МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ТРУХАНСЬКА ВЛАДИСЛАВА ОЛЕКСІЇВНА

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«____»_____ 20__ р.

РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ CRM-СИСТЕМИ ДЛЯ ШКОЛИ ІНОЗЕМНИХ МОВ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
О. В. Зелінська, доцент кафедри
інформаційних технологій,
к. т. н., доцент
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____

Вінниця – 2025

АНОТАЦІЯ

Труханська В.О. Розробка клієнтської частини CRM-системи для школи іноземних мов. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2025.

У кваліфікаційній (бакалаврській) роботі досліджено функціональні можливості та структуру CRM-систем, проаналізовано сучасні технології розробки клієнтської частини та обґрунтовано вибір архітектурного підходу. Результатом роботи є розроблена клієнтська частина CRM-системи для школи іноземних мов, реалізована мовою C# з використанням WPF. Система має основні функції керування студентами, курсами та фінансами, а також може бути інтегрована з серверною частиною.

Ключові слова: CRM-система, front-end, десктопний застосунок, інтерфейс, WPF, Figma

68 ст., 27 рис., 6 дод., 42 джерела.

ABSTRACT

Trukhanska V.O. **Development of the Client Part of the CRM System for a School of Foreign Languages**. Specialty 122 «Computer Science», educational program «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia, 2025.

In the qualification (bachelor's) work the functional capabilities, structure of CRM systems, modern technologies for developing the front-end are investigated and the choice of architectural approach is justified. The result of the work is the development of the client-side of the CRM system for a school of foreign languages, implemented in C# using WPF. The system offers essential tools for students, courses and finance management and can also be integrated with the server-side.

Keywords: CRM system, front-end, desktop application, interface, WPF, Figma

ЗМІСТ

ВСТУП

У сучасному світі інформаційні технології відіграють ключову роль у забезпеченні ефективності та конкурентоспроможності бізнесу, зокрема у сфері освітніх послуг. Одним із найбільш важливих інструментів для автоматизації внутрішніх процесів організацій виступають CRM-системи (Customer Relationship Management systems) – системи управління взаємовідносинами з клієнтами. У контексті приватних освітніх установ, зокрема шкіл іноземних мов, впровадження CRM-систем стає необхідною умовою для підтримки високого рівня сервісу, організації взаємодії з клієнтами, планування навчального процесу, ведення фінансової звітності, контролю якості обслуговування тощо.

Однак попри широку представленість CRM-рішень на ринку, більшість з них або орієнтовані на великі корпорації, або мають надлишкову функціональність і складний інтерфейс, що ускладнює адаптацію таких систем для невеликих приватних шкіл. Крім того, значна частина таких систем орієнтована на веб або мобільні платформи, тоді як потреба в десктопних рішеннях зберігається, особливо у випадках, коли навчальні заклади прагнуть мати автономне програмне забезпечення. Відтак постає завдання розробки власної CRM-системи, адаптованої до реалій і потреб конкретної школи іноземних мов, з особливою увагою до клієнтської частини – тієї складової, яка безпосередньо забезпечує взаємодію користувача з функціоналом системи. Тому дослідження та практична реалізація клієнтської частини CRM-системи, орієнтованої на мовну школу, є актуальним завданням, яке відповідає сучасним потребам галузі.

Метою цієї кваліфікаційної роботи є розробка дизайну, створення прототипу, а також написання програмної реалізації клієнтської частини CRM-системи для школи іноземних мов як десктопного застосунку з урахуванням специфіки освітніх процесів, вимог користувачів та принципів зручності взаємодії. Для досягнення мети роботи необхідно виконати аналіз літературних джерел, основних технологій та інструментів, архітектури та функціональності

для розробки десктопного застосунку CRM-системи. На основі отриманих даних провести оцінку ефективності роботи застосунку та навести рекомендації щодо покращення його діяльності.

Завдання дослідження:

- проаналізувати роль CRM-систем у сучасних бізнес-процесах, зокрема у сфері освітніх послуг;
- дослідити сучасні технології для розробки клієнтської частини десктопного застосунку;
- сформулювати функціональні вимоги до клієнтської частини CRM-системи для мовної школи;
- узагальнити принципи побудови ефективного користувацького інтерфейсу (UI/UX);
- розробити прототип інтерфейсу з використанням інструментів проєктування;
- реалізувати функціональну клієнтську частину CRM-системи засобами обраної технології;
- протестувати реалізований інтерфейс та оцінити його відповідність поставленим вимогам.

Об’єктом дослідження є процес розробки клієнтської частини CRM-систем у контексті автоматизації діяльності закладів освіти.

Предметом дослідження є програмна реалізація інтерфейсу користувача CRM-системи для мовної школи та технології, що забезпечують її функціонування, що включає архітектурні рішення, інструменти UI/UX-дизайну та засоби розробки десктопних застосунків.

Практична значущість дослідження полягає у створенні клієнтської частини CRM-системи, що може бути впроваджена у діяльність шкіл іноземних мов для автоматизації управління навчальним процесом, ведення обліку клієнтів та оптимізації внутрішніх бізнес-процесів. Теоретичні результати, зокрема аналіз інструментів розробки front-end частини для десктопних застосунків, можуть

слугувати базою для подальших досліджень у сфері програмної інженерії, UI/UX-дизайну та автоматизації освітніх сервісів.

Результати дослідження апробовано на конференціях, де було розглянуто вплив нейромереж на розробку вебдизайну та проаналізовано актуальність фреймворку WPF для front-end розробки, а також у віснику СНТ ДонНУ, де досліджено NLP-технології для CRM-систем.

• *Труханська В. О., Зелінська О. В.* Вплив нейромереж на розробку вебдизайну. Збірник матеріалів III Міжнародної науково-практичної конференції «Прикладні аспекти сучасних міждисциплінарних досліджень», (м. Вінниця, 01 листопада 2024 р.). Вінниця: ДонНУ імені Василя Стуса, 2024. 296 с;

• *Труханська В. О., Зелінська О. В.* Аналіз актуальності фреймворку WPF для front-end розробки. Збірник матеріалів XI Міжнародної науково-практичної конференції студентів, аспірантів і молодих учених «Актуальні проблеми гуманітарних, технічних і природничих наук», м. Вінниця, 30 квітня 2025 року;

• *Труханська В. О., Зелінська О. В.* Аналіз NLP-технологій для CRM-систем. Вісник студентського наукового товариства Донецького національного університету імені Василя Стуса, м. Вінниця: ДонНУ імені Василя Стуса, 2025. Вип. 17, т. 1. 248 с.

Кваліфікаційна робота складається з трьох основних розділів, кожен з яких логічно розкриває етапи дослідження та реалізації клієнтської частини CRM-системи: розділ 1 присвячено теоретичному аналізу предметної області, огляду сучасних CRM-систем, технологій front-end розробки та формулюванню завдання розробки; розділ 2 містить етапи проєктування користувацького інтерфейсу, аналіз інструментів прототипування та розробку UI-дизайну у Figma; розділ 3 розкриває програмну реалізацію клієнтської частини у вигляді десктопного застосунку, включає структуру застосунку, реалізацію інтерфейсу та опис основних функціональних можливостей.

РОЗДІЛ 1
ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ FRONT-END ЧАСТИНИ CRM-СИСТЕМИ

1.1 Огляд CRM-систем та їх роль у бізнес-процесах

У нинішню епоху цифровізації бізнесу автоматизація процесів управління взаємовідносинами з клієнтами (Customer Relationship Management – CRM) стає надзвичайно важливою для підприємств різного масштабу та галузей діяльності. CRM-системи є комплексними програмними рішеннями, які забезпечують збирання, зберігання, обробку та аналітику інформації про клієнтів, а також координацію бізнес-процесів, орієнтованих на покращення якості обслуговування та підвищення лояльності споживачів [1]. Можна охарактеризувати також як стратегічну концепцію управління, що фокусується на формуванні та утриманні довгострокових взаємовідносин з клієнтами, що базується на глибокому аналізі їхніх потреб та поведінкових патернах [1]. З технічної точки зору CRM-система – це програмна платформа, яка реалізовує ці стратегічні принципи шляхом інтеграції інструментів для обробки комунікацій, обліку продажів, управління маркетинговими кампаніями та надання сервісних послуг.

Перші CRM-рішення з'явилися у 1990-х роках у відповідь на зростання потреб бізнесу в автоматизації процесів продажу та покращенні обслуговування клієнтів. З плином часу функціональні можливості CRM-систем значно розширилися: від простих баз даних контактів до складних екосистем, які охоплюють багатоканальні взаємодії, аналітику великих обсягів даних (Big Data), мобільний доступ, а також інтеграцію зі сторонніми сервісами та корпоративними ERP-системами, що на рис. 1.1 [2]. У контексті бізнес-процесів CRM-системи виконують роль центрального вузла управління клієнтським досвідом. Вони дозволяють підприємствам структуровано організовувати інформацію про кожного клієнта, що включає історію покупок, запити в службу підтримки, участь у маркетингових заходах та інші взаємодії з брендом. Ця

централізація сприяє формуванню цілісного бачення клієнта («360-градусний погляд»), що забезпечує більш персоналізований підхід до обслуговування.

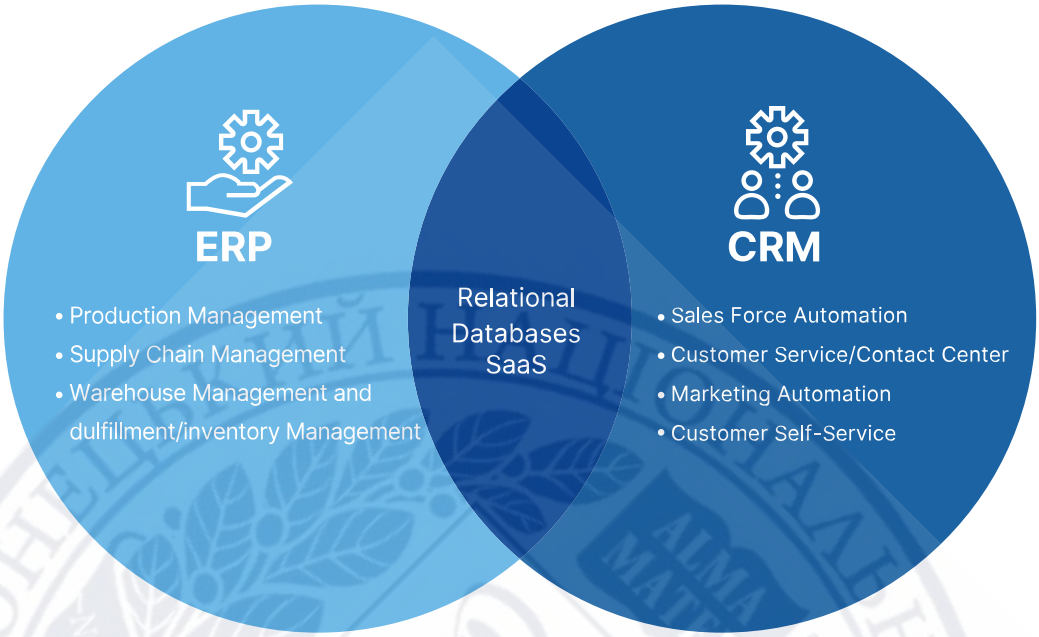

Рисунок 1.1 – Взаємодія функцій ERP та CRM-систем

Основна цінність CRM-систем полягає у підвищенні ефективності ключових бізнес-функцій, таких як: продажі, маркетинг та обслуговування клієнтів [3]. Завдяки автоматизації рутинних операцій CRM-рішення дозволяють оптимізувати витрати часу та ресурсів, скоротити тривалість циклу продажу та підвищити коефіцієнт конверсії потенційних клієнтів у реальних покупців. Більше того аналітичні модулі CRM-систем надають можливість здійснювати прогнозування поведінки споживачів, виявляти тренди ринку та формувати більш ефективні стратегії взаємодії.

