхід Entity Framework – Data Base First забезпечує високу ефективність розробки, зменшуючи кількість помилок і забезпечуючи синхронізацію між базою даних і кодом програми. Це дозволяє зосередитися на логіці програми та бізнес-процесах, замість ручного управління структурою бази даних.

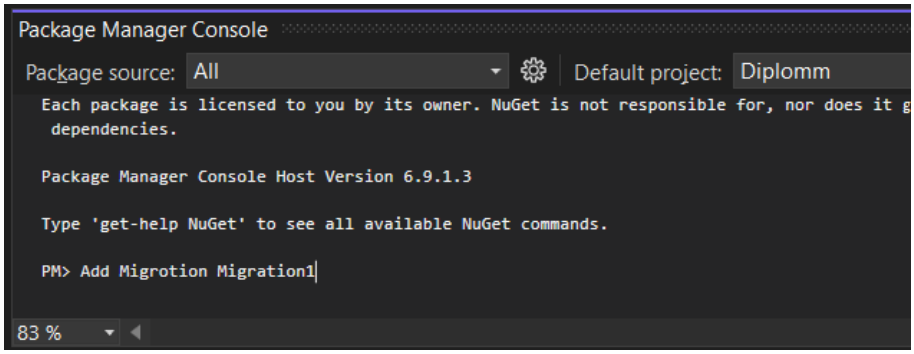

Рис. 10.3.6 – Команда, яка виконує міграцію

Кожна міграція генерує відповідний файл, який містить два основні методи: **Up** та **Down**. Метод **Up** відповідає за внесення змін до бази даних (наприклад, створення нових таблиць або додавання колонок), тоді як метод **Down** забезпечує можливість відкату цих змін (наприклад, видалення створених таблиць або колонок).

Окрім цього, використання міграцій забезпечує версійний контроль бази даних, що дозволяє легко відслідковувати зміни, повертатися до попередніх версій або інтегрувати нові зміни в командній розробці. Такий підхід значно спрощує процес оновлення структури бази даних та підтримує її в актуальному стані відповідно до вимог проекту.

Додатково, з використанням Entity Framework – Data Base First, ми маємо можливість автоматично синхронізувати наші моделі з актуальною структурою бази даних, що знижує ймовірність помилок, пов’язаних з невідповідністю моделі та бази даних. Це особливо корисно при роботі в великих командах або при частих змінах в структурі бази даних.

Також, використання міграцій дозволяє забезпечити безпеку бази даних, оскільки всі зміни вносяться за допомогою контрольованих та автоматизованих процедур, що мінімізує ризик помилок та випадкового втручання в структуру даних.

10.4 Обрання користувачем дисципліну

Після того як користувач натисне кнопку “Обрати дисципліну” (рис. 10.4.7) у його колонку таблиці базі даних, у стовпець «Вибрані дисципліни» (див.рис. 10.4.8) буде записана назва тієї дисципліни, яку він обере. Цей процес автоматизовано і виконується негайно, що забезпечує актуальність даних у реальному часі.

Таким чином, система забезпечує не тільки зручність і швидкість внесення виборів, але й гарантує можливість їх коригування, що сприяє більш

точному відображенню навчальних пріоритетів кожного студента. Це підвищує загальну ефективність процесу управління навчальними дисциплінами і покращує взаємодію між студентами та адміністрацією навчального закладу.

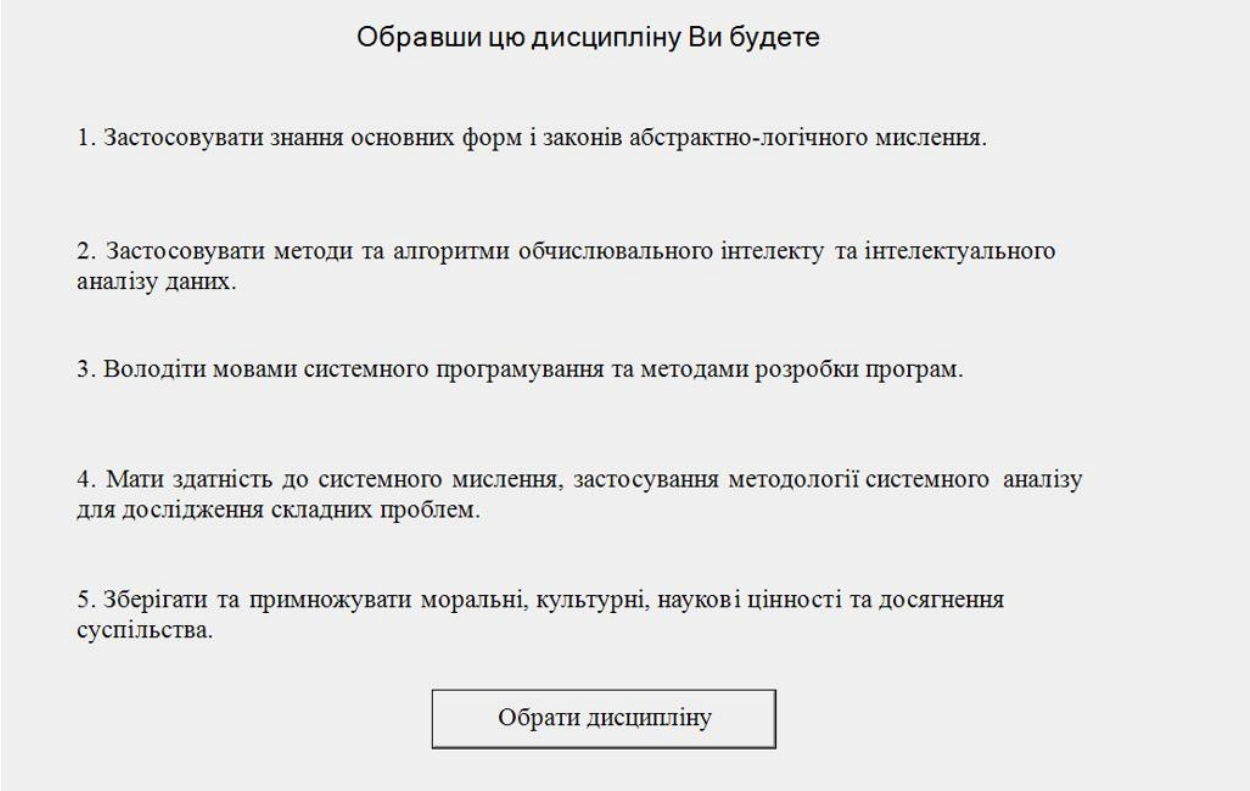

Рис. 10.4.7 – Одна з форм дисципліни

Якщо користувач нічого не вибере, у його стовпці «Обрана дисципліна» у базі даних буде null (порожнє значення), але всі дані, що стосуються користувача (ім’я, прізвище, по батькові, вік, пошта) будуть записані (рис. 10.4.8).

	50	Олексій	Шевченко	46	shevch@gm...	*NULL*	Григорович

Рис. 10.4.8 – Колонки тестового користувача

Також користувач може перевибирати дисципліну (поки він знаходиться в відкритому додатку), в таблицю бази даних буде записаний останній

вибраний результат (Вибрана дисципліна). Якщо користувач (рис. 10.4.9) поспішив, і захоче змінити ту дисципліну, яку він обрав але вже закрив додаток, для цього йому потрібно звернутися до адміністратора (див.рис. 10.5.12).

Рис. 10.4.9 – Користувач з обраною дисципліною

10.5 Форма для адміністраторів

Для того щоб отримати інформацію про користувачів, які уже пройшли опитування ми можемо перейти до головного меню у лівому верхньому куті програми Файл => Детально (рис. 10.5.10).

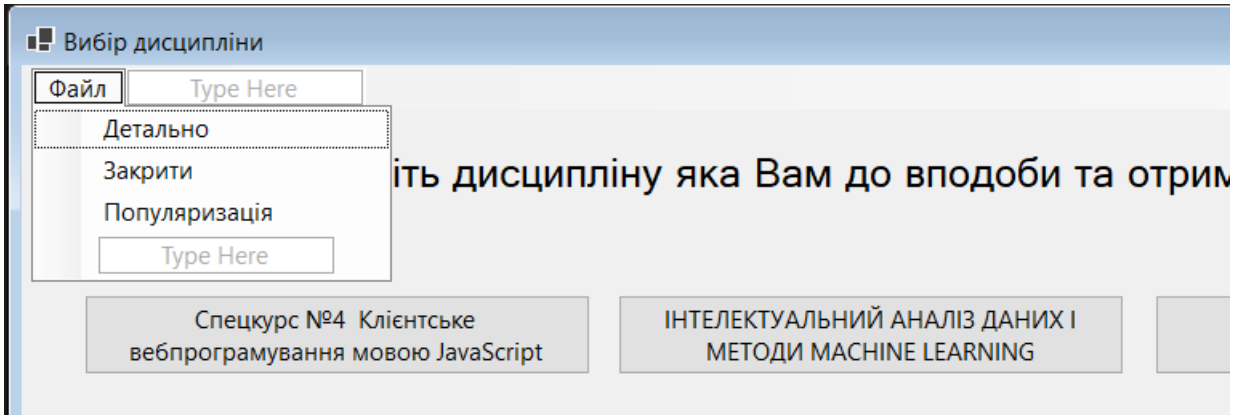

Рис. 10.5.10 – Детально

Після цього знову відкриється форма для авторизації (див.рис. 10.5.11), пройти яку можуть лише користувачі з певними правами доступу. Ця додаткова авторизація забезпечує захист конфіденційної інформації та гарантує, що доступ до чутливих даних мають тільки авторизовані особи, наприклад адміністратори або викладачі.

Завдяки цій функції, адміністратори можуть аналізувати зібрані дані, генерувати звіти та приймати обґрунтовані рішення щодо покращення

освітнього процесу. Водночас, викладачі можуть отримувати доступ до інформації про своїх студентів, що дозволяє їм краще розуміти потреби та вподобання студентів і відповідно коригувати навчальний процес.

Загалом, цей процес авторизації та доступу до детальної інформації сприяє підвищенню безпеки даних і забезпечує належне управління інформацією в межах освітньої системи (див. ДОДАТОК Г).

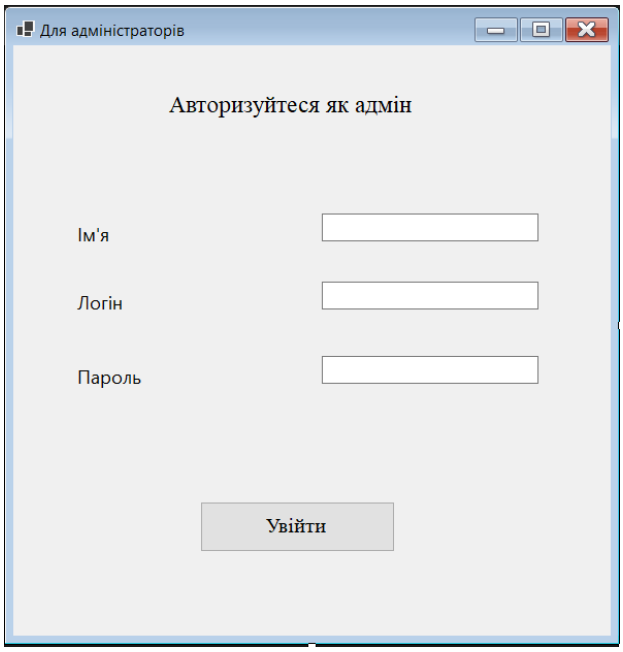

Рис. 10.5.11 – Форма для адмінів

Якщо уважно придивитися до рисунка 10.3.5, можна побачити ще одну таблицю «Admins», яка містить дані лише про користувачів, які мають роль адміністратора. Ця таблиця (рис. 10.5.12) зберігає інформацію про осіб з підвищеними правами доступу, що дозволяє їм виконувати адміністративні функції в системі, такі як редагування даних користувачів, управління дисциплінами та контроль за системою в цілому (див. ДОДАТОК Ж).

	Id	Name	Login	Password
▶	1	admin1234	admin@gm...	12345
*	NULL	NULL	NULL	NULL

Рис. 10.5.12 – Таблиця для адмінів

У даному випадку в таблиці знаходиться лише один тестовий об’єкт. Але тут можуть знаходитися дані про справжніх людей таких як викладачі, ректори, ментори, завучі, декани, методисти і тд.

Після введення даних адміністратора у відповідні поля відкриється форма (рис. 10.5.13), на якій відобразиться таблиця з інформацією про всіх здобувачів вищої освіти, які обрали дисципліну. Ця таблиця дозволяє адміністраторам та керувати даними здобувачів вищої освіти, забезпечуючи адміністрування та підтримку навчального процесу (див. ДОДАТОК Д).