З огляду на різноманітність бізнесів, на ринку представлено кілька типів CRM-систем, які відрізняються як архітектурно, так і за функціональним наповненням. Найпоширенішими є операційні, аналітичні та колаборційні CRM-системи [3]. Перші орієнтовані на автоматизацію повсякденних бізнес-процесів, пов’язаних із взаємодією з клієнтами. Такі системи забезпечують інструменти для співробітників компанії, що допомагають їм ефективніше виконувати свої функції, наприклад, швидко оформлювати замовлення, реєструвати звернення клієнтів, вести розклад зустрічей тощо. Операційний CRM концентрується на автоматизації фронт-офісних процесів – продажах,

сервісі, маркетингу, і часто включає модулі Sales Force Automation (SFA), Marketing Automation і Service/Support. Аналітичні CRM-системи фокусуються на аналізі накопичених даних про клієнтів з метою виявлення закономірностей і підтримки прийняття рішень [3]. У них використовуються інструменти бізнес-аналітики, data mining для сегментації клієнтської бази, оцінки прибутковості клієнтів, прогнозування поведінки споживачів. Аналітичний CRM допомагає компанії зрозуміти потреби та цінність різних клієнтів для підлаштування стратегії. Колаборaційні CRM-системи передбачають побудову прямих взаємодій із клієнтами за допомогою засобів самообслуговування та комунікаційних платформ. Ідея полягає в тому, що частина взаємодії здійснюється автоматично без участі співробітників компанії. Прикладами можуть бути клієнтські портали, системи інтернет-банкінгу, чат-боти, інтеграція з соціальними мережами. Колаборaційний CRM розширює можливості клієнтів самостійно отримувати інформацію та взаємодіяти з компанією онлайн, що підвищує зручність і швидкість обслуговування.

Рисунок 1.2 – Схема роботи CRM-системи

Усі три типи часто поєднуються в сучасних CRM-системах і надають комплексне рішення. На рис. 1.2 наведено узагальнену схему роботи CRM-системи в бізнесі: дані з різних каналів взаємодії з клієнтами надходять до єдиної бази, де піддаються класифікації та аналізу, після чого необхідна інформація розповсюджується до різних підрозділів компанії для підтримки їхньої роботи. Така інтеграція забезпечує єдиний інформаційний простір навколо клієнта.

З технічного боку CRM-системи реалізовуються як вебзастосунки, десктопні платформи або як мобільні версії. Сучасні тенденції розвитку ринку вказують на домінування хмарних (SaaS) рішень, які забезпечують високу масштабованість та доступність системи з будь-якої точки світу. Однак для специфічних потреб окремих організацій продовжують використовуватись локальні CRM-системи, що забезпечують більший контроль над зберіганням та обробкою конфіденційних даних.

У галузі освіти, зокрема в діяльності шкіл іноземних мов, CRM-системи стають незамінним інструментом для автоматизації широкого спектру процесів. До таких процесів належать: облік студентів, управління навчальним розкладом, контроль оплат за абонементи, надсилання повідомлень через електронну пошту, ведення звітів. У контексті мовної школи CRM-система виконує функцію єдиного інформаційного простору, в якому взаємодіють адміністратори, викладачі та студенти. Саме інтеграція цих користувачів в єдину систему дозволяє підвищити якість надання освітніх послуг та забезпечити гнучке управління освітніми програмами. З позиції бізнес-процесів мовної школи, CRM-система виконує не лише облікову, але й стратегічну функцію. Вона забезпечує збереження інформації про історію відвідувань та навчальні досягнення студентів, що дозволяє формувати персоналізовані рекомендації щодо подальшого навчання. Крім того, CRM допомагає аналізувати ефективність роботи викладачів, виявляти вузькі місця у навчальних програмах та оперативно реагувати на потреби ринку освітніх послуг.

Архітектура сучасних CRM-систем базується на багаторівневій моделі, де виділяються клієнтський рівень (front-end), серверна логіка (back-end) та рівень

зберігання даних (database). Front-end частина відповідає за реалізацію інтерфейсу користувача, через який здійснюється основна взаємодія із системою. Ефективність та зручність front-end компонента є вагомими факторами успіху системи, адже саме цей рівень визначає користувацький досвід (User Experience – UX) [4].

У розробці CRM-системи для школи іноземних мов особливої уваги потребує адаптація інтерфейсу під потреби різних категорій користувачів – адміністраторів, викладачів та студентів. Важливим аспектом є забезпечення доступності функціоналу, інтуїтивної навігації та високої продуктивності інтерфейсу, що особливо актуально при роботі з великими обсягами даних про студентів та навчальні програми [4].

Еволюція CRM-систем також пов’язана з інтеграцією новітніх технологій, таких як машинне навчання, інтелектуальний аналіз даних та автоматизовані чат-боти. Застосування штучного інтелекту дозволяє підвищити точність прогнозування поведінки клієнтів, а також автоматизувати рутинні завдання, такі як відповіді на типові запити або нагадування про майбутні заняття. У контексті мовної школи це може виражатися у автоматичному підборі навчальних матеріалів відповідно до рівня знань студента та його прогресу. Ще одним трендом розвитку CRM-систем є поява омніканальних платформ, що забезпечують інтеграцію різних каналів комунікації, таких як: електронна пошта, месенджери, соціальні мережі та телефонія, в єдине інформаційне середовище. Це дозволяє підприємствам будувати більш гнучкі та персоналізовані стратегії взаємодії з клієнтами.

Отже, CRM-системи стали ключовим елементом цифрової трансформації бізнесу, які виступають не лише як інструменти автоматизації, але й як стратегічні платформи для розвитку клієнтських відносин. Їх впровадження у різні сфери діяльності, включно з освітніми закладами, дозволяє не лише оптимізувати операційну діяльність, а й підвищити конкурентоспроможність на ринку шляхом поліпшення якості обслуговування та персоналізації послуг. У зв'язку з цим розробка ефективної front-end частини CRM-системи, яка б

забезпечувала зручний та функціональний доступ до всіх можливостей платформи, є пріоритетним завданням сучасних проєктів у сфері інформаційних технологій.

1.2 Аналіз технологій та інструментів для побудови клієнтської частини застосунку CRM-системи

У розробці інформаційних систем, зокрема CRM-систем, клієнтська частина відіграє чималу роль у забезпеченні ефективної взаємодії користувача із системою. Якість реалізації користувацького інтерфейсу безпосередньо впливає на продуктивність роботи персоналу, рівень задоволеності клієнтів і загальну конкурентоспроможність продукту. Підходи до побудови front-end частини CRM-систем є різноманітними та залежать як від типу застосунку, так і від обраної архітектури та технологічного стеку.

Загалом, існують три основні підходи до реалізації front-end частини CRM-системи, які визначають тип програмного інтерфейсу:

- десктопні застосунки;
- вебзастосунки;
- мобільні застосунки.

Кожен із підходів має свої переваги та обмеження, які необхідно враховувати при виборі архітектури системи.

Десктопні застосунки зберігають актуальність у випадках, коли потрібна висока продуктивність та інтеграція з локальними системами підприємства. У середовищі операційної системи Windows найбільш популярними технологіями для створення таких рішень є Windows Forms, Windows Presentation Foundation (WPF) та платформа Universal Windows Platform (UWP) [5].

Windows Forms – це набір бібліотек, який дозволяє створювати графічні інтерфейси користувача для застосунків під Windows. Вона надає простий спосіб створення вікон, кнопок, текстових полів та інших елементів інтерфейсу. Windows Forms популярна завдяки своїй простоті та зручності, однак вона має обмежені можливості в плані графічної складності та анімацій [5].

WPF – це потужна платформа для розробки графічних інтерфейсів у Windows, яка дає можливість створювати багатофункціональні та красиві інтерфейси з використанням складних графічних елементів, анімацій та переходів [6]. Один з основних компонентів WPF – це використання XAML (eXtensible Application Markup Language), який дозволяє відокремити логіку інтерфейсу від бізнес-логіки [6, 7]. WPF дозволяє реалізувати складні інтерфейси, де можна використовувати шари візуальних ефектів, як-от 3D-графіку, а також підтримує концепцію data binding, що спрощує роботу з даними. Завдяки своїм можливостям, WPF ідеально підходить для розробки CRM-системи, яка потребує інтерактивних та складних інтерфейсів [8].

UWP є платформою для розробки універсальних застосунків, які можуть працювати на різних пристроях в екосистемі Windows, що включає ПК, планшети, мобільні пристрої, Xbox та інші [8, 9]. Основна перевага UWP полягає в тому, що застосунки можуть працювати на різних пристроях із єдиним кодом, що забезпечує зручність у розробці та підтримці. UWP використовує XAML для розмітки інтерфейсів і надає доступ до широкого спектра API для роботи з пристроями [10].

Вебзастосунки працюють на сервері, а інтерфейс користувача завантажується в браузері. Вони зазвичай базуються на стандартних вебтехнологіях:

- HTML (HyperText Markup Language) – основа вебсторінки, яка визначає структуру контенту [11, 12].
- CSS (Cascading Style Sheets) – мова стилів, яка використовується для оформлення вебсторінок, включно з кольорами, шрифтами, розміткою [11, 12].
- JavaScript – мова програмування, яка дозволяє додавати інтерактивність і динамічні елементи на вебсторінки [11, 12].

Наразі популярними є фреймворки та бібліотеки, що реалізовують архітектурні патерни, спрямовані на ефективну побудову SPA (Single Page Applications), що, в свою чергу, забезпечує швидку реакцію інтерфейсу на дії

користувача і відповідно мінімізує кількість запитів до сервера та підвищує зручність користування [13, 14]. До таких фреймворків належать Angular, React, Vue.js та Svelte.

Angular забезпечує потужну основу для розробки складних односторінкових застосунків, завдяки своїй модульній архітектурі [15]. Він дозволяє організувати код у вигляді модулів, що спрощує масштабування та тестування. Однією з ключових особливостей Angular є система двостороннього зв'язку даних (two-way data binding), яка дозволяє автоматично синхронізувати зміни між даними та інтерфейсом. Це особливо корисно для створення динамічних інтерфейсів, де взаємодія з користувачем безпосередньо впливає на зміну даних. Окрім цього, Angular включає багато вбудованих функцій: анімації, форми, вбудовану маршрутизацію, а також підтримує роботу з HTTP-запитами, що дозволяє швидко створювати потужні та інтерактивні вебзастосунки [16, 17].

React, в свою чергу, є бібліотекою, яка підходить для створення високопродуктивних інтерфейсів. Розроблена Facebook, вона зосереджується на створенні компонентів, що дозволяє створювати масштабовані та гнучкі вебзастосунки. Однією з головних переваг React є його віртуальний DOM, який дозволяє оновлювати лише ті частини інтерфейсу, які змінилися, завдяки чому покращується продуктивність і зменшується кількість рендерів [15]. Крім того, React підтримує серверний рендеринг, що допомагає значно прискорити час завантаження сторінки, що є важливим фактором для користувачів з різними типами з’єднань.

Vue.js привертає увагу своєю простотою і гнучкістю, що робить його ідеальним вибором для новачків. Цей фреймворк дозволяє швидко створювати інтерактивні інтерфейси завдяки своїй компонентній архітектурі, що забезпечує структурованість і масштабованість проєктів. Однією з основних його переваг є підтримка двостороннього зв'язку даних, що дозволяє автоматично оновлювати інтерфейс при зміні стану програми [16]. Vue.js також відомий своєю легкістю в освоєнні, що робить його популярним серед розробників, які лише починають знайомитися з сучасними фреймворками.

Svelte пропонує абсолютно новий підхід до створення SPA. Замість того, щоб створювати віртуальний DOM, як це роблять інші фреймворки, Svelte компілює компоненти в чистий JavaScript, який безпосередньо маніпулює DOM. Це означає, що результати роботи застосунку займають менше місця, і продуктивність значно зростає, адже не потрібно виконувати додаткові кроки для управління віртуальним DOM [17]. Svelte є ідеальним вибором для розробників, які шукають легкий і швидкий фреймворк, котрий дозволяє досягати високих результатів при мінімальних витратах на ресурси.

Наступними є мобільні застосунки, які набули особливої популярності внаслідок зростання попиту користувачів на необхідність взаємодії в режимі реального часу. Для розробки таких застосунків використовують як нативні технології, так і кросплатформенні фреймворки [18,19]. Останні дозволяють створювати єдиний код для різних платформ, що знижує витрати на розробку та супровід системи. Тож детальніше про нативні технології:

- Swift – мова програмування для розробки нативних застосунків для iOS [18]. Вона розроблена Apple і є потужним інструментом для створення швидких та ефективних застосунків на платформі iOS.
- Kotlin – мова програмування, яка використовується для створення нативних застосунків для Android. Kotlin є офіційною мовою для Android розробки та підтримує всі функції сучасних мов програмування [20].
- Java – традиційна мова програмування для Android. Хоча Kotlin зараз домінує в розробці для Android, Java все ще широко використовується [20].

А також детальніше про кросплатформенні фреймворки:

- React Native дозволяє використовувати JavaScript для написання коду, який працюватиме як на iOS, так і на Android [18, 21].
- Flutter від Google, який дозволяє розробляти мобільні застосунки для iOS та Android з єдиним кодом. Flutter використовує мову програмування Dart і пропонує високу продуктивність завдяки рендерингу на основі власного механізму малювання [18, 21].

- Xamarin від Microsoft з використанням C# дозволяє розробляти застосунки для iOS та Android завдяки використанню спільної кодової бази [18, 19].

Сучасні підходи до побудови front-end частини CRM-систем також визначаються архітектурними патернами та принципами проєктування. Найбільш розповсюдженими патернами є MVC (Model-View-Controller), MVP (Model-View-Presenter), MVVM (Model-View-ViewModel), а також Code Behind, який прийнято відносити до менш формалізованих, але широко застосовуваних підходів у практичних проєктах [22].

Архітектурний патерн MVC (Model-View-Controller) є класичним вибором розробки, що передбачає чітке розділення відповідальностей на три основні компоненти [23]. Модель (Model) відповідає за бізнес-логіку та управління даними. Представлення (View) реалізовує відображення даних та взаємодію з користувачем. Контролер (Controller) виконує роль посередника, який обробляє події та координує оновлення моделі і представлення. Підхід MVC забезпечує високу масштабованість системи та спрощує тестування компонентів, оскільки кожна частина системи ізольована від інших. У веброзробці фреймворки на зразок ASP.NET MVC, Angular та Ruby on Rails активно впроваджують цей патерн. У контексті десктопних застосунків MVC менш поширений через складність інтеграції з моделлю подій GUI [24].

Патерн MVP (Model-View-Presenter) еволюційно розвиває ідеї MVC та широко застосовується в десктопних та мобільних застосунках [25]. У цій архітектурі Presenter замінює контролер і виступає як активний посередник між моделлю та представленням. На відміну від MVC, де View може напряму звертатись до Model, у MVP вся логіка взаємодії проходить через Presenter. Цей підхід забезпечує ще більшу ізоляцію представлення від бізнес-логіки, що підвищує тестованість UI та спрощує модульне тестування [24]. MVP активно використовується, наприклад, у WinForms-застосунках і в розробці під Android до поширення MVVM.

Патерн MVVM (Model-View-ViewModel) отримав найбільший розвиток саме у середовищі WPF, а також у UWP та Xamarin.Forms. Він є подальшим

удосконаленням MVP і повністю використовує переваги декларативного підходу до побудови інтерфейсу за допомогою XAML [26]. У патерні MVVM:

- модель (Model) зберігає бізнес-дані та правила;
- представлення (View) описує зовнішній вигляд інтерфейсу у XAML;
- ViewModel виступає як адаптер між View та Model, що реалізовує властивості для прив’язки даних (data binding) і команди (commands) для обробки подій.

Основною перевагою MVVM є мінімізація залежності між View та бізнес-логікою, що дозволяє ефективно застосовувати Unit-тестування для ViewModel та полегшує підтримку проєктів великого масштабу з багатокористувацькою взаємодією [26]. Однак ці переваги супроводжуються суттєвим ускладненням архітектури – потребою впровадження додаткових шарів абстракції, механізмів прив’язки даних та впорядкування команд, що може бути надмірним для невеликих локальних проєктів.

Підхід Code Behind є найпростішим і найбільш прямолінійним способом побудови логіки взаємодії з інтерфейсом у середовищі WPF [27]. У цьому підході події, які виникають у представленні, наприклад, натискання кнопки, зміна тексту у полі вводу, обробляються безпосередньо у відповідному файлі класу, пов’язаному з XAML-розміткою. У Code Behind логіка дій розташована поруч із описом інтерфейсу, що спрощує розробку та налагодження. Головною перевагою підходу Code Behind є його простота та швидкість розробки, що робить його ідеальним вибором для малих і середніх локальних проєктів, де немає потреби у суворому розділенні шарів абстракції [28]. Застосування цього підходу дозволяє мінімізувати накладні витрати на розробку та зосередитися на досягненні прикладних функціональних цілей системи. Саме тому для розробки клієнтської частини CRM-системи для школи іноземних мов було обрано підхід Code Behind як найбільш доцільний.

При використанні Code Behind для локальних CRM-систем забезпечується безпосередній контроль над логікою роботи інтерфейсу, що дозволяє гнучко обробляти різноманітні сценарії взаємодії із користувачем без потреби в

складних механізмах прив’язки даних [27, 28]. Це також спрощує обслуговування проєкту в разі, коли його супровід здійснюється невеликою командою або одним розробником. На противагу цьому, патерн MVVM є оптимальним для великих корпоративних рішень із десятками форм та сотнями користувачів, де важливо забезпечити модульність, розширюваність та автоматичне тестування інтерфейсу.

Важливим аспектом побудови front-end частини є забезпечення якісного користувацького досвіду (UX) та привабливого користувацького інтерфейсу (UI) [29]. Дизайн інтерфейсу CRM-системи має відповідати принципам доступності, інтуїтивності та послідовності. Вебтехнології пропонують численні UI-бібліотеки, такі як Material UI, Ant Design та Bootstrap, що полегшують реалізацію адаптивного дизайну та типових компонентів системи [29]. У WPF для цього використовується система стилів і шаблонів, які дозволяють досягти єдиного візуального вигляду застосунку. Технічні вимоги до front-end CRM-систем передбачають також реалізацію механізмів безпеки, таких як аутентифікація користувачів, контроль доступу до ресурсів, захист від атак типу CSRF та XSS. У десктопних застосунках захист забезпечується на рівні авторизації доступу до функцій програми та шифрування даних під час зберігання чи передачі.

1.3 Постановка завдання

З урахуванням актуальних вимог до програмних рішень, постановка завдання розробки клієнтської частини CRM-системи для школи іноземних мов передбачає чітке визначення функціональних та нефункціональних вимог, вибір оптимальних технологій реалізації та формування архітектури майбутнього застосунку.

З огляду на різні підходи розробки з попереднього підрозділу головною метою цієї роботи є створення front-end частини CRM-системи у вигляді десктопного застосунку з використанням технології Windows Presentation Foundation (WPF). Вибір саме цієї платформи зумовлений необхідністю побудови

багатофункціонального інтерфейсу, який забезпечуватиме зручну роботу адміністративного персоналу, викладачів та керівництва школи. Використання WPF забезпечить можливість створення складного інтерфейсу з великим набором елементів управління, таких як: таблиці даних, календарі, вкладки та графіки, що особливо важливо для CRM-системи, яка оперує значними обсягами інформації про студентів, викладачів та навчальні курси. Завдяки цьому можливо буде реалізувати гнучкі механізми візуалізації даних та організації складних сценаріїв взаємодії із системою.

Окрему увагу варто звернути на вибір архітектурного підходу до реалізації логіки клієнтської частини. З огляду на масштаб проєкту, а саме локальну CRM-систему з відносно невеликою кількістю користувачів, не доцільно застосовувати патерн MVVM, який хоч і забезпечує високий рівень відокремлення логіки представлення, але вимагає значних ресурсів на впровадження та підтримку. Тож, для розробки застосунку варто обрати підхід Code Behind, що дозволить реалізувати логіку взаємодії з інтерфейсом безпосередньо у відповідних класах подій. Такий підхід зменшить складність розробки проєкту на початковому етапі та дозволить досягти швидкого результату без впровадження додаткових шарів абстракції.

Завдання розробки передбачає створення інтерфейсу, який охоплює основні функціональні можливості CRM-системи: облік студентів і працівників, управління навчальним розкладом, відправка повідомлень, контроль фінансових операцій та формування звітів. Інтерфейс має забезпечувати доступ до функціоналу відповідно до ролі користувача, що вимагає реалізації механізмів аутентифікації та авторизації. Користувацький досвід (UX) має бути побудований з урахуванням принципів зручності, доступності та швидкої навігації по системі.

При розробці передбачається застосування модульного підходу до побудови інтерфейсу. Це дозволить забезпечити гнучкість структури програми та можливість подальшого розширення функціоналу без суттєвого перероблення вже реалізованих компонентів. Основні модулі включатимуть: модуль обліку студентів, модуль розкладу занять, модуль фінансів, модуль аналітики та

звітності, а також модуль налаштувань системи. Для підвищення зручності роботи передбачається реалізація функцій пошуку, фільтрації та сортування записів.

Ключовим етапом постановки завдання є визначення вимог до інтеграції клієнтської частини з серверною частиною системи та базою даних. Застосунок має взаємодіяти з сервером через API-запити, а також обробляти помилки під час передачі інформації. Це забезпечить чітку синхронізацію даних між клієнтською та серверною частинами системи. У процесі розробки також враховуватиметься необхідність реалізації механізмів валідації введених користувачем даних, що сприятиме зниженню ризику помилок при введенні інформації. Важливим аспектом є забезпечення коректної роботи програми з великими обсягами даних, що потребує оптимізації процесів завантаження та відображення інформації у відповідних елементах інтерфейсу. Щодо питань безпеки, доступ до конфіденційних даних має бути захищений відповідними механізмами шифрування та контролю доступу.