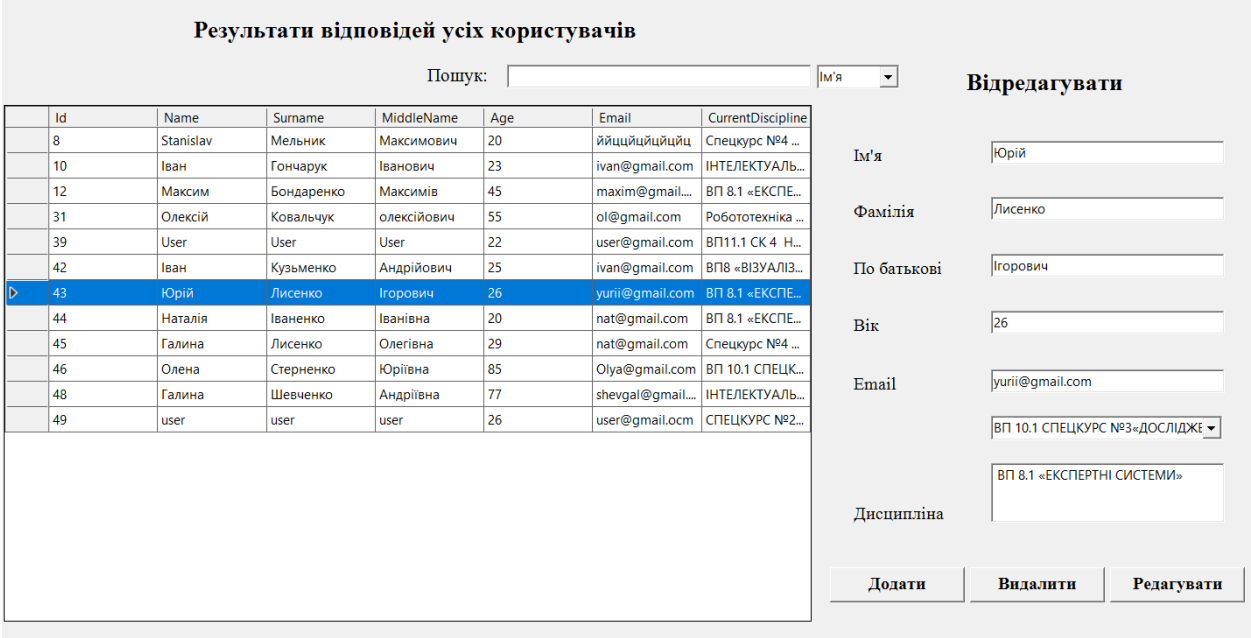

Рис. 10.5.13 – Дані про здобувачів вищої освіти, до яких мають доступ тільки адміни

Як видно з рисунка, адміністратори цієї програми можуть не тільки подивитися хто які дисципліни обрав, але й відредагувати дані стосовно студента. Наприклад, якщо студент при реєстрації ввів неправильно своє ім’я, прізвище, по батькові, вік, email або обрав не ту дисципліну яку хотів, чи передумав, саме адміністратор може внести остаточні зміни щодо студента.

Адміністратор має можливість додавати нових здобувачів вищої освіти, видаляти існуючих або редагувати їхні дані. Для цього в інтерфейсі форми є

відповідні кнопки, які дозволяють виконувати всі необхідні маніпуляції з даними.

Усі зміни, внесені адміністратором, негайно відображаються у відповідних колонках таблиці бази даних.

Щоб адміністратор міг почати роботу з даними студента, йому потрібно обрати потрібного користувача з таблиці, що знаходиться на формі. Усі властивості, які стосуються вибраного студента, відобразяться у текстових полях під лейблом «Відредагувати». Це забезпечує зручне та швидке редагування даних студента.

Ось приклад додання нового користувача адміном (рис. 10.5.14), (рис. 10.5.15):

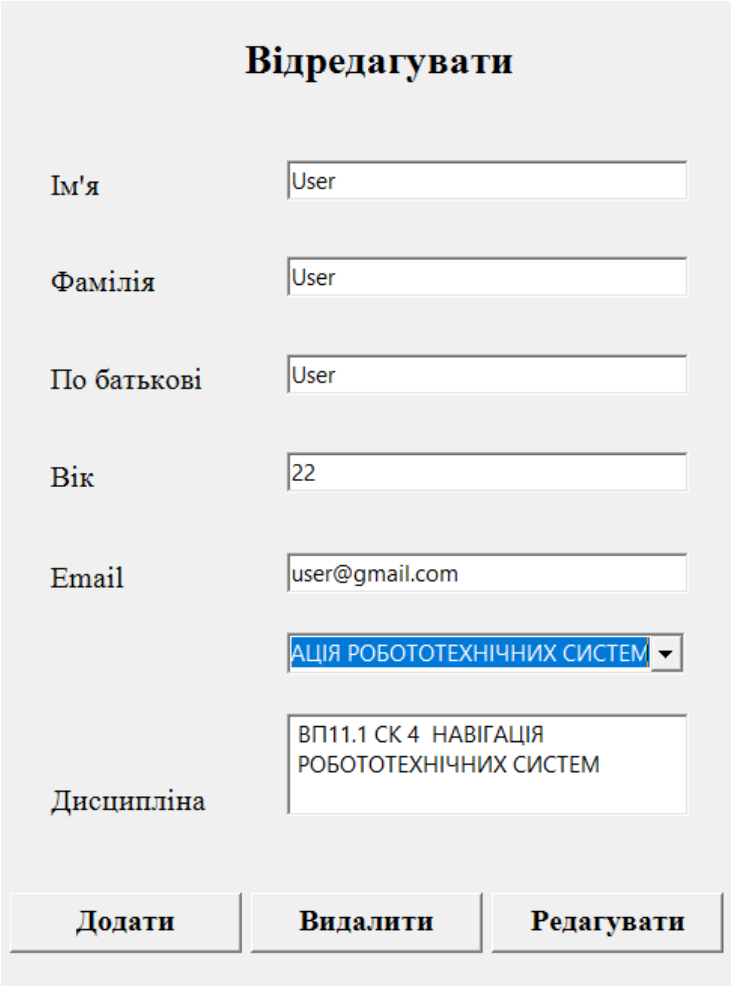

Рис. 10.5.14 – Тестове додання користувача

Список доступних дисциплін для вибору

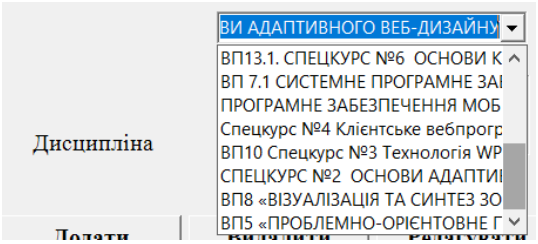

Рис. 10.5.15 – Колекція усіх доступних дисциплін

Після цього як адміністратор натисне кнопку “Додати” йому висвітиться повідомлення про успішне створення користувача (рис. 10.5.16), якщо дані були введено коректно. Якщо же дані були введено неправильно, ми також побачимо відповідне повідомлення (рис. 10.5.17).

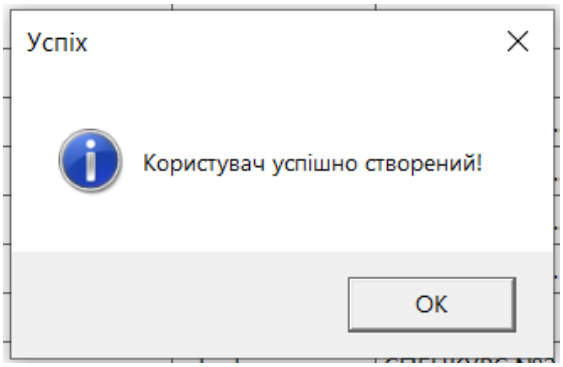

Рис. 10.5.16 – Користувач успішно створений

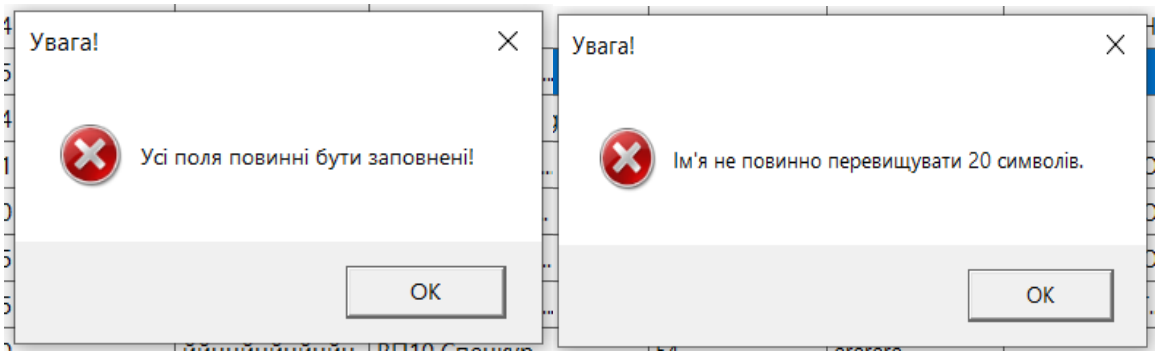

Рис. 10.5.17 – Попередження

Такі повідомлення будуть відображатися при будь-якій зміні об’єктів у таблиці, що дозволяє користувачам та адміністраторам бути завжди в курсі останніх подій та змін. Це сприяє покращенню комунікації та співпраці між усіма учасниками системи та забезпечує швидке реагування на поточні потреби та вимоги.

Таблиця після створення користувача (рис. 10.5.18):

Рис. 10.5.18 – Вигляд таблиці після створення користувача

Як було зазначено раніше, всі зміни, внесені адміністратором, негайно відображаються в таблиці бази даних. Це забезпечує актуальність і точність даних, дозволяючи адміністраторам оперативно керувати інформацією про здобувачів вищої освіти та їхні вибрані дисципліни. Завдяки цьому підходу, система завжди містить актуальні дані, що є критично важливим для ефективного адміністрування навчального процесу (див.рис. 10.5.19).

Крім того, ця система дозволяє адміністраторам вносити зміни та відстежувати активність користувачів, що сприяє підвищенню рівня безпеки та контролю над даними. Такий підхід дозволяє оперативно реагувати на потреби користувачів та забезпечує ефективне управління системою.

Id	Ім'я	Фамілія	Вік	Email	Обрана дисципліна	По батькові
8	Stanislav	Мельник	20	ййццйцйцй...	Спецкурс №4 Клієнтське вебпрограмування мовою JavaScript	Максимович
10	Іван	Гончарук	23	ivan@gmail...	ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ І МЕТОДИ MACHINE LEA...	Іванович
12	Максим	Бондаренко	45	maxim@gm...	ВП 8.1 «ЕКСПЕРТНІ СИСТЕМИ»	Максимів
31	Олексій	Ковальчук	55	ol@gmail.c...	Робототехніка та мехатроніка	олексійович
39	User	User	22	user@gmail...	ВП11.1 СК 4 НАВІГАЦІЯ РОБОТОТЕХНІЧНИХ СИСТЕМ	User
42	Іван	Кузьменко	25	ivan@gmail...	ВП8 «ВІЗУАЛІЗАЦІЯ ТА СИНТЕЗ ЗОБРАЖЕНЬ ЗА ДОПОМОГО...	Андрійович
43	Юрій	Лисенко	26	yurii@gmail...	ВП 8.1 «ЕКСПЕРТНІ СИСТЕМИ»	Ігорович
44	Наталія	Іваненко	20	nat@gmail....	ВП 8.1 «ЕКСПЕРТНІ СИСТЕМИ»	Іванівна
45	Галина	Лисенко	29	nat@gmail....	Спецкурс №4 Клієнтське вебпрограмування мовою JavaScript	Олегівна
46	Олена	Стерненко	85	Olya@gmai...	ВП 10.1 СПЕЦКУРС №3«ДОСЛІДЖЕННЯ ТА ПРОЕКТУВАННЯ С...	Юріївна
48	Галина	Шевченко	77	shevgal@g...	ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ І МЕТОДИ MACHINE LEA...	Андріївна
49	User	User	22	user@gmail...	ВП11.1 СК 4 НАВІГАЦІЯ РОБОТОТЕХНІЧНИХ СИСТЕМ	User
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Рис. 10.5.19 – Таблиця у базі даних після додання користувача

10.6 Додаткові властивості програми

Також у головному меню в лівому верхньому куті є ще одна вкладка «Файл => Популяризація» (рис. 10.6.20), доступ до якої відкритий для всіх користувачів. У цій вкладці розміщується графік (див.рис. 10.6.21), який наочно демонструє популярність різних дисциплін серед здобувачів вищої освіти.