З технічної точки зору, серед технологій, що використовуватимуться в проєкті, можна виділити мову програмування C#, яка є основною для розробки застосунків під платформу .NET. Дизайн інтерфейсу реалізовуватиметься за допомогою мови розмітки XAML, що дозволить чітко відокремити структуру представлення від логіки роботи. Такий підхід забезпечить більш структуровану організацію коду та полегшить супровід програми.

Підхід до побудови front-end частини CRM-системи має враховувати також вимоги до масштабованості та можливості подальшого розвитку. У разі необхідності система має бути адаптована до нових бізнес-вимог. Технологія WPF, як частина екосистеми .NET, забезпечує таку гнучкість завдяки підтримці модульної архітектури та розширюваності застосунку. Слід також зазначити, що сучасні тенденції в розробці front-end частин CRM-систем передбачають активне використання принципів адаптивного дизайну та мобільної доступності. Хоча у межах проєкту розробка веб або мобільної версії не передбачена, архітектура

застосунку закладає потенціал для подальшої реалізації такого розширення у випадку масштабування діяльності школи.

Отже, постановка завдання розробки клієнтської частини CRM-системи для школи іноземних мов передбачає визначення чіткої функціональної архітектури, вибір оптимальних технологій реалізації, а також формування вимог до безпеки, продуктивності та масштабованості програмного продукту. Застосування технології WPF у поєднанні з архітектурним підходом Code Behind є обґрунтованим рішенням, яке дозволяє забезпечити високу якість інтерфейсу та гнучкість у реалізації функціональних сценаріїв взаємодії з користувачем. Такий вибір відповідає специфіці завдання й потребам освітнього закладу, що створює підґрунтя для подальшого розвитку системи та її адаптації до змінних бізнес-вимог.

РОЗДІЛ 2
ПРОЄКТУВАННЯ КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ КЛІЄНТСЬКОЇ ЧАСТИНИ ЗАСТОСУНКУ CRM-СИСТЕМИ

2.1 Основні принципи UI/UX та аналіз інструментів для прототипування інтерфейсів

У сучасному програмному забезпеченні якість користувацького інтерфейсу (UI) та досвіду взаємодії користувача (UX) є передовим фактором успіху цифрових продуктів. Проєктування передбачає врахування не лише естетичних вимог, але й психологічних, ергономічних та функціональних чинників, що формують цілісний досвід використання системи, аби використання було максимально простим, зручним і зрозумілим.

Перш за все варто почати з детальнішого визначення того, що таке UI та UX. Термін UI (User Interface), що перекладається як «інтерфейс користувача», охоплює всі елементи візуальної складової програмного продукту, з якими взаємодіє користувач. Це включає кнопки, текстові поля, меню, іконки, колірні схеми, типографіку та розташування елементів на екрані. Завдання UI-дизайну полягає в створенні зрозумілих, привабливих і функціональних інтерфейсів, які забезпечують інтуїтивний доступ до функцій системи. UX (User Experience), або «досвід користувача», має ширший зміст і включає в себе всі аспекти взаємодії людини з продуктом або послугою. UX-дизайн зосереджується на емоційних реакціях користувача, його задоволенні, легкості виконання завдань та загальному враженні від використання продукту. Ефективний UX забезпечує зрозумілість навігації, передбачуваність поведінки інтерфейсу, швидкий доступ до основних функцій та мінімізацію помилок.

Щоб інтерфейс був не лише привабливим, а й справді зручним для користувачів, у проєктуванні дизайну застосовуються фундаментальні принципи юзабіліті. Евристичні правила Якоба Нільсена, запропоновані ще у 1994 році, і сьогодні залишаються базою для якісного UI/UX-дизайну. Їхня універсальність дозволяє застосовувати ці правила до будь-якого інтерфейсу – від вебсайтів до

десктопних застосунків, зокрема CRM-систем або освітніх платформ, які потребують максимальної зручності для користувачів різних рівнів технічної підготовки.

Рисунок 2.1 – 10 евристичних правил юзабіліті Нільсена

Принципи, зображені на рис. 2.1, стали основою для розробки ефективних і зрозумілих користувацьких інтерфейсів і активно застосовуються в сучасній практиці UI/UX-дизайну. Ці правила допомагають уникнути поширених помилок при проєктуванні інтерфейсів та забезпечують позитивний користувацький досвід, що особливо актуально при розробці CRM-систем для освітніх установ. Тож, нижче наведені саме ці 10 принципів юзабіліті для дизайну інтерфейсів:

1. статус системи має бути зрозумілим,
2. відповідність між системою та реальним світом,
3. контроль і свобода для користувача,
4. послідовність і стандарти,
5. запобігання помилок,
6. розпізнавання замість запам’ятовування,
7. гнучкість та ефективність використання
8. простота та мінімалізм у дизайні,
9. допомога користувачам у розпізнаванні та виправленні помилок,
10. довідка та документація [30].

Далі буде розглянуто детальніше кожен з принципів. Тож перш за все будь-яка система повинна чітко показувати користувачеві, що зараз відбувається. Це принцип видимості статусу системи [30]. Наприклад, коли адміністратор у CRM-системі натискає кнопку збереження даних учня, система має показати прогрес чи повідомлення про успіх, аби уникнути невизначеності. Користувач має відчувати контроль і бути впевненим у своїх діях.

Після цього важливо, щоб система розмовляла мовою користувача. Це друге правило – відповідність реальному світу [30]. Якщо в CRM замість слова «група» буде технічний термін на кшталт «кластер користувачів», навіть найдосвідченіший викладач відчує дискомфорт. Інтерфейс повинен бути зрозумілим не лише програмістам, а й людям без спеціальних технічних знань.

Природно, що під час роботи користувачі помиляються або змінюють свої рішення. Саме тому третій принцип – контроль і свобода, що передбачає можливість скасування дій чи повернення на попередній крок [30]. У випадку CRM це може бути можливість скасувати помилково доданий запис про заняття або швидко видалити непотрібну інформацію.

Четверте правило про послідовність і стандарти допомагає уникнути хаосу в інтерфейсі. Якщо кнопка «Зберегти» функціонує або виглядає по-різному залежно від вікна програми, це дезорієнтує користувача [30]. В CRM-системі послідовність у вигляді однакових форм для додавання різних типів записів (учнів, груп, викладачів) спрощує навчання нових користувачів і знижує ймовірність схибити.

П’яте правило, яке часто недооцінюють – це запобігання помилок. Замість того щоб виводити повідомлення про хибу вже після некоректної дії, краще попередити користувача на етапі введення [30]. Наприклад, якщо адміністратор вводить неправильний формат електронної пошти учня, система повинна одразу підсвітити помилку ще до натискання кнопки «Зберегти».

Шостий принцип – допомога у розпізнаванні замість запам’ятовування, працює як свого роду підтримка для користувача [30]. Наприклад, у списку, що випадає, де міститься інформація про викладачів краще одразу показувати їхні

прізвища й імена, ніж вимагати від користувача пам’ятати та вводити їх вручну. Це полегшує навігацію і мінімізує помилки.

Гнучкість та ефективність використання, що становить сьому евристику, передбачає, що інтерфейс має бути зручним як для початківців, так і для досвідченого користувача [30]. Для новачків у CRM-системі важливо мати підказки і зрозумілий послідовний сценарій дій, а для компетентних – можливість пришвидшити роботу за допомогою гарячих клавіш чи швидких фільтрів.

Восьме правило про те, що естетичний і мінімалістичний дизайн більше, ніж просто про «красивий вигляд». Це про чистоту інтерфейсу, де кожен елемент має чітке призначення, аби надмірна деталізація не ускладнювала сприйняття [30]. У CRM-системі важливо уникати візуального шуму, щоб викладачі чи адміністратори могли сконцентруватися на своїх завданнях, а не витрачати час на пошук потрібної кнопки серед зайвих елементів.

Дев’яте правило наголошує на тому, що коли помилки все ж трапляються, повідомлення про них мають бути чіткими і корисними, аби користувачу одразу все було зрозуміло [30]. Наприклад, замість абстрактного «Помилка 500» система має пояснити: «Не вдалося зберегти запис. Перевірте з’єднання з сервером і спробуйте ще раз». Такий підхід допомагає користувачам швидко виправляти помилки без звернення до технічної підтримки.

Найкраще, якщо система не потребує жодних додаткових пояснень, однак іноді може виникнути потреба в цьому, тож нарешті десяте правило про документацію. Матеріали довідок мають бути легкими для пошуку й зосередженими на завданнях користувача. Інформація повинна бути стислою та містити чітко сформульовані кроки, які потрібно виконати.

У сукупності ці принципи не тільки забезпечують зручність користування системою, а й формують позитивний користувацький досвід, що в свою чергу підвищує ефективність роботи та знижує витрати на навчання персоналу. Саме тому при проєктуванні клієнтської частини CRM-системи для школи іноземних мов їх дотримання є важливим.

На етапі прототипування інтерфейсів ключовим завданням є трансформація концептуальних рішень у візуальні макети, які дозволяють перевірити ідеї ще до етапу реалізації в коді. Для цього застосовуються спеціалізовані інструменти прототипування, які забезпечують можливість швидкого створення інтерактивних моделей інтерфейсів. Тож далі будуть розглянуті найбільш популярні інструменти, які широко використовуються в індустрії для проєктування UI/UX, а саме: Adobe XD, Sketch, Figma, Lunacy та Penpot.

Adobe XD – це високопродуктивний інструмент, розроблений спеціально для дизайну та прототипування вебзастосунків, мобільних додатків та інших інтерфейсів [31]. Цей програмний продукт надає широкий спектр можливостей для створення привабливих та функціональних дизайнів, який спрощує процес розробки та спільної роботи з командою. Adobe XD доступний для користувачів Windows та MacOS, що робить його доступним для широкого кола користувачів [32]. Також програма пропонує безкоштовний план, який має деякі обмеження, а саме кількість активних проєктів та співробітників. Для розблокування додаткових функцій і можливостей доступні платні плани з місячною або річною підпискою.

Функціональні можливості Adobe XD:

— надання розширеного набору інструментів для швидкого створення макетів інтерфейсу, що дозволяє легко створювати різноманітні елементи, такі як кнопки, поля вводу, меню тощо;

— створення інтерактивних прототипів, в яких можна додавати переходи між екранами, створювати анімацію та моделювати взаємодію користувача із застосунком;

— забезпечення можливості спільної роботи над проєктами в режимі реального часу, що дозволяє командам швидко обмінюватися ідеями, вносити зміни та спільно вдосконалювати дизайн;

— інтеграція з іншими продуктами Adobe – Photoshop та Illustrator для зручної взаємодії з проєктами між різними програмами Adobe [32].

За допомогою Adobe Photoshop можна доповнити дизайн додатковими графічними елементами. Програма надає широкий набір інструментів для роботи з растровою графікою та фотографіями. Це може бути особливо корисно, якщо необхідно створити унікальні ілюстрації та ефекти або обробити зображення. Крім того, для створення векторних ілюстрацій або графічних елементів з гладкими краями і можливістю масштабування ви можете використовувати Adobe Illustrator. Він дозволяє створювати складні векторні малюнки, такі як логотипи, іконки або ілюстрації, які легко можна використовувати у вебдизайні.