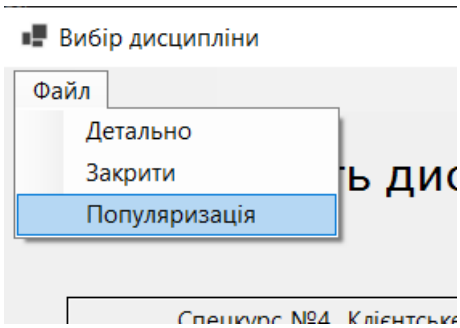

Рис. 10.6.20 – Головне меню

Графік допомагає користувачам зрозуміти, які курси є найбільш затребуваними, що може сприяти прийняттю більш обґрунтованих рішень щодо вибору дисциплін. Така функціональність робить систему більш інформативною і корисною для всіх учасників навчального процесу (див. ДОДАТОК Е).

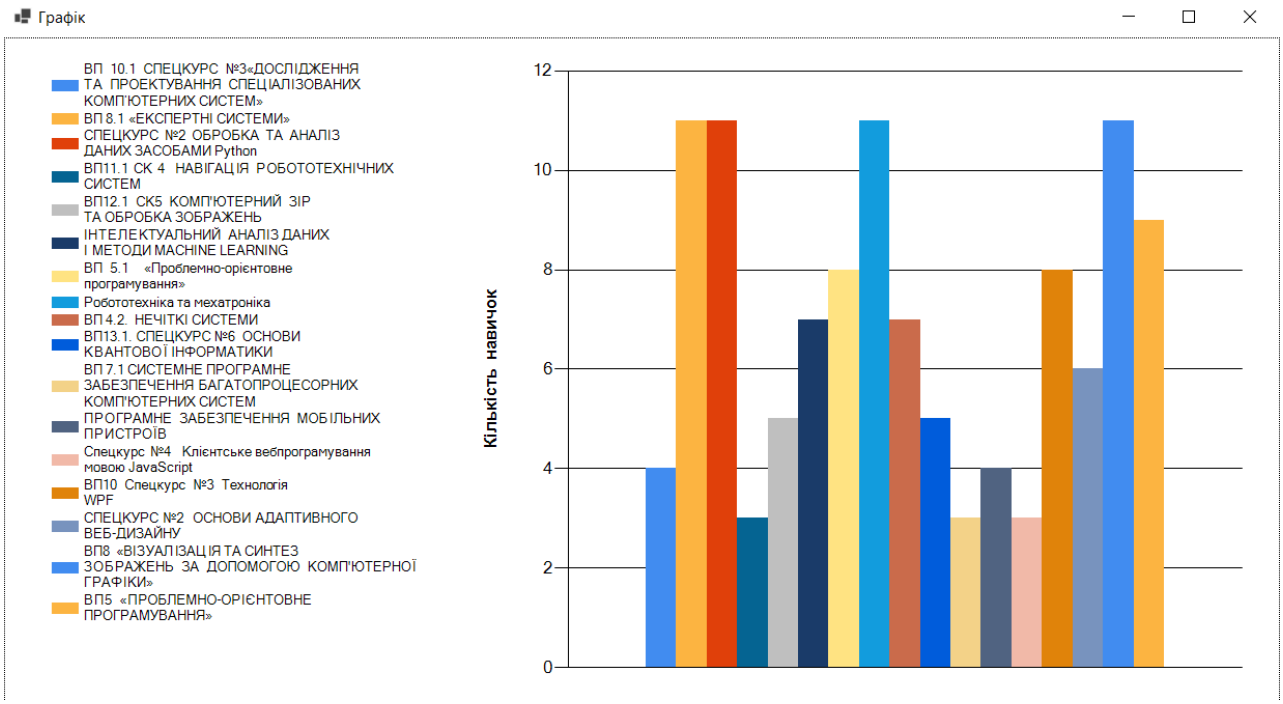

Рис. 10.6.21 – Графік

Графік демонструє розподіл дисциплін в залежності від кількості навичок, що студент отримує в результаті їх вивчення. Це дає можливість користувачам бачити, які дисципліни надають найбільше знань та навичок, а також визначити їхню популярність серед студентської аудиторії. Такий графік допомагає керівництву навчального закладу приймати рішення щодо вдосконалення навчальної програми та вибору найефективніших курсів для здобувачів вищої освіти.

Висновок за розділом

У цьому розділі ми детально розглянули функціональні можливості системи та їхні взаємозв’язки. Зокрема, ми розглянули процес реєстрації користувачів, їхню можливість вибору дисциплін, а також адміністративні функції, доступні адміністраторам. Пояснили, як система забезпечує взаємодію з базою даних та оперативне відображення змін. Додатково висвітлили роль графіка у візуалізації популярності дисциплін серед здобувачів вищої освіти.

Загальний висновок полягає в тому, що система надає зручний та ефективний інструмент для управління та моніторингу навчального процесу. Шляхом використання різноманітних функцій і можливостей користувачі можуть ефективно взаємодіяти з системою та досягати своїх навчальних цілей.

11 СЕРВЕРНА ЧАСТИНА

11.1 Архітектура проекту

Використання монолітної архітектури в проекті

Монолітна архітектура - це архітектурний підхід, в якому всі компоненти програмного забезпечення розглядаються як одна велика система, розвивається і розгортається як єдине ціле. У цьому проекті використовується монолітна архітектура, яка має такі особливості:

1. **Одномонолітний проект**: Весь функціонал додатку зібраний у одному проекті (рис. 11.1.1).

2. **Централізоване управління**: Всі компоненти, такі як класи Admin, CourseSelectionHandler, Admins, розглядаються як частини цілого і мають доступ до загальних ресурсів і функціоналу.

3. **Простота в розгортанні**: Додаток розгортається як єдине ціле, що спрощує процес розгортання та встановлення.

4. **Одночасний розвиток**: Усі зміни та оновлення вносяться безпосередньо в один проект, що забезпечує одночасний розвиток всіх його компонентів.

5. **Єдиний механізм збереження даних:** Моделі даних, які розташовані у папці Models, використовуються для взаємодії з базою даних, забезпечуючи єдиний механізм доступу до інформації.

6. **Єдність технологічного стеку:** Використання одного Solution та одного проекту сприяє уніфікації технологічного стеку. Всі частини додатку використовують одні й ті ж бібліотеки та інструменти, що спрощує розробку та забезпечує єдність у підтримці.

7. **Монолітна архітектура та швидкість розробки:** Завдяки централізованій структурі та єдиному проекту розробка нових функціональностей може відбуватися швидко. Ви можете легко вносити зміни

у будь-яку частину додатку без необхідності координації між різними сервісами чи модулями.

Монолітна архітектура підходить для проектів з невеликим обсягом та обмеженим числом різнорідних функцій. Вона дозволяє швидко розробляти та впроваджувати програмні продукти, забезпечуючи простоту управління та розгортанням.

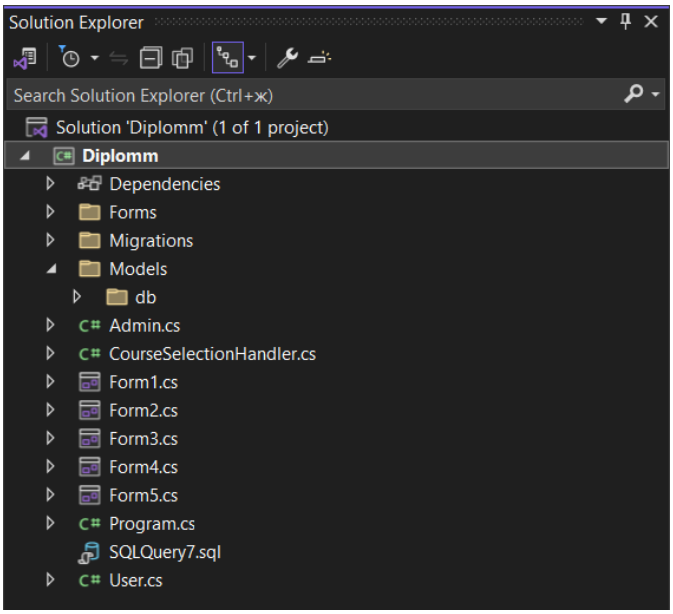

Рис. 11.1.1 – Solution Explorer

Папка Forms (рис. 11.1.2) у цьому проекті містить усі форми, які відповідають за візуальне представлення дисциплін:

1. **Управління інтерфейсом користувача**: Папка Forms включає у себе всі вікна та форми, які використовуються для відображення та взаємодії з даними щодо дисциплін. Кожна форма відповідає за конкретний аспект або функцію системи, наприклад, відображення списку дисциплін, їх редагування або вибір студентами.

2. **Реактивність та інтерактивність**: Форми відповідають за інтерактивне взаємодію з користувачем, забезпечуючи можливість введення даних, вибору параметрів і відображення результатів. Це включає обробку подій, валідацію даних та навігацію між різними екранами.

3. **Дизайн та єдність інтерфейсу**: Завдяки розміщенню усіх форм в одній папці ви забезпечуєте єдність стилю і дизайну інтерфейсу користувача. Це дозволяє зберігати консистентність у візуальному оформленні та взаємодії користувача з додатком.

Папка Forms є ключовою частиною вашої монолітної архітектури, оскільки вона об'єднує всі важливі елементи взаємодії з користувачем у вигляді графічного інтерфейсу.

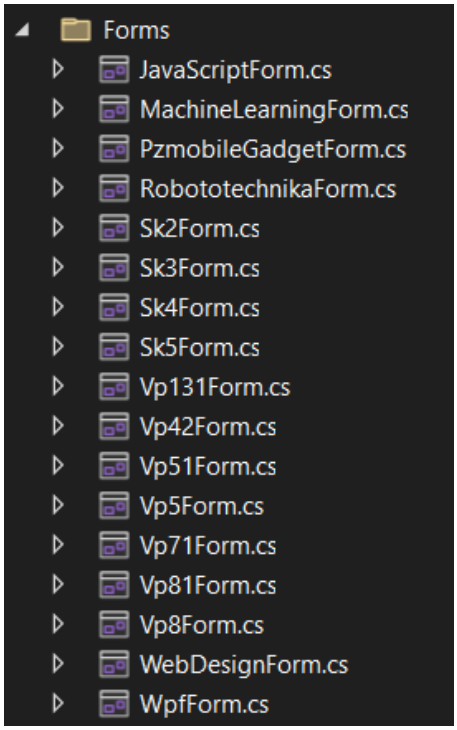

Рис. 11.1.2 – Папка Forms

Папка Migrations у проекті відіграє важливу роль в контексті роботи з базою даних, зокрема з механізмом Entity Framework Core, а саме:

1. **Керування структурою бази даних**: Папка Migrations містить всі необхідні файли, що відповідають за створення, модифікацію і оновлення структури вашої бази даних. Кожен файл міграції представляє собою інструкції для змін у схемі бази даних, таких як створення нових таблиць, зміни типів колонок або індексів.

2. **Історія змін бази даних**: Кожен файл міграції містить історію змін, які були внесені до бази даних. Це дозволяє вам відстежувати всі етапи розвитку схеми бази даних, включаючи створення та відкат попередніх змін.

3. **Версійний контроль бази даних**: Використання механізму міграцій забезпечує версійний контроль бази даних. Ви можете легко відкотитися до попередньої версії бази даних або застосувати нові зміни, що зроблені в розробці.

4. **Автоматична синхронізація з моделями даних**: Папка Migrations працює в парі з вашими моделями даних у папці Models. При зміні моделей автоматично генеруються нові міграції, що забезпечує збереження узгодженості між структурою вашої бази даних і логікою вашої програми.

Папка Migrations є важливим компонентом для забезпечення структури і цілісності бази даних в контексті монолітного застосунку. Вона спрощує процес розгортання, управління і моніторингу бази даних, забезпечуючи зручність в роботі з її структурою.

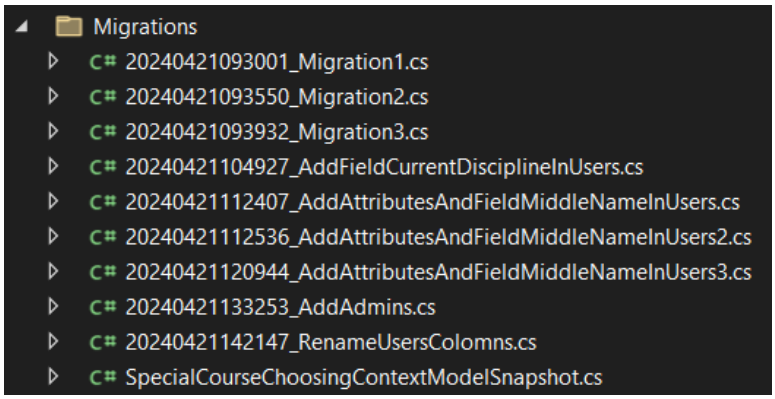

Рис. 11.1.3 – Папка Migrations

Папка Models => db у проекті містить класи сутностей, які представляють об'єкти бази даних для усіх дисциплін:

1. **Класи сутностей (Entity Classes)**: Папка db у папці Models містить класи, які відповідають сутностям бази даних. Кожен клас зазвичай

представляє одну таблицю в базі даних або її частину, і містить властивості, що відображають поля цієї таблиці.