Отже, Adobe XD, Photoshop та Illustrator – це потужні інструменти, які разом доповнюють один одного та дозволяють створювати повноцінний та функціональний дизайн для вебзастосунків та інші проєктів з увагою до деталей. Робота з цими програмами в одній екосистемі Adobe Creative Cloud дозволяє легко переміщуватись між проєктами, обмінюватися ресурсами та спільно працювати з командою [31, 32].

Sketch є відмінним інструментом для дизайну векторної графіки та прототипування інтерфейсів, який набув значної популярності серед дизайнерів за останні роки. Цей програмний продукт розроблений компанією Bohemian Coding та спрямований на роботу з макетами вебсайтів, застосунків та інших цифрових проєктів [33].

Основні функції Sketch:

— інтерфейс програми призначений для максимальної продуктивності дизайнера. Зручні інструменти та інтуїтивний інтерфейс дозволяють швидко створювати і редагувати векторні об'єкти та макети;

— для макетування та дизайну інтерфейсів представлений широкий набір інструментів. Він має вбудовані компоненти, які спрощують створення стандартних елементів дизайну, таких як кнопки, поля вводу тощо [33];

— є базові можливості для створення прототипів, але вони не такі розвинені, як у деяких інших програмах для дизайну, таких як Adobe XD.

Перевагами Sketch є простий та зручний інтерфейс, який сприяє продуктивності, велика кількість розширень та плагінів для збільшення

функціональності та підтримка бібліотек та символів для ефективного управління дизайном [33].

Проте головними недоліками програми є те, що він доступний лише для MacOS, що обмежує доступність для користувачів, які працюють на Linux та Windows, Sketch є платним, хоча має безкоштовний пробний період для тестування, проте для постійного використання потрібно придбати ліцензію, а на останок має обмежену можливість прототипування порівняно з іншими програмами.

Підсумувавши, Sketch є потужним інструментом для дизайну інтерфейсів зі спеціалізованими функціями та простим інтерфейсом. Він може бути особливо корисним для дизайнерів, які працюють на платформі MacOS та шукають зручний інструмент для розробки дизайну для цифрових продуктів.

Figma є вебзастосунком для дизайну та прототипування, який дозволяє користувачам працювати разом над проєктами в реальному часі. Він має подібний до Sketch інтерфейс, але його можна використовувати в браузері і на будь-якій операційній системі та працювати з командою в режимі онлайн. Figma пропонує безкоштовний план для особистого використання та платні плани для команд [34, 35]. Тобто сильними сторонами є:

- кросплатформеність і доступність через браузер;
- спільна робота в реальному часі;
- розширені можливості для дизайну та інтерактивного прототипування;
- безкоштовний тариф для малих команд;
- постійний розвиток і додавання нових функцій, таких як варіанти компонентів, токени дизайну, і режим для розробників.

Для прототипування десктопного застосунку CRM-системи буде використано саме Figma, адже цей сервіс простий і зручний у роботі, сумісний з ОС Windows та надає увесь потрібний функціонал у безкоштовній версії програми. Тому більш детально про функції, можливості та переваги Figma буде розглянуто в наступному підрозділі.

Lunacy – відносно новий сервіс у сфері UI/UX-дизайну, розроблений компанією Icons8. Це безкоштовний графічний редактор для дизайну інтерфейсів, що працює на Windows, Mac і Linux, і може слугувати альтернативою Figma або Sketch [36]. Спочатку Lunacy задумувався як редактор, сумісний зі Sketch-файлами на Windows, аби користувачі Windows могли відкривати та редагувати .sketch файли. Нині ж Lunacy еволюціонував у повноцінний автономний інструмент дизайну з багатьма цікавими можливостями. Він підтримує роботу як офлайн, так і з використанням хмари Icons8 для синхронізації і спільної роботи.

Ключовими особливостями Lunacy є можливість відкриття та експорту проєктів Sketch без конвертації, вбудована бібліотека безкоштовних ресурсів від Icons8 (іконки, ілюстрації, фотографії, генерація тексту та зображень на основі ШІ), а також з версії 8+ – командна робота над дизайном у режимі реального часу. Інтерфейс та концепції Lunacy дуже схожі на Sketch/Figma: є артборди, сторінки, символи (компоненти), стилі. Прототипування в Lunacy наразі базове – звичайне перемикання екранів, без складних анімацій. Зате Lunacy вигідно вирізняється продуктивністю і роботою офлайн – всі файли зберігаються локально або опціонально можна підключити синхронізацію в хмару. Крім того, Lunacy інтегрує деякі функції на базі штучного інтелекту: генерація облич, видалення фонів, автозаповнення текстом, які можуть прискорити роботу дизайнера [36].

Переваги Lunacy:

— безкоштовне використання,

— відсутність потреби в потужному обладнанні,

— кросплатформеність,

— підтримка форматів Sketch,

— наявність вбудованих матеріалів (іконки, ілюстрації) з бібліотек Icons8.

Недоліками є те, що спільна робота і можливості онлайн поки менш допрацьовані, ніж у Figma, також менш розвинута екосистема плагінів і сторонніх інтеграцій. Lunacy може розглядатися, якщо потрібен офлайн-режим або безкоштовне використання без обмежень у кількості проєктів. Втім, наразі з

огляду на популярність Figma та її можливості, Lunacy швидше альтернатива, ніж основний вибір.

Penpot – цікавий сучасний інструмент, що позиціонується як перша повністю відкрита open-source платформа для дизайну і прототипування. Він з'явився у 2021 році і привернув увагу дизайнерів, особливо після новин про покупку Figma компанією Adobe. Penpot працює в браузері і багато в чому нагадує Figma за функціональністю: користувачі можуть створювати векторні дизайни інтерфейсів, компоненти, бібліотеки, а також інтерактивні прототипи на основі цих макетів. Головна відмінність – відкритий код і орієнтація на спільноту розробників. Penpot написаний на відкритих вебстандартах (SVG), має відкритий API для експорту коду, що особливо приваблює front-end розробників. За рахунок open-source моделі Penpot можна розгорнути на власному сервері, отримавши повний контроль над даними – це важливо для організацій, які бажають приватності або кастомізації [36].

Функціональні можливості Penpot:

— спільне редагування в реальному часі (подібно до Figma),

— інтуїтивний UI редактора,

— підтримка компонентів і стилів,

— базове прототипування – створення інтерактивних посилань між екранами.

Розробники підкреслюють, що Penpot дозволяє дизайнерам і розробникам тісніше співпрацювати: дизайнери можуть створювати прототипи та дизайн-системи, а розробники – легко отримувати з Penpot готовий код CSS/SVG, оскільки все побудовано на вебстандартах [36]. Станом на 2025 р., Penpot поки поступається лідерам (Figma, Adobe) через меншу масштабованість: інтерфейс і інструменти трохи менш відшліфовані, прототипування підтримує тільки базові переходи без анімацій, менша спільнота з готовими шаблонами. Однак проєкт стрімко розвивається.

Переваги Penpot:

— безкоштовне використання і відсутність ліцензійних обмежень (необмежена кількість користувачів, файлів тощо),

— можливість адаптації під свої потреби або власноруч дописувати функціонал,

— кросплатформеність

— фокус на інтеграції з процесом розробки.

Недоліками наразі є дещо менший набір функцій і відсутність просунутих можливостей анімації прототипів. Для розробки для бакалаврської роботи Penpot може бути цікавим як альтернатива, але для отримання швидкого і якісного результату, доцільніше обрати більш функціональний і досконалий інструмент - Figma.

2.2 Figma для проєктування інтерфейсу CRM-системи

Figma є одним з найбільш популярних інструментів для UX/UI-дизайну, який широко використовують як малі стартапи, так і великі компанії, такі як Netflix, Zoom, Discord та Stripe. Ця програма з відкритим вихідним кодом доступна на різних платформах і є популярним серед вебдизайнерів, верстальників, UX-дослідників, маркетологів, керівників проєктів та розробників мобільних застосунків, сайтів тощо.

Отже, Figma – це багатоплатформовий онлайн-сервіс для вебдизайну, з допомогою якого можна створювати векторні ілюстрації, інтерактивні дизайни сайтів і мобільних застосунків, а також елементи інтерфейсу [34].

Figma має низку переваг, таких як робота у режимі реального часу, спільна робота, доступність та зручність використання для користувачів. Для більш комфортної роботи в Figma можна створювати компоненти, що спрощує процес створення дизайну та дозволяє повторно використовувати елементи інтерфейсу. Також Figma надає можливість створювати прототипи та презентації проєктів, що дозволяє дизайнерам працювати над функціоналом та взаємодією користувача з макетом до його реалізації. Крім того, Figma не обмежується роботою тільки з векторною графікою, а також підтримує використання

растрових зображень, що є особливо корисним для дизайнерів, які хочуть створювати інтерактивні прототипи з використанням реалістичних зображень.

Головною особливістю сервісу є можливість спільної роботи з іншими розробниками. Figma – це єдиний графічний редактор, у якому можна створити команду і одночасно працювати над проєктом разом з іншими в реальному часі [35]. Ця функція особливо важлива під час роботи над великим завданням. Програма створює єдине місце для всіх доданих користувачів:

— дизайнери створюють деталі продукту в одному вікні;

— верстальники вносять зміни та редагують файл;

— менеджер проєкту в режимі онлайн відстежує кожен етап роботи;

— розробники вносять виправлення та зміни у дизайнерський файл до етапу затвердження клієнтом, щоб оцінити обсяг майбутніх робіт та вносити виправлення для уникнення проблем при реалізації проєкту.

Дизайнери можуть залишати коментарі та ставити запитання всередині вікна програми, що дозволяє спростити процес узгодження та скоротити час, який витрачається на листування в чатах. Коментарі, залишені в програмі, зберігаються в історії файлу, тому не можуть бути втрачені [35].

Також у Figma можна повноцінно працювати не лише в завантаженому на пристрій застосунку, а й і в браузерній версії сервісу [34]. Така можливість додає зручності у використанні, наприклад, для швидкої роботи на іншому пристрої можна скористатись браузером, увійти у свій обліковий запис і внести необхідні редагування. Щоправда одним з недоліків є те, що для обох варіантів використання необхідне постійне інтернет-підключення.

Ще однією перевагою Figma є те, що вона має вбудований хмарний сервіс, який автоматично зберігає макети проєктів і зберігає їх протягом 30 днів. За необхідності можна зайти в історію редагування і повернутись до проєктів, скопіювати і відредагувати їх. Крім того, проєкт можна завантажити як звичайний документ і зберігати його на комп’ютері. Тобто можна відмовитися від сторонніх сервісів на кшталт Google Диска або Dropbox, що є дуже зручною функцією при спільному редагуванні [34, 35].

Варто також зазначити, що у Figma доступні як платний тариф, так і безкоштовний. У функціональність останнього входять:

– можливість працювати лише над трьома проєктами одночасно;

– до роботи над проєктом може бути залучено не більше одного редактора;

– проєкти, збережені в історії версій, доступні протягом 30 днів [35].

Незважаючи на цей невеликий перелік обмежень, у безкоштовній версії доступні усі інструменти і функціональні можливості. Тому, якщо немає потреби одночасної роботи над великою кількістю проєктів і залучення до редагування більш, ніж одну людину, то безкоштовної версії буде цілком достатньо.