2. **Відображення структури даних**: Класи сутностей відображають структуру даних вашої бази даних у коді. Вони дозволяють здійснювати зручний доступ до даних через об'єктно-орієнтований підхід, що спрощує роботу з даними у вашому додатку.

3. **Зв'язки між таблицями**: Класи сутностей також відображають зв'язки між таблицями бази даних за допомогою асоціацій між класами. Наприклад, зв'язок один до багатьох або багато до багатьох між об'єктами даних.

4. **Анотації Entity Framework**: Класи сутностей можуть містити анотації Entity Framework, що надають додаткову інформацію про спосіб мапінгування між класами і таблицями бази даних.

Папка Models => db є ключовою для визначення структури даних додатку і забезпечує зручний доступ до даних через об'єктно-орієнтований підхід. Вона співпрацює з папкою Migrations для забезпечення коректності структури бази даних і дозволяє ефективно управляти даними в монолітному застосунку.

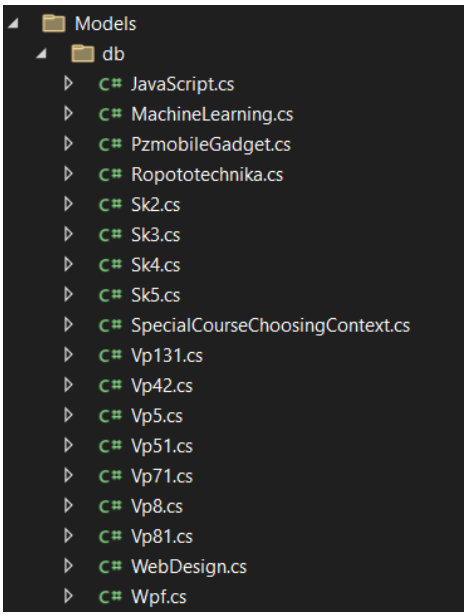

Рис. 11.1.4 – Папка сутностей Models

Та просто допоміжні класи:

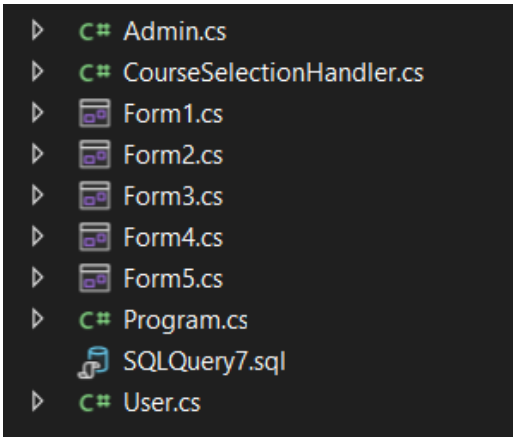

Рис. 11.1.5 – Інші сервіси програми

11.2 Сторонні бібліотеки

Сторонні бібліотеки (third-party libraries) є незалежними програмними компонентами, які розробники можуть використовувати для розширення функціональності своїх програм. Вони зазвичай розроблені окремими командами або компаніями та надаються для використання через відкриті або комерційні ліцензії. Ось деякі основні причини для використання сторонніх бібліотек:

1. **Розширення функціональності**: Бібліотеки дозволяють швидко і легко додавати нові функції до вашого проекту без необхідності писати власний код з нуля. Наприклад, готові бібліотеки можуть містити інструменти для роботи з графікою, обробки даних, мережами і т.д.

2. **Підвищення продуктивності**: Використання сторонніх бібліотек дозволяє зосередитися на ключових аспектах вашого проекту, замість написання і підтримки власного коду для завдань загального призначення.

3. **Спільнотна підтримка і активний розвиток**: Популярні сторонні бібліотеки часто мають активні спільноти розробників, які підтримують і розвивають їх, виправляють помилки і додають нові можливості.

Підключення сторонніх бібліотек у проект

Підключення сторонніх бібліотек до проекту може відрізнятися залежно від типу проекту і платформи, але основні кроки зазвичай виглядають так:

1. **Вибір бібліотеки**: Знаходження потрібної бібліотеки через платформу пошуку пакетів (наприклад, NuGet для .NET) або вручну завантаження з веб-сайту розробника.

2. **Встановлення через менеджер пакетів**: Використання менеджера пакетів для додавання бібліотеки до вашого проекту. Наприклад, в .NET це може бути NuGet Package Manager, доступний через Visual Studio або в командному режимі.

3. **Підключення в коді**: Після встановлення бібліотеки ви можете підключити її до вашого коду, імпортувавши необхідні простори імен і викликаючи методи чи класи з бібліотеки.

Entity Framework Core

Microsoft Entity Framework Core (EF Core) є сучасною технологією доступу до даних для .NET, яка дозволяє розробникам працювати з реляційними базами даних через об'єктно-орієнтовану модель даних. Основні особливості EF Core включають:

- **Підтримка різних провайдерів баз даних**: EF Core підтримує різні СУБД, такі як SQL Server, PostgreSQL, MySQL, SQLite та інші, що дозволяє розробникам вибирати підходящу СУБД для своїх проектів.
- **Міграції баз даних**: EF Core має вбудовану підтримку міграцій, що дозволяє автоматично створювати та оновлювати схему бази даних на основі змін у моделях даних.
- **Лінійка запитів**: EF Core надає LINQ (Language Integrated Query) для написання типобезпечних запитів до бази даних, що спрощує роботу з даними та знижує ймовірність помилок.
- **Підтримка асинхронності**: Бібліотека підтримує асинхронне виконання запитів, що поліпшує продуктивність додатків, особливо в веб-додатках.

Підключення EF Core до проекту (рис. 11.2.6) включає встановлення пакетів через NuGet, створення контексту бази даних та моделей даних для взаємодії з базою даних.

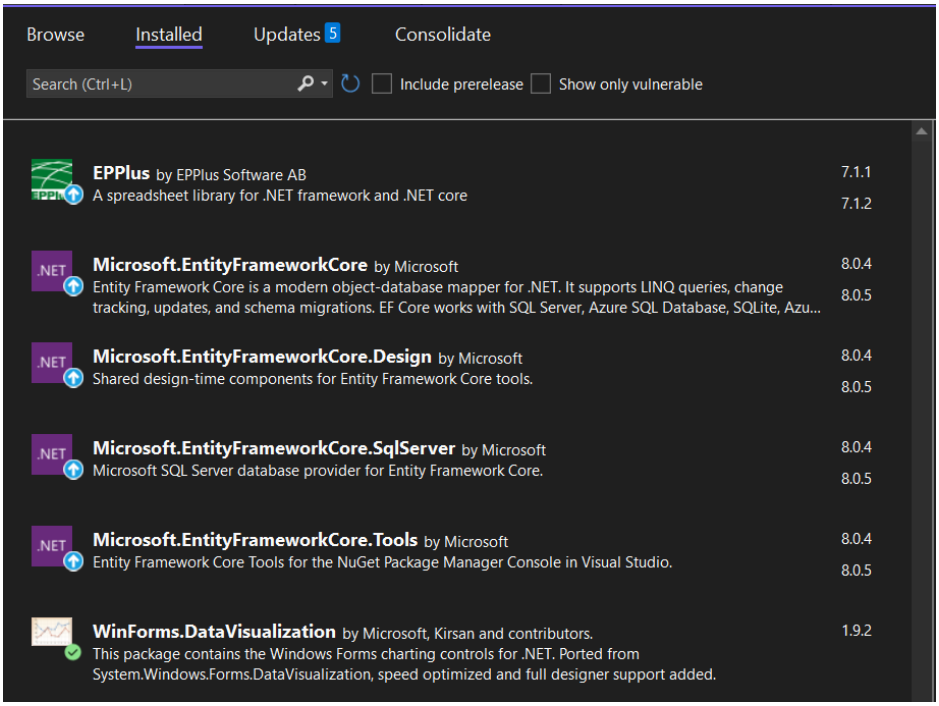

Рис. 11.2.6 – NuGet Packages

Ось так виглядають усі остаточні залежності проекта (рис.11.2.7):

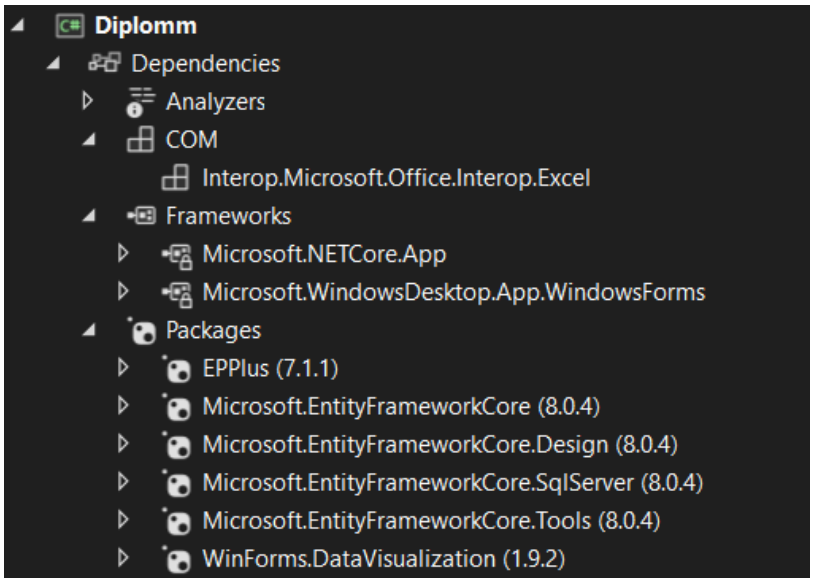

Рис. 11.2.7 – Залежності

ВИСНОВОК

У процесі розробки інформаційно-довідкової системи змісту навчальних дисциплін з вибіркової компоненти освітньої програми ми створили комплексний та ефективний інструмент, що полегшує адміністрування та управління навчальним процесом. Дана система забезпечує функціональність, необхідну для реєстрації користувачів, вибору дисциплін, а також управління та моніторингу даних здобувачів вищої освіти з боку адміністраторів. Впровадження таких рішень покращує організацію навчального процесу, підвищує його прозорість та ефективність, а також полегшує взаємодію між студентами та адміністрацією навчального закладу.

Основні результати роботи

1. ***Реєстрація користувачів:*** Ми розробили зручний інтерфейс для реєстрації нових користувачів, що включає валідацію введених даних на стороні клієнта та бази даних. Це забезпечує коректність та цілісність даних, що зберігаються в системі, знижуючи ймовірність помилок та покращуючи загальну стабільність програми.

2. ***Вибір дисциплін:*** Система дозволяє користувачам вибирати дисципліни з переліку доступних варіантів. Всі обрані дисципліни зберігаються у базі даних, забезпечуючи збереження інформації навіть при повторному вході в систему. Це сприяє більш персоналізованому підходу до навчання, враховуючи інтереси та потреби здобувачів вищої освіти.

3. ***Адміністративні функції:*** Адміністратори мають доступ до розширених функцій системи, включаючи можливість редагування даних здобувачів вищої освіти, управління дисциплінами та моніторинг вибору дисциплін. Це забезпечує ефективне адміністрування та підтримку навчального процесу, дозволяючи швидко реагувати на зміни та потреби здобувачів вищої освіти.

4. ***Взаємодія з базою даних:*** Використання підходу Entity Framework – Data Base First дозволило забезпечити автоматичну синхронізацію моделей з актуальною структурою бази даних. Міграції забезпечують версійний контроль, дозволяючи відслідковувати зміни та підтримувати базу даних у актуальному стані.

5. ***Графічна візуалізація:*** Графік популярності дисциплін, доступний для всіх користувачів, наочно демонструє розподіл обраних дисциплін та допомагає студентам приймати обґрунтовані рішення щодо вибору курсів. Це сприяє більш прозорому та інформованому процесу вибору дисциплін, а також допомагає адміністрації навчального закладу оптимізувати навчальну програму.

Практичне значення роботи

Розроблена інформаційно-довідкова система має велике практичне значення для навчальних закладів, оскільки вона дозволяє автоматизувати та оптимізувати багато процесів, пов'язаних з вибором навчальних дисциплін. Система сприяє зниженню навантаження на адміністративний персонал, підвищує точність та актуальність даних, а також забезпечує зручність та прозорість для здобувачів вищої освіти.

Перспективи подальших досліджень

У майбутньому можливим напрямком розвитку системи є інтеграція з іншими інформаційними системами навчального закладу, розширення функціональних можливостей системи може включати інтеграцію з системами онлайн-навчання, що дозволить забезпечити більш гнучкий та інтерактивний підхід до навчання та створити єдину інформаційну екосистему. Також варто розглянути можливість впровадження додаткових функцій, таких як

автоматичне формування розкладів на основі вибраних дисциплін або аналіз академічної успішності здобувачів вищої освіти.

Вплив на освітню сферу

Впровадження розробленої системи може значно поліпшити якість освіти шляхом покращення управління та моніторингу навчального процесу. Забезпечення більшої прозорості, доступності та персоналізації навчання сприятиме підвищенню мотивації здобувачів вищої освіти та покращенню їхніх навчальних результатів.

Загальний висновок

У даній кваліфікаційна роботі було розроблено та проаналізовано інформаційно-довідкову систему змісту навчальних дисциплін, що дозволяє ефективно управляти та моніторити навчальний процес. Розроблена система має практичне значення для навчальних закладів, забезпечуючи зручність, прозорість та ефективність у взаємодії між студентами та адміністрацією. Її впровадження сприятиме покращенню якості освіти та створенню сприятливого середовища для навчання та розвитку здобувачів вищої освіти.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Статті, тези:

1. М.В. Розум. Теорія прийняття рішень. Навчальний посібник для спеціальностей 122 – Комп’ютерні науки, 124 – Системний аналіз, 125 – Кібербезпека та захист інформації (для денної та заочної форм навчання). – Одеса: Видавництво «ОНМУ», 2023.- 290 с.
2. Новожилова М. В. Розробка експертних систем в середовищі CLIPS : навч. посібник // М. В. Новожилова, О. О. Петрова ; Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова. – Харків : ХНУМГ ім. О. М. Бекетова, 2019.–130 с.
3. Люльков М.М., Цеслів О.В. Моделювання управління інвестиційною діяльністю підприємства в умовах кризи. Збірник наукових праць молодих вчених. Актуальні проблеми економіки і управління. Випуск 14, 2020. С. 153-158.
4. A. Chepok, D. Larin, L. Martynovych, B. Panchenko, I. Sharipova. On fundamentals of creating an expert system for digitized texts’ style identification - Перспективні напрямки сучасної електроніки, інформаційних і комп'ютерних систем (MEICS-2023). Тези доповідей на VIII Всеукраїнській науково-практичній конференції: 22-24 листопада 2023 р., м. Дніпро / Укладач Іванченко О. В. – Дніпро, Дніпровський національний університет імені Олеся Гончара, ПП «Ліра ЛТД», 2023. – 262 с.- С. 138-139.
5. Методичні настанови до виконання здобувачами вищої освіти спеціальності 122 Комп’ютерні науки курсового проєкту з дисципліни «Методи та системи штучного інтелекту» / уклад. Єпік О.М.; Одеса, Одеський національний університет імені І. І. Мечникова. 2022. 19 с.
6. Нікітіна Л. О. Експертні системи [Електронний ресурс] : навч. посібник / Л. О. Нікітіна ; Нац. техн. ун-т "Харків. політехн. ін-т". – Електрон.

текст. дані. – Харків, 2023. – 210 с. – Режим доступа: https://repository.kpi.kharkov.ua/handle/KhPI-Press/65497.

7. Робоча програма навчальної дисципліни "Штучний інтелект та експертні системи" [Електронний ресурс] : [для здобувачів] другого (магістер.) рівня вищої освіти денної форми навчання спец. 125 "Кібербезпека" / розроб.: О. В. Мілов ; Нац. техн. ун-т "Харків. політехн. ін-т". – Електрон. текст. дані. – Харків, 2022. – 16 с. – Режим доступа: http://repository.kpi.kharkov.ua/handle/KhPI-Press/63836.
8. Роате М. А. Дослідження методу експертної класифікації / М. А. Роате, Д. Б. Єльчанінов // Теоретичні та практичні дослідження молодих вчених : зб. тез доп. 16-ї Міжнар. наук.-практ. конф. магістрантів та аспірантів, 14-16 грудня 2022 р. / ред. Є. І. Сокол ; Нац. техн. ун-т "Харків. політехн. ін-т" [та ін.]. – Харків : НТУ "ХПІ", 2022. – С. 45-46 – Режим доступа: https://repository.kpi.kharkov.ua/handle/KhPI-Press/62080.
9. Mamedov V. M. Expert systems and military reconnaissance data processing / V. M. Mamedov, A. A. Bayramov // Проблеми інформатизації : тези доп. 7-ї міжнар. наук.-техн. конф., 13-15 листопада 2019 р., м. Черкаси, м. Харків, м. Баку, м. Бельсько-Бяла : [у 3 т.]. Т. 3 / Черк. держ. технолог. ун-т [та ін.]. – Харків : Петров В. В., 2019. – С. 96. – Режим доступа: https://repository.kpi.kharkov.ua/handle/KhPI-Press/68694.
10. Ray Barrera, Aung Sithu Kyaw, and Thet Naing Swe titled Unity 2017 Game AI Programming – Third Edition.
11. Smart Computing and Self-Adaptive Systems. Edited BySimar Preet Singh, Arun Solanki, Anju Sharma, Zdzislaw Polkowski, Rajesh Kumar CRC Press, 2022 – 288pp
12. Северин С., Шаріпова І.В. Розробка інформаційно-довідкової системи змісту навчальних дисциплін з вибіркової компоненти освітньої програми. Міжнародна науково-технічна конференція здобувачів вищої освіти та молодих вчених “Комп’ютерні науки, інформаційні технології та системи управління” CSYSC-2023 : тези доп., 21-22 грудня 2023 р.,

Прикарпатський національний університет імені Василя Стефаника. С. 242-243 – Режим доступа: https://csysc.pnu.edu.ua/en/home-en/

13. Шаріпова І. В., Сєвєрін С. М. Розробка прототипу експертної системи - swi-програма «Розробка інформаційно-довідкової системи змісту навчальних дисциплін з вибіркової компоненти освітньої програми» // Двадцять перша всеукраїнська конференція студентів і молодих науковців «Інформатика, інформаційні системи та технології» - Одеса, 2024, с. 51-54

Web - сторінки:

14. Оцінка програми впровадження стандартів якості в вищій освіті – Режим доступа: https://forms.gle/pRu4Sq6wQ4Hwq8FcA (дата звернення 18.04.2024)
15. Методології створення експертних систем – Режим доступа: http://www.tsatu.edu.ua/kn/wp-content/uploads/sites/16/lekcija-_-3.pdf (дата звернення 22.03.2024 р.)
16. Експертні системи як навчальна дисципліна – Режим доступа: https://ua.kursoviks.com.ua/kompyuterni/ekspertni-sistemi (дата звернення 11.04.2024 р.)
17. Експертні системи – Режим доступа: https://pidru4niki.com/10811007/informatika/ekspertni_sistemi (дата звернення 14.03.2024 р.)
18. Експертні системи: структура і класифікація – Режим доступа: http://phys.bspu.by/static/lib/inf/posob/stu_m/glaves/glava16/gl_16_2.htm – (дата звернення 15.03.2024 р.)
19. Класифікація інформаційних систем – Режим доступа: https://ua5.org/isystem/374-klasifkacja-nformacjjnikh-sistem.html (дата звернення 16.03.2024 р.)
20. MSDN C# – Режим доступа: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення 25.01.2024 р.)

21. MSDN Entity Framework core – Режим доступа: https://learn.microsoft.com/en-us/ef/core/ (дата звернення 26.03.2024 р.)
22. MSDN Sql Server – Режим доступа: https://support.microsoft.com/uk-ua/topic/sql-%D0%B2-access-%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%96-%D0%BF%D0%BE%D0%BD%D1%8F%D1%82%D1%82%D1%8F-%D0%B3%D0%BB%D0%BE%D1%81%D0%B0%D1%80%D1%96%D0%B9-%D1%96-%D1%81%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81-444d0303-cde1-424e-9a74-e8dc3e460671 (дата звернення 14.04.2024 р.)
23. СУБД, які бувають, як вибрати – Режим доступа: https://highload.today/uk/subd-yaki-buvayut-yak-vibrati/ (дата звернення 25.03.2024 р.)
24. Object Relational Mapping – Режим доступа: https://javarush.com/ua/quests/lectures/ua.questhibernate.level09.lecture00 (дата звернення 12.04.2024 р.)
25. Створення першої програми Windows Forms на C# на прикладі гри «хрестики-нулики» – Режим доступа: http://isearch.kiev.ua/uk/news/programs/o-appl/1593-create-your-first-windows-forms-application-in-c-for-example-the-game-qtic-tac-toeq (дата звернення 09.03.2024 р.)
26. Використання Ado. Net Entity Framework для створення інформаційної системи управління документообігом кафедри – Режим доступа: https://cyberleninka.ru/article/n/vikoristannya-ado-net-entity-framework-dlya-stvorennya-informatsiynoyi-sistemi-upravlinnya-dokumentoobigom-kafedri (дата звернення 02.04.2024 р.)
27. Що таке платформи моделей додатків .NET? – Режим доступа: https://lemon.school/blog/shho-take-platformy-modelej-dodatkiv-net (дата звернення 01.03.2024 р.)

28. Про середовище програмування C# – Режим доступа: https://lemon.school/blog/shho-take-platformy-modelej-dodatkiv-net (дата звернення 20.04.2024 р.)
29. ДСТУ ISO/IEC 25000:2015. ДСТУ ISO/IEC 25000:2015 Інженерія програмних засобів і систем. Вимоги щодо якості та оцінювання систем і програмного продукту (SQuaRE). Настанова щодо SQuaRE (ISO/IEC 25000:2014, IDT). Чинний від 2016-01-01. Вид. офіц. – Режим доступа: https://www.iso.org/ru/standard/64764.html (дата звернення: 26.03.2024 р.)
30. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Чинний від 2016-07-01. Вид. офіц. Київ : УкрНДНЦ, 2016. 16 с

ДОДАТОК А

Міграції таблиць бази даних у класи моделей

Міграції таблиць бази даних у класи моделей. Клас контексту у C# є основою для взаємодії з базою даних у програмах, що використовують ORM (Object-Relational Mapping) фреймворки, такі як Entity Framework. Він відіграє ключову роль у забезпеченні ефективного та безпечного доступу до даних.

Оскільки у своїй програмі я використав підхід DataBase First, спочатку я створив таблиці в базі даних MSSQL і заповнив їх даними. Після цього, завдяки відповідній команді міграції механізм Entity Framework Core автоматично згенерував класи сутностей з назвами, що відповідають таблицям.

Основні функції класу контексту:

1. **Управління з'єднанням з базою даних**:
    - Клас контексту відповідає за відкриття та закриття з'єднання з базою даних. Він автоматично керує життєвим циклом з'єднання, забезпечуючи безпечний доступ до даних.
2. **Моделювання бази даних**:
    - Клас контексту містить властивості типу **DbSet<T>**, де **T** - це тип сутності. Ці властивості відображають таблиці бази даних. Наприклад, якщо у вас є таблиця **Students**, ви можете мати властивість **DbSet<Student> Students**, яка дозволяє вам виконувати CRUD (Create, Read, Update, Delete) операції з даними цієї таблиці.
3. **Забезпечення відстеження змін**:
    - Клас контексту відстежує всі зміни, які вносяться до об'єктів сутностей. Це дозволяє автоматично застосовувати зміни до бази даних при виклику методу **SaveChanges()**. Він підтримує збереження стану об'єктів і синхронізацію змін з базою даних.
4. **Виконання запитів до бази даних**:

- Клас контексту дозволяє виконувати запити LINQ (Language Integrated Query) для витягування даних з бази даних. Це забезпечує потужний і гнучкий спосіб вибірки даних, використовуючи знайомі C# конструкції.

5. **Конфігурація моделі даних**:
   - Через клас контексту можна конфігурувати модель даних за допомогою методу **OnModelCreating(ModelBuilder modelBuilder)**. Це дозволяє налаштовувати властивості сутностей, визначати ключі, встановлювати зв'язки між таблицями, застосовувати правила валідації тощо.