Професійний тариф розрахований на командне редагування і дає можливість працювати над будь-якою кількістю проєктів. Вартість – $12 на місяць за кожного редактора. Обмежень на зберігання історії проєктів немає, до того ж всі учасники можуть працювати із загальною бібліотекою елементів. Підписка для організацій буде коштувати $45 за кожного редактора. Вона надає всі можливості професійного тариф, а також додаткові інструменти для командної роботи, системну аналітику, доступ до спеціальних плагінів і поліпшену безпеку [35].

2.3 Розробка дизайну та прототипу інтерфейсу користувача

Розробка клієнтської частини CRM-системи для школи іноземних мов передбачає створення інтуїтивно зрозумілого, функціонального та естетично привабливого інтерфейсу користувача. Цей підрозділ присвячено процесу проєктування дизайну та створення інтерактивного прототипу інтерфейсу з використанням інструменту Figma, який візуалізує логіку взаємодії користувача із системою, забезпечує чітке уявлення про розташування основних функціональних блоків та дозволяє провести первинне тестування зручності користування.

У процесі розробки інтерфейсу було враховано ключові принципи сучасного UI/UX-дизайну, які розглянуто у підрозділі 2.1 та які забезпечують ефективність, зручність та привабливість системи для кінцевих користувачів,

зокрема: консистентності стилів, візуальну ієрархію, доступність, простоту використання та гнучкість у роботі з великою кількістю функцій. Система передбачає багатофункціональне середовище для адміністраторів і викладачів, яке дозволяє оперативно працювати з даними, вести внутрішню комунікацію, здійснювати фінансовий контроль та організацію навчального процесу.

Для візуального оформлення інтерфейсу було обрано палітру, яка зображена на рис. 2.2 і базується на поєднанні відтінків зеленого та сірого кольорів. Основний акцент зроблено на насиченому, але приємному за сприйняттям, зеленому кольорі у поєднанні з нейтральними сірими відтінками. Такий вибір кольорів ґрунтується на психології кольору та особливостях сприйняття інтерфейсу.

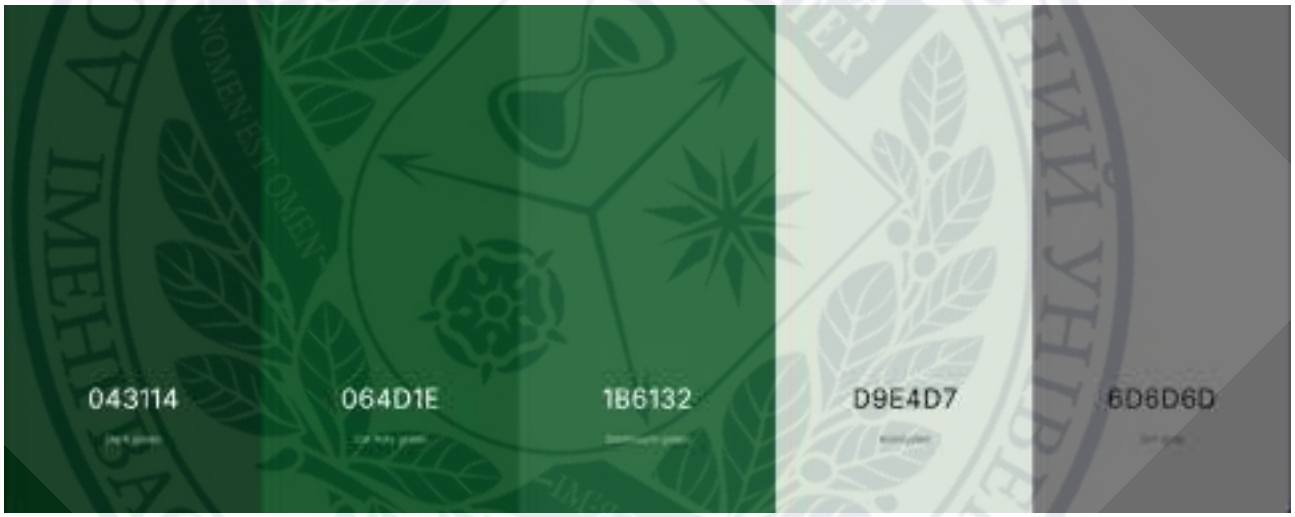

Рисунок 2.2 – Кольорова гама для дизайну світлої теми застосунку

Зелений колір асоціюється зі свіжістю, розвитком, стабільністю та навчанням. У контексті школи іноземних мов це символізує зростання знань, нові можливості, довіру до системи. Зелений також чинить заспокійливу дію на користувача, що особливо важливо для інтерфейсів із великою кількістю інформації. Сірі відтінки виступають як фон та допоміжні кольори, які не конкурують з основними елементами, а навпаки – підкреслюють їх. Сірий додає стриманості, офіційності та професійного вигляду, що добре відповідає бізнес-призначенню CRM-системи. Комбінація зеленого і сірого дозволяє досягти балансу між функціональністю та естетикою. Система не виглядає занадто

яскравою чи перевантаженою, і водночас має виразні акценти, що полегшують орієнтацію у великій кількості функцій.

Також обрано палітру і для темної теми, яка зображена на рис. 2.3. Обидві теми демонструють хорошу контрастність, відповідність стандартам доступності (WCAG) та збереження зорового комфорту при тривалому використанні. Усі інтерактивні елементи, такі як кнопки, активні вкладки чи індикатори статусів, оформлені з використанням зеленого кольору для чіткої візуальної ідентифікації. Неактивні елементи, навпаки, мають сіре оформлення, що створює зрозумілу ієрархію елементів керування.

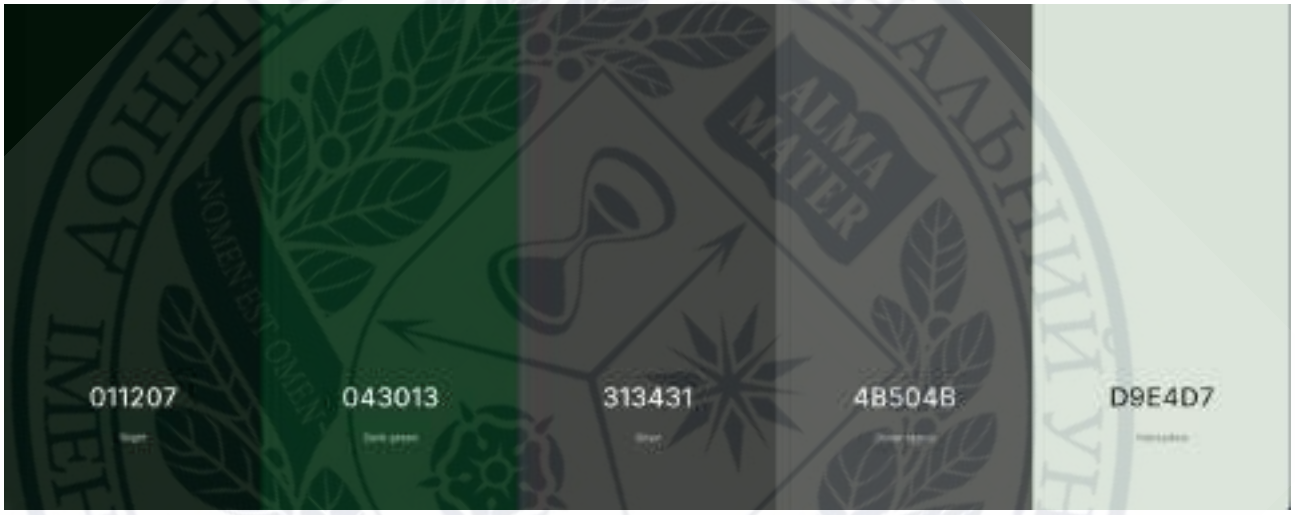

Рисунок 2.3 – Кольорова гама для дизайну темної теми застосунку

Для оформлення всього тексту використано шрифт «Alata Regular», що зображений на рис. 2.4.

Рисунок 2.4 – Шрифт тексту застосунку

Тож далі можна перейти до поетапного створення дизайну інтерфейсу десктопного застосунку. У межах CRM-системи кожне вікно виконує окрему функціональну роль, що утворює повноцінну логічну структуру взаємопов’язаних компонентів.

Рисунок 2.5 – Вікно авторизації

Спочатку розроблене вікно для авторизації, яке забезпечує перший контакт користувача із системою (рис. 2.5). Дизайн цього екрана максимально простий та мінімалістичний, аби взаємодія була інтуїтивною, а зайві деталі не заважали. Поля для введення логіна і пароля супроводжуються підписами. Є окрема кнопка для реєстрації, що дозволяє створити обліковий запис для нових користувачів. Відповідно вигляд вікна реєстрації зображено на рис. 2.6.

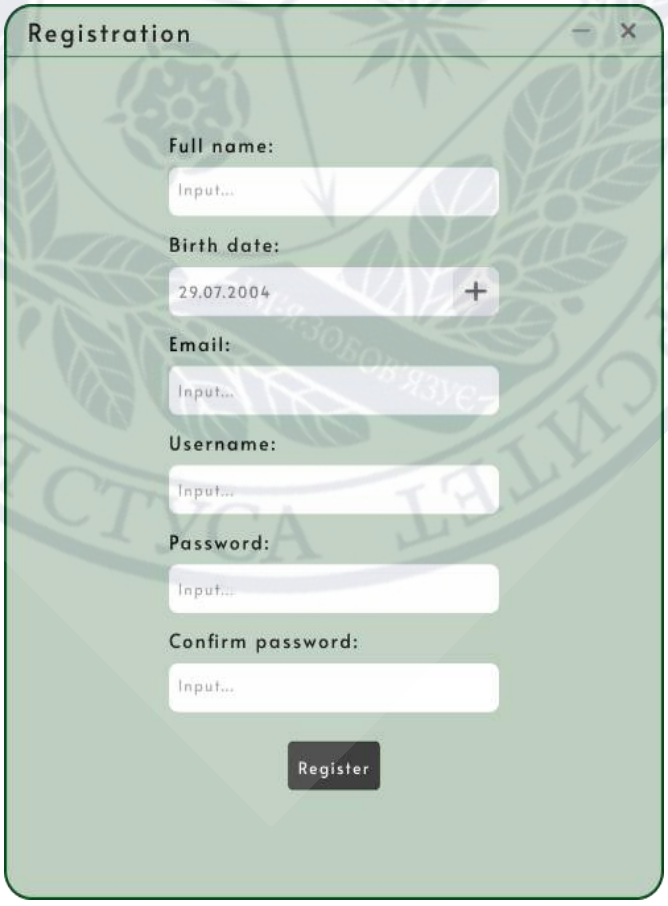

Рисунок 2.6 – Вікно реєстрації

Після успішного входу користувач потрапляє на головну сторінку, яка виконує роль інформаційного дашборду (рис. 2.7). У вікні відображаються ключові показники системи: актуальна кількість студентів і працівників, прибуток школи, графік візуалізації тренду чисельності учнів кожного місяця, а також найближчі заплановані події. Візуальна композиція дотримується принципів зорової ієрархії: найважливіша інформація розміщена у верхній частині сторінки та виділена акцентними кольорами.