SpecialCourseChoosingContext

```
public partial class SpecialCourseChoosingContext : DbContext
{
    public SpecialCourseChoosingContext()
    {
        //Database.EnsureDeleted();
        Database.EnsureCreated();
        //Database.Migrate();
    }

    public
SpecialCourseChoosingContext(DbContextOptions<SpecialCourseChoos
ingContext> options)
        : base(options)
    {
    }

    public virtual DbSet<JavaScript> JavaScripts { get; set; }

    public virtual DbSet<MachineLearning> MachineLearnings {
get; set; }

    public virtual DbSet<PzmobileGadget> PzmobileGadgets { get;
set; }

    public virtual DbSet<Ropototechnika> Ropototechnikas { get;
set; }
```

Лістинг А1, лист 1 – Приклад класу контексту у C# з використанням Entity Framework Core

```
    public virtual DbSet<Sk2> Sk2s { get; set; }

    public virtual DbSet<Sk3> Sk3s { get; set; }

    public virtual DbSet<Sk4> Sk4s { get; set; }

    public virtual DbSet<Sk5> Sk5s { get; set; }

    public virtual DbSet<Vp131> Vp131s { get; set; }

    public virtual DbSet<Vp42> Vp42s { get; set; }

    public virtual DbSet<Vp5> Vp5s { get; set; }

    public virtual DbSet<Vp51> Vp51s { get; set; }

    public virtual DbSet<Vp71> Vp71s { get; set; }

    public virtual DbSet<Vp8> Vp8s { get; set; }

    public virtual DbSet<Vp81> Vp81s { get; set; }

    public virtual DbSet<WebDesign> WebDesigns { get; set; }

    public virtual DbSet<Wpf> Wpfs { get; set; }

    public virtual DbSet<User> Users { get; set; }

    public virtual DbSet<Admin> Admins { get; set; }

    protected override void
OnConfiguring(DbContextOptionsBuilder optionsBuilder)
#warning To protect potentially sensitive information in your
connection string, you should move it out of source code. You
can avoid scaffolding the connection string by using the Name=
syntax to read it from configuration - see
https://go.microsoft.com/fwlink/?linkid=2131148. For more
guidance on storing connection strings, see
https://go.microsoft.com/fwlink/?LinkId=723263.
        =>
optionsBuilder.UseSqlServer("Server=Stanislav003\\SQLEXPRESS;Dat
abase=SpecialCourseChoosing;Trusted_Connection=True;TrustServerC
ertificate=True;");

    protected override void OnModelCreating(ModelBuilder
modelBuilder)
    {
        modelBuilder.Entity<JavaScript>(entity =>
        {
```

Лістинг А1, лист 2

```
            entity
                .HasNoKey()
                .ToTable("JavaScript");


            entity.Property(e =>
e.Prn14).HasColumnName("PRN14");
            entity.Property(e => e.Sk8).HasColumnName("SK8");
            entity.Property(e => e.Zk2).HasColumnName("ZK2");
        });

        modelBuilder.Entity<MachineLearning>(entity =>
        {
            entity
                .HasNoKey()
                .ToTable("MachineLearning");

            entity.Property(e =>
e.Prn02).HasColumnName("PRN02");
            entity.Property(e =>
e.Prn03).HasColumnName("PRN03");
            entity.Property(e =>
e.Prn04).HasColumnName("PRN04");
            entity.Property(e =>
e.Prn05).HasColumnName("PRN05");
            entity.Property(e => e.Sk02).HasColumnName("SK02");
            entity.Property(e => e.Sk03).HasColumnName("SK03");
            entity.Property(e => e.Sk09).HasColumnName("SK09");
        });

        modelBuilder.Entity<PzmobileGadget>(entity =>
        {
            entity
                .HasNoKey()
                .ToTable("PZMobileGadgets");

            entity.Property(e => e.Ik).HasColumnName("IK");
            entity.Property(e => e.Pr14).HasColumnName("PR14");
            entity.Property(e => e.Sk8).HasColumnName("SK8");
            entity.Property(e => e.Zk2).HasColumnName("ZK2");
        });

        modelBuilder.Entity<Ropototechnika>(entity =>
        {
            entity
                .HasNoKey()
                .ToTable("Ropototechnika");

            entity.Property(e => e.Pr11).HasColumnName("PR11");
            entity.Property(e => e.Pr17).HasColumnName("PR17");
            entity.Property(e => e.Pr3).HasColumnName("PR3");
```

Лістинг А1, лист 3

```
    entity.Property(e => e.Pr4).HasColumnName("PR4");
    entity.Property(e => e.Pr5).HasColumnName("PR5");
    entity.Property(e => e.Pr7).HasColumnName("PR7");
    entity.Property(e => e.Pr9).HasColumnName("PR9");
    entity.Property(e => e.Sk5).HasColumnName("SK5");
    entity.Property(e => e.Sk6).HasColumnName("SK6");
    entity.Property(e => e.Sk7).HasColumnName("SK7");
    entity.Property(e => e.Vp2).HasColumnName("VP2");
});

modelBuilder.Entity<Sk2>(entity =>
{
    entity
        .HasNoKey()
        .ToTable("SK2");

    entity.Property(e => e.Ik).HasColumnName("IK");
    entity.Property(e => e.Pr1).HasColumnName("PR1");
    entity.Property(e => e.Pr12).HasColumnName("PR12");
    entity.Property(e => e.Pr13).HasColumnName("PR13");
    entity.Property(e => e.Pr17).HasColumnName("PR17");
    entity.Property(e => e.Pr3).HasColumnName("PR3");
    entity.Property(e => e.Pr4).HasColumnName("PR4");
    entity.Property(e => e.Zk1).HasColumnName("ZK1");
    entity.Property(e => e.Zk2).HasColumnName("ZK2");
    entity.Property(e => e.Zk3).HasColumnName("ZK3");
    entity.Property(e => e.Zk8).HasColumnName("ZK8");
});

modelBuilder.Entity<Sk3>(entity =>
{
    entity
        .HasNoKey()
        .ToTable("SK3");

    entity.Property(e => e.Pr2).HasColumnName("PR2");
    entity.Property(e => e.Pr5).HasColumnName("PR5");
    entity.Property(e => e.Sk12).HasColumnName("SK12");
    entity.Property(e => e.Sk8).HasColumnName("SK8");
});

modelBuilder.Entity<Sk4>(entity =>
{
    entity
        .HasNoKey()
        .ToTable("SK4");

    entity.Property(e => e.Pr14).HasColumnName("PR14");
    entity.Property(e => e.Sk8).HasColumnName("SK8");
```

Лістинг А1, лист 4

```
    entity.Property(e => e.Zk2).HasColumnName("ZK2");
});

modelBuilder.Entity<Sk5>(entity =>
{
    entity
        .HasNoKey()
        .ToTable("SK5");

    entity.Property(e => e.Pr1).HasColumnName("PR1");
    entity.Property(e => e.Pr12).HasColumnName("PR12");
    entity.Property(e => e.Pr13).HasColumnName("PR13");
    entity.Property(e => e.Sk6).HasColumnName("SK6");
    entity.Property(e => e.Zk15).HasColumnName("ZK15");
});

modelBuilder.Entity<Vp131>(entity =>
{
    entity
        .HasNoKey()
        .ToTable("VP131");

    entity.Property(e => e.Pr12).HasColumnName("PR12");
    entity.Property(e => e.Pr13).HasColumnName("PR13");
    entity.Property(e => e.Sk10).HasColumnName("SK10");
    entity.Property(e => e.Sk8).HasColumnName("SK8");
    entity.Property(e => e.Sk9).HasColumnName("SK9");
});
modelBuilder.Entity<Vp42>(entity =>
{
    entity
        .HasNoKey()
        .ToTable("VP42");
    entity.Property(e => e.Pr1).HasColumnName("PR1");
    entity.Property(e => e.Pr3).HasColumnName("PR3");
    entity.Property(e => e.Pr4).HasColumnName("PR4");
    entity.Property(e => e.Sk1).HasColumnName("SK1");
    entity.Property(e => e.Sk5).HasColumnName("SK5");
    entity.Property(e => e.Zk1).HasColumnName("ZK1");
    entity.Property(e => e.Zk2).HasColumnName("ZK2");
});
modelBuilder.Entity<Vp5>(entity =>
{
    entity
        .HasNoKey()
        .ToTable("VP5");
    entity.Property(e => e.Pr1).HasColumnName("PR1");
    entity.Property(e => e.Pr10).HasColumnName("PR10");
    entity.Property(e => e.Sk1).HasColumnName("SK1");
    entity.Property(e => e.Sk10).HasColumnName("SK10");
```

Лістинг А1, лист 5

```
    entity.Property(e => e.Sk16).HasColumnName("SK16");
    entity.Property(e => e.Sk2).HasColumnName("SK2");
    entity.Property(e => e.Sk5).HasColumnName("SK5");
    entity.Property(e => e.Zk1).HasColumnName("ZK1");
    entity.Property(e => e.Zk2).HasColumnName("ZK2");
});
modelBuilder.Entity<Vp51>(entity =>
{
    entity
        .HasNoKey()
        .ToTable("VP51");
    entity.Property(e => e.Pr1).HasColumnName("PR1");
    entity.Property(e => e.Pr10).HasColumnName("PR10");
    entity.Property(e => e.Sk1).HasColumnName("SK1");
    entity.Property(e => e.Sk10).HasColumnName("SK10");
    entity.Property(e => e.Sk16).HasColumnName("SK16");
    entity.Property(e => e.Sk5).HasColumnName("SK5");
    entity.Property(e => e.Zk1).HasColumnName("ZK1");
    entity.Property(e => e.Zk2).HasColumnName("ZK2");
});
modelBuilder.Entity<Vp71>(entity =>
{
    entity
        .HasNoKey()
        .ToTable("VP71");
    entity.Property(e => e.Pr1).HasColumnName("PR1");
    entity.Property(e => e.Pr10).HasColumnName("PR10");
    entity.Property(e => e.Sk13).HasColumnName("SK13");
});
modelBuilder.Entity<Vp8>(entity =>
{
    entity
        .HasNoKey()
        .ToTable("VP8");
    entity.Property(e => e.Pr1).HasColumnName("PR1");
    entity.Property(e => e.Pr12).HasColumnName("PR12");
    entity.Property(e => e.Pr13).HasColumnName("PR13");
    entity.Property(e => e.Pr3).HasColumnName("PR3");
    entity.Property(e => e.Pr4).HasColumnName("PR4");
    entity.Property(e => e.Sk13).HasColumnName("SK13");
    entity.Property(e => e.Sk8).HasColumnName("SK8");
    entity.Property(e => e.Zk1).HasColumnName("ZK1");
    entity.Property(e => e.Zk2).HasColumnName("ZK2");
    entity.Property(e => e.Zk3).HasColumnName("ZK3");
    entity.Property(e => e.Zk9).HasColumnName("ZK9");
});
modelBuilder.Entity<Vp81>(entity =>
{
    Entity
```

Лістинг А1, лист 6

```
                .HasNoKey()
                .ToTable("VP8.1");
            entity.Property(e => e.Pr1).HasColumnName("PR1");
            entity.Property(e => e.Pr12).HasColumnName("PR12");
            entity.Property(e => e.Pr13).HasColumnName("PR13");
            entity.Property(e => e.Pr3).HasColumnName("PR3");
            entity.Property(e => e.Pr4).HasColumnName("PR4");
            entity.Property(e => e.Sk1).HasColumnName("SK1");
            entity.Property(e => e.Sk13).HasColumnName("SK13");
            entity.Property(e => e.Sk2).HasColumnName("SK2");
            entity.Property(e => e.Sk8).HasColumnName("SK8");
            entity.Property(e => e.Zk3).HasColumnName("ZK3");
            entity.Property(e => e.Zk9).HasColumnName("ZK9");
        });
        modelBuilder.Entity<WebDesign>(entity =>
        {
            entity
                .HasNoKey()
                .ToTable("WebDesign");
            entity.Property(e => e.Ik).HasColumnName("IK");
            entity.Property(e => e.Pr12).HasColumnName("PR12");
            entity.Property(e => e.Pr13).HasColumnName("PR13");
            entity.Property(e => e.Pr17).HasColumnName("PR17");
            entity.Property(e => e.Pr3).HasColumnName("PR3");
            entity.Property(e => e.Pr4).HasColumnName("PR4");
        });
        modelBuilder.Entity<Wpf>(entity =>
        {
            entity
                .HasNoKey()
                .ToTable("WPF");
            entity.Property(e => e.Pr1).HasColumnName("PR1");
            entity.Property(e => e.Pr12).HasColumnName("PR12");
            entity.Property(e => e.Pr13).HasColumnName("PR13");
            entity.Property(e => e.Pr15).HasColumnName("PR15");
            entity.Property(e => e.Pr16).HasColumnName("PR16");
            entity.Property(e => e.Pr8).HasColumnName("PR8");
            entity.Property(e => e.Sk12).HasColumnName("SK12");
            entity.Property(e => e.Sk8).HasColumnName("SK8");
        });
           OnModelCreatingPartial(modelBuilder);
    }

    partial void OnModelCreatingPartial(ModelBuilder
modelBuilder);
}
```

Лістинг А.1, лист 7

ДОДАТОК Б

Головна форма програми

Клас Form1 відповідає за головну форму програми, яка надає користувачу можливість обирати та відкривати різні форми для роботи з різними модулями системи. Він також містить меню для додаткових функцій, таких як перегляд детальної інформації та популяризація.

Form1

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        this.StartPosition = FormStartPosition.CenterScreen;
    }

    private void JavaScriptButton_Click(object sender, EventArgs
e)
    {
        JavaScriptForm javaScriptForm = new JavaScriptForm();
        javaScriptForm.ShowDialog();
    }

    private void MachineLearningButton_Click(object sender,
EventArgs e)
    {
        MachineLearningForm machineLearningForm = new
MachineLearningForm();
        machineLearningForm.ShowDialog();
    }

    private void PzmobileGadgetButton_Click(object sender,
EventArgs e)
    {
        PzmobileGadgetForm pzmobileGadgetForm = new
PzmobileGadgetForm();
        pzmobileGadgetForm.ShowDialog();
    }

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

Лістинг Б.1, лист 1– Клас головного меню програми

```
        RobototechnikaForm robototechnikaForm = new
RobototechnikaForm();
        robototechnikaForm.ShowDialog();
    }

    private void Sk2Button_Click(object sender, EventArgs e)
    {
        Sk2Form sk2Form = new Sk2Form();
        sk2Form.ShowDialog();
    }

    private void Sk3Button_Click(object sender, EventArgs e)
    {
        Sk3Form sk3Form = new Sk3Form();
        sk3Form.ShowDialog();
    }

    private void Sk4Button_Click(object sender, EventArgs e)
    {
        Sk4Form sk4Form = new Sk4Form();
        sk4Form.ShowDialog();
    }

    private void Sk5Button_Click(object sender, EventArgs e)
    {
        Sk5Form sk5Form = new Sk5Form();
        sk5Form.ShowDialog();
    }

    private void Vp131Button_Click(object sender, EventArgs e)
    {
        Vp131Form vp131Form = new Vp131Form();
        vp131Form.ShowDialog();
    }

    private void Vp42Button_Click(object sender, EventArgs e)
    {
        Vp42Form vp42Form = new Vp42Form();
        vp42Form.ShowDialog();
    }

    private void Vp5Button_Click(object sender, EventArgs e)
    {
        Vp5Form vp5Form = new Vp5Form();
        vp5Form.ShowDialog();
    }

    private void VP51Button_Click(object sender, EventArgs e)
    {
        Vp51Form vp51Form = new Vp51Form();
```

Лістинг Б.1, лист 2

```
        vp51Form.ShowDialog();
    }
    private void Vp71Button_Click(object sender, EventArgs e)
    {
        Vp71Form vp71Form = new Vp71Form();
        vp71Form.ShowDialog();
    }
    private void Vp8Button_Click(object sender, EventArgs e)
    {
        Vp8Form vp8Form = new Vp8Form();
        vp8Form.ShowDialog();
    }

    private void VP81Button_Click(object sender, EventArgs e)
    {
        Vp81Form vp81Form = new Vp81Form();
        vp81Form.ShowDialog();
    }
    private void WebDesignButton_Click(object sender, EventArgs
e)
    {
        WebDesignForm webDesignForm = new WebDesignForm();
        webDesignForm.ShowDialog();
    }
    private void WPFButton_Click(object sender, EventArgs e)
    {
        WpfForm wpfForm = new WpfForm();
        wpfForm.ShowDialog();
    }
    private void детальноToolStripMenuItem_Click(object sender,
EventArgs e)
    {
        Form3 form3 = new Form3();
        form3.ShowDialog();
    }
    private void закритиToolStripMenuItem_Click(object sender,
EventArgs e)
    {
        DialogResult = DialogResult.No;
        this.Close();
    }
    private void популяризаціяToolStripMenuItem_Click(object
sender, EventArgs e)
    {
        Form5 form5 = new Form5();
        form5.ShowDialog();
    }
}
```

Лістинг Б.1, лист 3

Опис методів

1. Конструктор Form1:
    - Встановлює початкову позицію форми в центрі екрану.
2. Методи для обробки подій натискання кнопок:
    - JavaScriptButton_Click: Відкриває форму JavaScriptForm.
    - MachineLearningButton_Click: Відкриває форму MachineLearningForm.
    - PzmobileGadgetButton_Click: Відкриває форму PzmobileGadgetForm.
    - RobototechnikaButton_Click: Відкриває форму RobototechnikaForm.
    - Sk2Button_Click: Відкриває форму Sk2Form.
    - Sk3Button_Click: Відкриває форму Sk3Form.
    - Sk4Button_Click: Відкриває форму Sk4Form.
    - Sk5Button_Click: Відкриває форму Sk5Form.
    - Vp131Button_Click: Відкриває форму Vp131Form.
    - Vp42Button_Click: Відкриває форму Vp42Form.
    - Vp5Button_Click: Відкриває форму Vp5Form.
    - VP51Button_Click: Відкриває форму Vp51Form.
    - Vp71Button_Click: Відкриває форму Vp71Form.
    - Vp8Button_Click: Відкриває форму Vp8Form.
    - VP81Button_Click: Відкриває форму Vp81Form.
    - WebDesignButton_Click: Відкриває форму WebDesignForm.
    - WPFButton_Click: Відкриває форму WpfForm.
3. Методи для обробки подій меню:
    - детальноToolStripMenuItem_Click: Відкриває форму Form3 для детального перегляду.
    - закритиToolStripMenuItem_Click: Закриває головну форму.

ДОДАТОК В

Форма для введення даних користувача

Клас Form2 відповідає за введення даних користувача та їх збереження в базі даних.

Form2

```
public partial class Form2 : Form
{
    public Form2()
    {
        InitializeComponent();
        this.StartPosition = FormStartPosition.CenterScreen;
    }

    private void button1_Click(object sender, EventArgs e)
    {
        using (SpecialCourseChoosingContext context = new
SpecialCourseChoosingContext())
        {                if
(string.IsNullOrWhiteSpace(nameTextBox.Text) ||

string.IsNullOrWhiteSpace(surnameTextBox.Text) ||
                    string.IsNullOrWhiteSpace(ageTextBox.Text)
||

string.IsNullOrWhiteSpace(emailTextBox.Text))
                {
                    MessageBox.Show(
                        text: "Будь ласка, заповніть всі поля
перед продовженням.",
                        caption: "Попередження!",
                        buttons: MessageBoxButtons.OK,
                        icon: MessageBoxIcon.Error
                    );
                    return;
                }

                if (!int.TryParse(ageTextBox.Text, out int age)
|| age < 16 || age > 120)
                {
                    MessageBox.Show(
                        text: "Вік повинен бути цілим сислом від
16 до 120.",
                        caption: "Помилка",
```

Лістинг В.1, лист 1 – Клас Form2

```
                        buttons: MessageBoxButtons.OK,
                        icon: MessageBoxIcon.Error
                    );
                    return;
                }
                Dictionary<TextBox, int> textBoxesMaxLength =
new Dictionary<TextBox, int>
                {
                    { nameTextBox, 20 },
                    { surnameTextBox, 30 },
                    { middleNameTextBox, 30 },
                    { emailTextBox, 100 }
                };

                foreach (var pair in textBoxesMaxLength)
                {
                    if (pair.Key.Text.Length > pair.Value)
                    {
                        MessageBox.Show(
                            text: $"Перевищено максимальну
довжину поля '{pair.Key.Tag}'. Максимальна довжина -
{pair.Value} символів. Спробуйте ще раз!",
                            caption: "Помилка",
                            buttons: MessageBoxButtons.OK,
                            icon: MessageBoxIcon.Error
                        );
                        pair.Key.Text = string.Empty;
                        return;
                    }
                }
                User user = new User
                {
                    Name = nameTextBox.Text,
                    Surname = surnameTextBox.Text,
                    MiddleName = middleNameTextBox.Text,
                    Age = age,
                    Email = emailTextBox.Text,
                };
                context.Users.Add(user);
                context.SaveChanges();
                //DialogResult = DialogResult.OK;
                Form1 form1 = new Form1();
                form1.ShowDialog();
                this.Close();
            }
        }
    }
}
```

Лістинг В.1, лист 2

ДОДАТОК Г

Форма авторизації адміністратора

Цей код демонструє використання Windows Forms у поєднанні з Entity Framework для реалізації функції авторизації адміністратора. Код форми чітко структурований та відповідає вимогам чистого програмного забезпечення, забезпечуючи коректну роботу із базою даних та інтерфейсом користувача.

Form 3

```
public partial class Form3 : Form
{
    public Form3()
    {
        InitializeComponent();
        this.StartPosition = FormStartPosition.CenterScreen;
    }

    private void button1_Click(object sender, EventArgs e)
    {
        using (SpecialCourseChoosingContext context = new
SpecialCourseChoosingContext())
        {
            string name = nameTextBox.Text;
            string login = loginTextBox.Text;
            string password = passwordTextBox.Text;

            var admin = context.Admins.FirstOrDefault(a =>
a.Name == name && a.Login == login && a.Password == password);

            if (admin != null)
            {
                this.Close();

                Form4 form4 = new Form4();
                form4.ShowDialog();
            }
            else
            {
                MessageBox.Show(
                    text: "Такого користувача не існує!",
                    caption: "Помилка",
                    buttons: MessageBoxButtons.OK,
```

Лістинг Г.1, лист 1 – Клас Form3

```
                    icon: MessageBoxIcon.Error
                );

                nameTextBox.Text = string.Empty;
                loginTextBox.Text = string.Empty;
                passwordTextBox.Text = string.Empty;
                return;
            }
        }
    }
}
```

Лістинг Г.1, лист 2

Форма **Form3** відповідає за авторизацію адміністратора в системі. Основні елементи форми:

- **Конструктор форми** (Form3()): ініціалізує компоненти форми та встановлює положення на центрі екрану.
- **Обробник події кнопки** (button1_Click): викликається при натисканні на кнопку для авторизації. Виконує наступні дії:
    - Зчитує значення з текстових полів (nameTextBox, loginTextBox, passwordTextBox).
    - Проводить пошук адміністратора в базі даних за допомогою LINQ запиту FirstOrDefault.
    - Якщо адміністратор знайдений, закриває поточну форму і відкриває **Form4** (form4.ShowDialog()).
    - Якщо адміністратор не знайдений, показує повідомлення про помилку і очищає тексти бокси для повторного введення.

ДОДАТОК Д

Дисципліни та здобувачі вищої освіти: система управління

Цей код призначений для управління реєстрацією користувачів на спеціалізовані курси, які надаються навчальним закладом. Він дозволяє додавати нових користувачів, видаляти та оновлювати їхні дані, а також переглядати інформацію про поточні спеціалізації, які вони обрали.

Form4

```
public partial class Form4 : Form
{
    private User selectedUser;

    public Form4()
    {
        InitializeComponent();
        this.StartPosition = FormStartPosition.CenterScreen;
        UsersDataGridView.AutoSizeColumnsMode =
DataGridViewAutoSizeColumnsMode.Fill;
        InitializeDisciplineComboBox();
    }

    private void InitializeDisciplineComboBox()
    {
        List<string> disciplines = new List<string>
        {
            "ВП 10.1 СПЕЦКУРС №3«ДОСЛІДЖЕННЯ ТА ПРОЕКТУВАННЯ
СПЕЦІАЛІЗОВАНИХ КОМП’ЮТЕРНИХ СИСТЕМ»",
            "ВП 8.1 «ЕКСПЕРТНІ СИСТЕМИ»",
            "СПЕЦКУРС №2 ОБРОБКА ТА АНАЛІЗ ДАНИХ ЗАСОБАМИ
Python",
            "ВП11.1 СК 4  НАВІГАЦІЯ РОБОТОТЕХНІЧНИХ СИСТЕМ",
            "ВП12.1 СК5 КОМП'ЮТЕРНИЙ ЗІР ТА ОБРОБКА ЗОБРАЖЕНЬ",
            "ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ І МЕТОДИ MACHINE
LEARNING",
            "ВП 5.1  «Проблемно-орієнтовне програмування»",
            "Робототехніка та мехатроніка",
            "ВП 4.2.  НЕЧІТКІ СИСТЕМИ",
            "ВП13.1. СПЕЦКУРС №6  ОСНОВИ КВАНТОВОЇ ІНФОРМАТИКИ",
            "ВП 7.1 СИСТЕМНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ
БАГАТОПРОЦЕСОРНИХ КОМП'ЮТЕРНИХ СИСТЕМ",
            "ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ МОБІЛЬНИХ ПРИСТРОЇВ",
            "Спецкурс №4 Клієнтське вебпрограмування мовою
```

Лістинг Д.1, лист 1 – Клас Form4

```
JavaScript",
            "ВП10 Спецкурс №3 Технологія WPF",
            "СПЕЦКУРС №2  ОСНОВИ АДАПТИВНОГО ВЕБ-ДИЗАЙНУ",
            "ВП8 «ВІЗУАЛІЗАЦІЯ ТА СИНТЕЗ ЗОБРАЖЕНЬ ЗА ДОПОМОГОЮ
КОМП'ЮТЕРНОЇ ГРАФІКИ»",
            "ВП5 «ПРОБЛЕМНО-ОРІЄНТОВНЕ ПРОГРАМУВАННЯ»"
        };

        disciplineComboBox.DataSource = disciplines;
    }

    private void Form4_Load(object sender, EventArgs e)
    {
        using (SpecialCourseChoosingContext context = new
SpecialCourseChoosingContext())
        {
            var users = context.Users.ToList();

            if (users != null)
            {
                UsersDataGridView.DataSource = null;
                UsersDataGridView.DataSource = users;
            }
            else
            {
                MessageBox.Show(
                    text: "Ще жоден користувач не обрав
спецкурс!",
                    caption: "Увага",
                    buttons: MessageBoxButtons.OK,
                    icon: MessageBoxIcon.Error
                );

                UsersDataGridView.DataSource = null;
            }
        }
    }

    private void disciplineComboBox_SelectedIndexChanged(object
sender, EventArgs e)
    {
        disciplineTextBox.Text =
disciplineComboBox.SelectedItem?.ToString();
    }

    private void UsersDataGridView_CellClick(object sender,
DataGridViewCellEventArgs e)
    {
        if (UsersDataGridView.CurrentRow != null)
        {
```