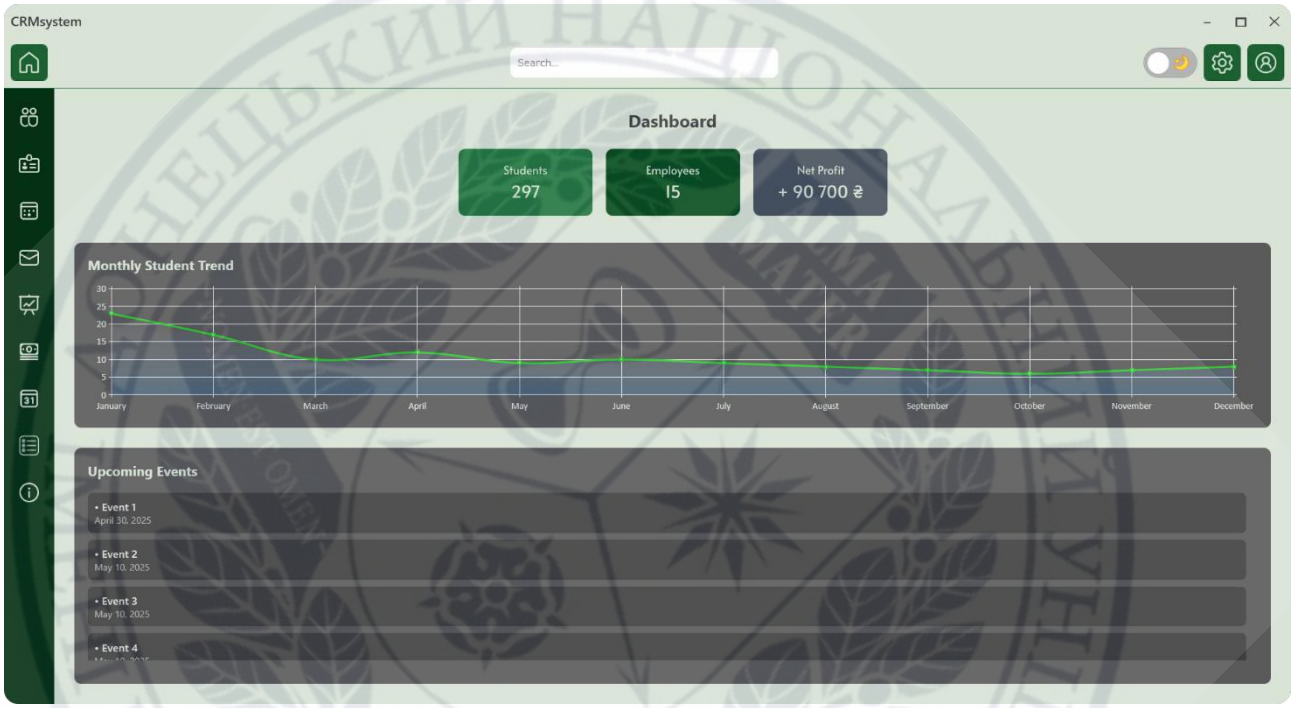

Рисунок 2.7 – Головна сторінка із базовими елементами інтерфейсу

Також, як можна побачити з рис. 2.7, макет цієї сторінки є базовим і для всіх інших, він містить такі елементи інтерфейсу:

— ліворуч у верхньому куті розташований текстовий логотип програми для візуалізації ідентичності системи;

— по центру зверху – пошуковий рядок для доступу до необхідного у всій системі;

— праворуч зверху звичайні кнопки згортання/розгортання, збільшення/зменшення та закриття програми;

— нижче кнопка перемикання теми, яка дозволяє обрати світлу або темну візуальну тему, яка автоматично застосовується до всіх вікон інтерфейсу;

— поруч кнопка налаштувань відкриває меню загальних параметрів, поки таких як: мова інтерфейсу та звукові сповіщення;

— далі кнопка профілю із меню, яке з’являється як діалогове вікно і включає усю інформацію про користувача система з можливостями додавання фото профілю, зміни даних і пароля, збереження змін, а також виходу з системи, як показано на рис. 2.8;

— з лівого боку інтерфейсу знаходиться бічна навігаційна панель, яка є основним інструментом перемикання між вікнами системи. Вона фіксована, адаптивна до роздільної здатності екрану і містить піктограми та текстові назви відповідних розділів, що зручно для навігації.

Рисунок 2.8 – Діалогове вікно профілю користувача

Сторінки управління даними студентів та працівників є ключовим інструментом для адміністрації школи (рис. 2.9). Таблична структура із підтримкою сортування, фільтрації та пошуку дозволяє швидко знаходити необхідні записи. Вікна про учнів та співробітників мають майже ідентичне представлення, тому вигляд сторінки буде показано на прикладі з інформацією про клієнтів. Власне, у цих списках міститься базова інформація про студентів школи: ПІБ, дата народження, електронна пошта, дата підписання договору,

номер договору, чи є оплата за поточний місяць. У списках про працівників також мітяться ПІБ, дата народження, електронна пошта, а також посада та розмір заробітної плати.

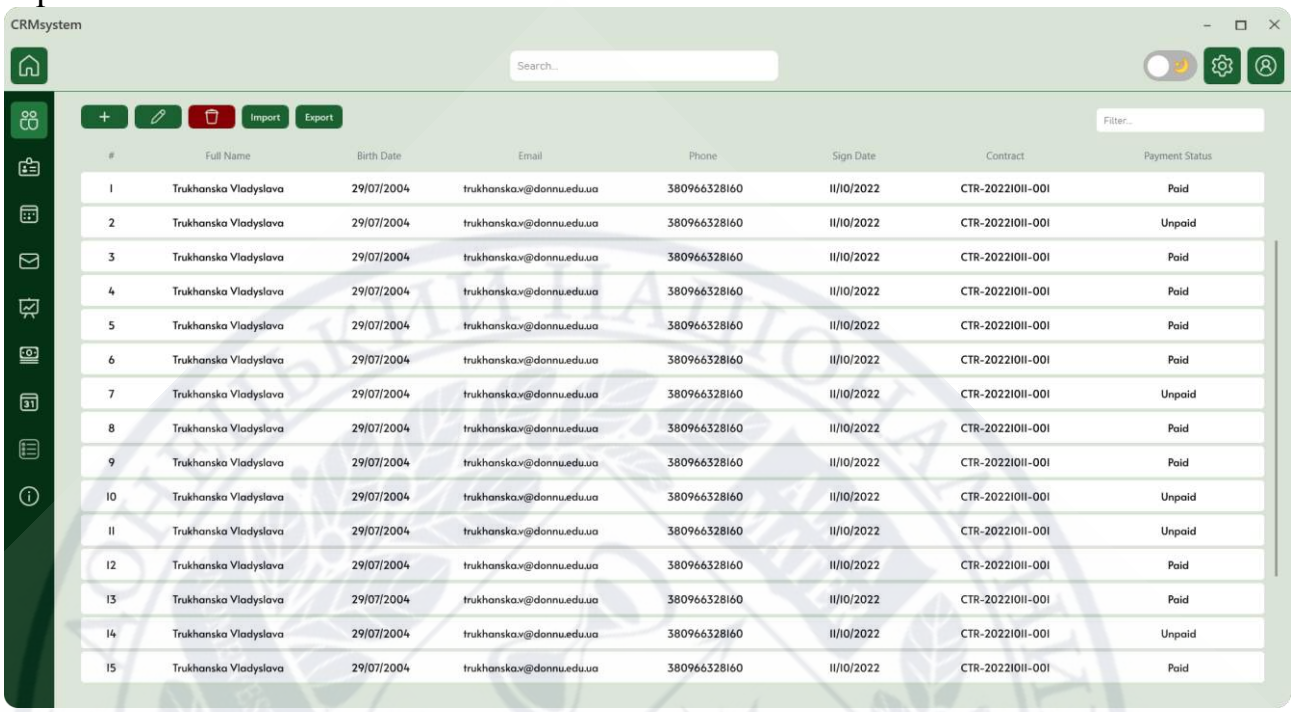

Рисунок 2.9 – Вікно із тестовим загальним списком студентів школи

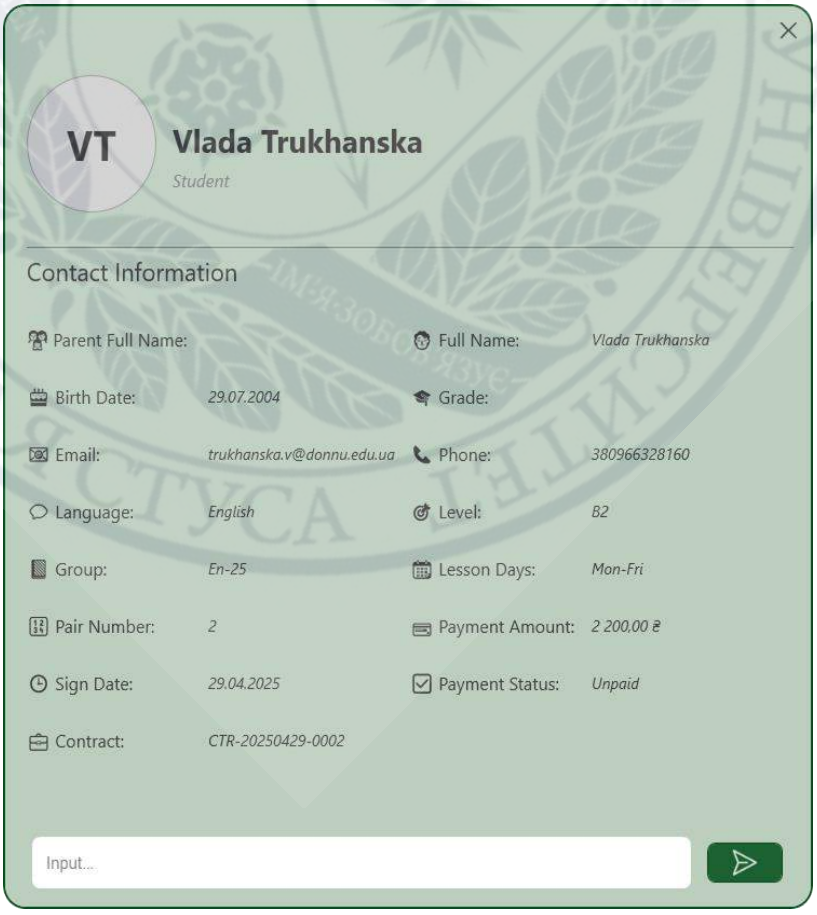

Рисунок 2.10 – Діалогове вікно картки студента

Для перегляду детальної інформації про студента або співробітника реалізоване діалогове вікно у вигляді картки (на рис. 2.10, власне, про студента), яка відображається, клікнувши на потрібний рядок. Також у вікні є можливість відправки повідомлення на вказану електронну пошту, що зручно для нагадування, наприклад, про оплату або заняття.

Сторінка розкладу занять візуалізує навчальний графік. Розклад подається у форматі розподілу груп за класами у школі, днями і часом занять. Кожному з викладачів присвоєний свій колір, відповідно кожна з груп у розкладі виділена тим кольором, коли викладач проводить заняття. У розклад можна як додавати нові заняття, так і видаляти їх. Представлення цієї сторінки зображено на рис. 2.11.