Лістинг Д.1, лист 2

```
            DataGridViewRow row = UsersDataGridView.CurrentRow;
            selectedUser = (User)row.DataBoundItem;

            nameTextBox.Text = selectedUser.Name;
            surnameTextBox.Text = selectedUser.Surname;
            middlenameTextBox.Text = selectedUser.MiddleName;
            ageTextBox.Text = selectedUser.Age.ToString();
            emailTextBox.Text = selectedUser.Email;
            disciplineTextBox.Text =
selectedUser.CurrentDiscipline;
        }
    }

    private void addButton_Click(object sender, EventArgs e)
    {
        if (!ValidateTextBoxes())
        {
            return;
        }

        using (SpecialCourseChoosingContext context = new
SpecialCourseChoosingContext())
        {
            User newUser = new User
            {
                Name = nameTextBox.Text,
                Surname = surnameTextBox.Text,
                MiddleName = middlenameTextBox.Text,
                Age = Convert.ToInt32(ageTextBox.Text),
                Email = emailTextBox.Text,
                CurrentDiscipline = disciplineTextBox.Text,
            };

            context.Users.Add(newUser);
            context.SaveChanges();

            UpdateDataGridView(context);

            MessageBox.Show("Користувач успішно створений!",
"Успіх", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }

    private void deleteButton_Click(object sender, EventArgs e)
    {
        if (selectedUser != null)
        {
            using (SpecialCourseChoosingContext context = new
SpecialCourseChoosingContext())
            {
```

Лістинг Д.1, лист 3

```
                context.Users.Remove(selectedUser);
                context.SaveChanges();

                UpdateDataGridView(context);

                MessageBox.Show("Користувач успішно видалений!",
"Успіх", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
        }
    }

    private void updateButton_Click(object sender, EventArgs e)
    {
        if (selectedUser != null)
        {
            if (!ValidateTextBoxes())
            {
                return;
            }

            using (SpecialCourseChoosingContext context = new
SpecialCourseChoosingContext())
            {
                var existingUser =
context.Users.Find(selectedUser.Id);

                if (existingUser != null)
                {
                    existingUser.Name = nameTextBox.Text;
                    existingUser.Surname = surnameTextBox.Text;
                    existingUser.MiddleName =
middlenameTextBox.Text;
                    existingUser.Age =
Convert.ToInt32(ageTextBox.Text);
                    existingUser.Email = emailTextBox.Text;
                    existingUser.CurrentDiscipline =
disciplineTextBox.Text;

                    context.SaveChanges();

                    UpdateDataGridView(context);
                    MessageBox.Show("Дані успішно змінені!",
"Успіх", MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
            }
        }
        else
        {
            MessageBox.Show("Користувач не знайдений в базі
даних!.");
```

Лістинг Д.1, лист 4

```
        }
    }

    private void UpdateDataGridView(SpecialCourseChoosingContext
context)
    {
        var users = context.Users.ToList();

        UsersDataGridView.DataSource = null;
        UsersDataGridView.DataSource = users;

        UsersDataGridView.ResetBindings();
    }
    private bool ValidateTextBoxes()
    {
        if (string.IsNullOrWhiteSpace(nameTextBox.Text) ||
string.IsNullOrWhiteSpace(surnameTextBox.Text) ||
            string.IsNullOrWhiteSpace(middlenameTextBox.Text) ||
string.IsNullOrWhiteSpace(ageTextBox.Text) ||
            string.IsNullOrWhiteSpace(emailTextBox.Text) ||
string.IsNullOrWhiteSpace(disciplineTextBox.Text))
        {
            MessageBox.Show("Усі поля повинні бути заповнені!",
"Увага!", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return false;
        }
        if (nameTextBox.Text.Length > 20)
        {
            MessageBox.Show("Ім'я не повинно перевищувати 20
символів.", "Увага!", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            nameTextBox.Text = string.Empty;
            return false;
        }
        if (surnameTextBox.Text.Length > 30)
        {
            MessageBox.Show("Прізвище не повинно перевищувати 30
символів.", "Увага!", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            surnameTextBox.Text = string.Empty;
            return false;
        }
        if (middlenameTextBox.Text.Length > 30)
        {
            MessageBox.Show("По-батькові не повинно перевищувати
30 символів.", "Увага!", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            middlenameTextBox.Text = string.Empty;
            return false;
```

Лістинг Д.1, лист 5

```
          }
        int age;
        if (!int.TryParse(ageTextBox.Text, out age) || age < 16
|| age > 120)
        {
            MessageBox.Show("Вік має бути цілим числом від 16 до
120.", "Увага!", MessageBoxButtons.OK, MessageBoxIcon.Error);
            ageTextBox.Text = string.Empty;
            return false;
        }

        if (emailTextBox.Text.Length > 100)
        {
            MessageBox.Show("Email не повинен перевищувати 100
символів.", "Увага!", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            emailTextBox.Text = string.Empty;
            return false;
        }

        return true;
    }
}
```

Лістинг Д.1, лист 6

Цей код включає функціональність для роботи з користувачами із спеціалізованою інформацією про курси. Ось основні моменти:

1. **InitializeDisciplineComboBox()**: Ініціалізує комбобокс зі списком спеціалізацій курсів.
2. **Form4_Load()**: Завантажує користувачів у DataGridView при завантаженні форми.
3. **disciplineComboBox_SelectedIndexChanged()**: Відображає вибрану спеціалізацію у відповідному текстовому полі.
4. **UsersDataGridView_CellClick()**: Обробляє клік по комірці DataGridView для відображення даних вибраного користувача.
5. **addButton_Click()**: Додає нового користувача з усіма перевірками валідності введених даних.
6. **deleteButton_Click()**: Видаляє обраного користувача.

7. **updateButton_Click()**: Оновлює існуючі дані користувача з усіма перевірками валідності введених даних.
8. **UpdateDataGridView()**: Оновлює вміст DataGridView після змін у базі даних.
9. **ValidateTextBoxes()**: Перевіряє всі введені текстові поля на валідність (наприклад, на наявність даних та правильність формату).

ДОДАТОК Е

Аналіз спецкурсів за кількістю навичок

Цей код створює форму Form5, яка містить діаграму Chart, що відображає кількість навичок для кожної дисципліни. Ось код класу Form5, який включає налаштування графіку:

Form5

```
public partial class Form5 : Form
{
    public Form5()
    {
        InitializeComponent();
        DrawChart();
        this.StartPosition = FormStartPosition.CenterScreen;
    }
    private void DrawChart()
    {
        Chart chart = new Chart();
        chart.Dock = DockStyle.Fill;

        ChartArea chartArea = new ChartArea();
        chart.ChartAreas.Add(chartArea);
        List<Tuple<string, int>> classes = new
List<Tuple<string, int>>
        {
            Tuple.Create("ВП 10.1 СПЕЦКУРС №3«ДОСЛІДЖЕННЯ ТА
ПРОЕКТУВАННЯ СПЕЦІАЛІЗОВАНИХ КОМП’ЮТЕРНИХ СИСТЕМ»", 4),
            Tuple.Create("ВП 8.1 «ЕКСПЕРТНІ СИСТЕМИ»", 11),
            Tuple.Create("СПЕЦКУРС №2 ОБРОБКА ТА АНАЛІЗ  ДАНИХ
ЗАСОБАМИ Python", 11),
            Tuple.Create("ВП11.1 СК 4  НАВІГАЦІЯ РОБОТОТЕХНІЧНИХ
СИСТЕМ", 3),
            Tuple.Create("ВП12.1 СК5 КОМП'ЮТЕРНИЙ ЗІР ТА ОБРОБКА
ЗОБРАЖЕНЬ", 5),
            Tuple.Create("ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ І МЕТОДИ
MACHINE LEARNING", 7),
            Tuple.Create("ВП 5.1  «Проблемно-орієнтовне
програмування»", 8),
            Tuple.Create("Робототехніка та мехатроніка", 11),
```

Лістинг Е.1, лист1 – Клас Form5

```
            Tuple.Create("ВП 4.2.  НЕЧІТКІ СИСТЕМИ", 7),
            Tuple.Create("ВП13.1. СПЕЦКУРС №6  ОСНОВИ КВАНТОВОЇ
ІНФОРМАТИКИ", 5),
            Tuple.Create("ВП 7.1 СИСТЕМНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ
БАГАТОПРОЦЕСОРНИХ КОМП'ЮТЕРНИХ СИСТЕМ", 3),
            Tuple.Create("ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ МОБІЛЬНИХ
ПРИСТРОЇВ", 4),
            Tuple.Create("Спецкурс №4  Клієнтське
вебпрограмування мовою JavaScript", 3),
            Tuple.Create("ВП10 Спецкурс №3 Технологія WPF", 8),
            Tuple.Create("СПЕЦКУРС №2  ОСНОВИ АДАПТИВНОГО ВЕБ-
ДИЗАЙНУ", 6),
            Tuple.Create("ВП8 «ВІЗУАЛІЗАЦІЯ ТА СИНТЕЗ ЗОБРАЖЕНЬ
ЗА ДОПОМОГОЮ КОМП'ЮТЕРНОЇ ГРАФІКИ»", 11),
            Tuple.Create("ВП5 «ПРОБЛЕМНО-ОРІЄНТОВНЕ
ПРОГРАМУВАННЯ»", 9),
        };
        foreach (var cls in classes)
        {
            Series series = new Series();
            //series.ChartType = SeriesChartType.Bar;
            series.Name = cls.Item1;
            series.Points.AddXY(cls.Item1, cls.Item2);
            series.CustomProperties = "PixelPointWidth=500";
            chart.Series.Add(series);
        }
        chartArea.AxisY.Title = "Кількість навичок";
        chartArea.AxisY.TitleFont = new Font("Arial", 10,
FontStyle.Bold);
        chartArea.AxisX.Enabled = AxisEnabled.False;
        //chartArea.AxisX.Title = "Дисципліни";
        Legend legend = new Legend();
        legend.Docking = Docking.Left;
        chart.Legends.Add(legend);
        Controls.Add(chart);
    }
}
```

Лістинг Е.1, лист2

У цьому коді додано функціональність для відображення стовпчикової діаграми, яка відображає кількість навичок на кожній дисципліни.

ДОДАТОК Ж

Сутності адміністраторів системи

Модель Admin представляє собою клас, який визначає структуру даних для зберігання інформації про адміністраторів у системі.

Модель **Admin**

```
public class Admin
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }

    [Required]
    [MaxLength(20)]
    public string Name { get; set; } = null!;

    [Required]
    [MaxLength(100)]
    public string Login { get; set; } = null!;

    [Required]
    [MaxLength(50)]
    public string Password { get; set; } = null!;
}
```

Лістинг Ж.1, лист 1 – Клас моделі Admin

Основні особливості цієї моделі:

1. **Id**: Це поле представляє унікальний ідентифікатор для кожного адміністратора. Позначене атрибутами [Key] та [DatabaseGenerated(DatabaseGeneratedOption.Identity)], що вказує на автоматичне генерування значення.
2. **Name**: Поле для зберігання імені адміністратора. Встановлено обов'язковість заповнення за допомогою атрибута [Required]. Максимальна довжина імені обмежена 20 символами за допомогою [MaxLength(20)].

3. **Login**: Поле для зберігання логіну адміністратора. Обов'язкове для заповнення ([Required]) і обмежене до 100 символів за допомогою [MaxLength(100)].
4. **Password**: Поле для зберігання паролю адміністратора. Як і поля Login і Name, вимагається обов'язкове заповнення ([Required]). Максимальна довжина паролю обмежена 50 символами ([MaxLength(50)]).

Ця модель використовується для створення таблиці Admins в базі даних, де кожен запис відповідає одному адміністратору системи, зберігаючи їхні основні ідентифікаційні дані.

ДОДАТОК К

Керування користувачами та їхніми даними

Цей код представляє модель даних User для роботи з базою даних у зазначеній програмі.

Модель **User**

```
public class User
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }

    [Required]
    [MaxLength(20)]
    [Column("Ім'я")]
    public string Name { get; set; } = null!;

    [Required]
    [MaxLength(30)]
    [Column("Фамілія")]
    public string Surname { get; set; } = null!;

    [Required]
    [MaxLength(30)]
    [Column("По батькові")]
    public string MiddleName { get; set; } = null!;

    [Required]
    [Range(16, 120)]
    [Column("Вік")]
    public int Age { get; set; }

    [Required]
    [MaxLength(100)]
    public string Email { get; set; } = null!;

    [Column("Обрана дисципліна")]
    public string? CurrentDiscipline { get; set; }
}
```

Лістинг К.1 – Клас моделі User