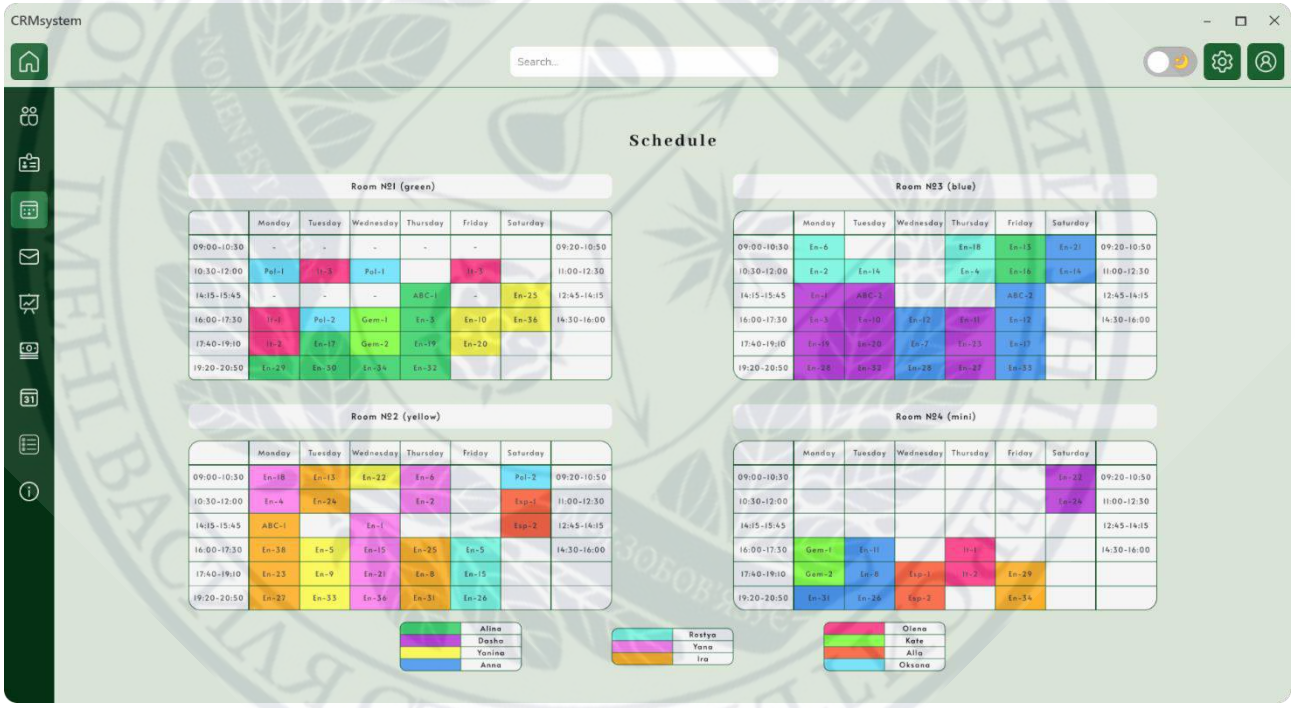

Рисунок 2.11 – Вікно розкладу занять

Сторінка з повідомленнями відображає вже відправлені листи на пошту, чи то студентам школи, чи то співробітникам. Інтерфейс сторінки має таку ж структуру, як і вікна з списками студентів і працівників – також у табличному представленні, з колонками, в яких міститься інформація листа: про тему, отримувача, скорочений основний текст, дату відправки та пошти, з якої надіслано повідомлення. Розгорнутий основний текст можна переглянути у діалоговому вікні, клікнувши на потрібний рядок.

Рисунок 2.12 – Вікно із тестовими даними фінансової звітності

Сторінка фінансової звітності на рис. 2.11 дозволяє керівництву переглядати узагальнені дані про фінансову діяльність школи: очікувані та дійсні доходи від навчання, витрати на заробітні плати працівникам і дохід школи. Інтерфейс оформлений у вигляді інтерактивних графіків цих даних за поточний місяць.

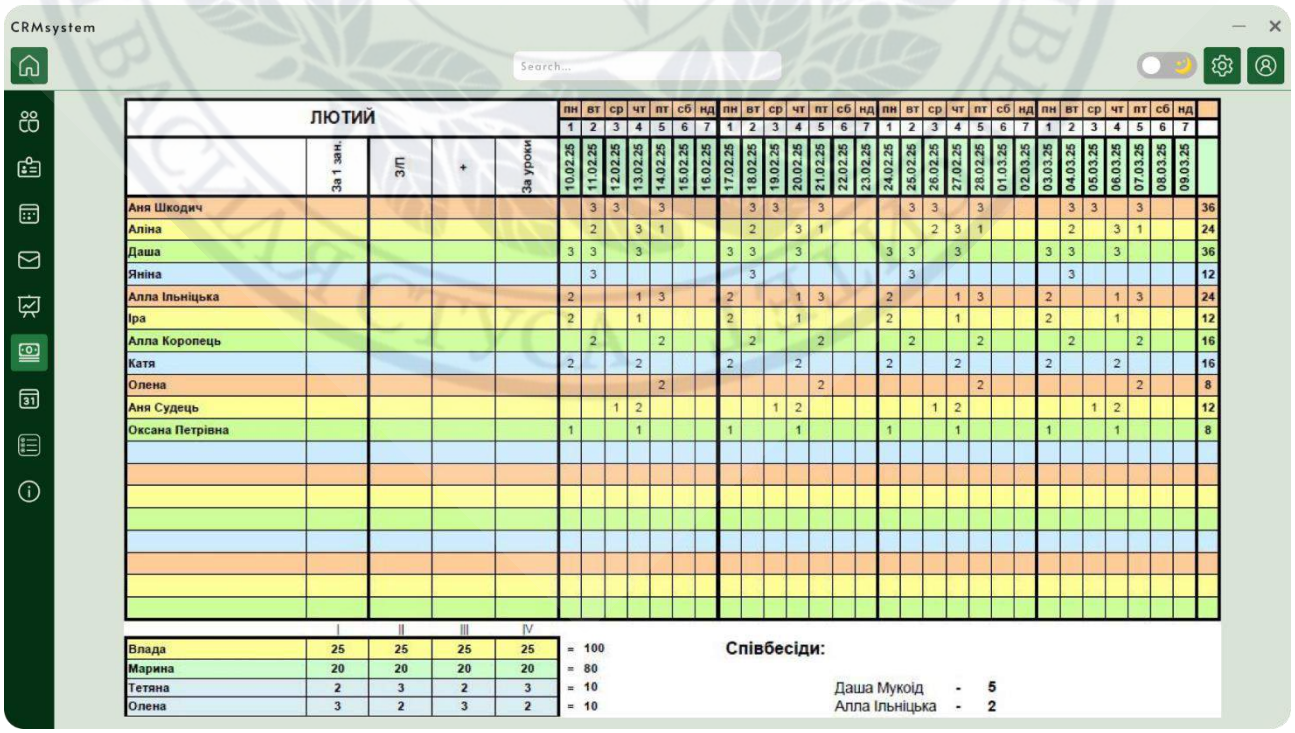

Рисунок 2.13 – Вікно із тестовими даними про заробітну плату працівників

На рис. 2.13 показано сторінку зарплатного листа, який містить інформацію про кількість проведених занять, вартість 1 заняття, бонуси і суму заробітної плати. Кожен запис про викладачів оформлено в табличному вигляді для простоти сприйняття і розрахунків. Також нижче є менша таблиця з іншими співробітниками, в яких рахуються години роботи. А праворуч кількість проведених співбесід викладачів з новими клієнтами.

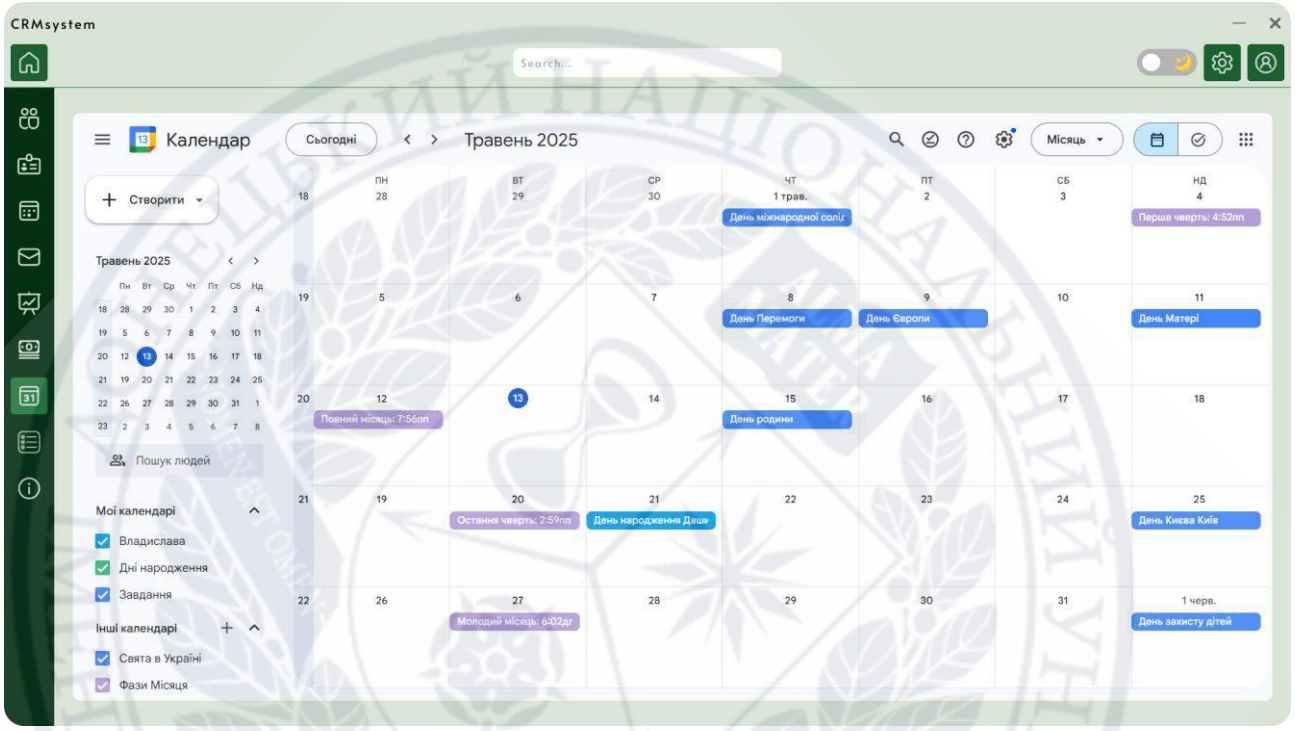

Рисунок 2.14 – Вікно календаря

Сторінка календаря, зображена на рис. 2.14, виконує функцію персонального та командного планувальника. Події відображаються у вигляді щоденного/тижневого/місячного огляду. Користувач може додавати нагадування, ділитися подіями, встановлювати дедлайни. На сторінці відображається календар від компанії Google, оскільки він функціональний та зручний для спільної роботи.

Сторінка таск-менеджера призначена для створення, позначення виконання і моніторингу завдань (рис. 2.15). Задачі можна виділити за важливістю, а також відсортувати за пріоритетністю, датою дедлайну, за алфавітом і за датою створення. Виконані завдання зберігаються у рядку «Completed», а нові задачі можна додати знизу сторінки.

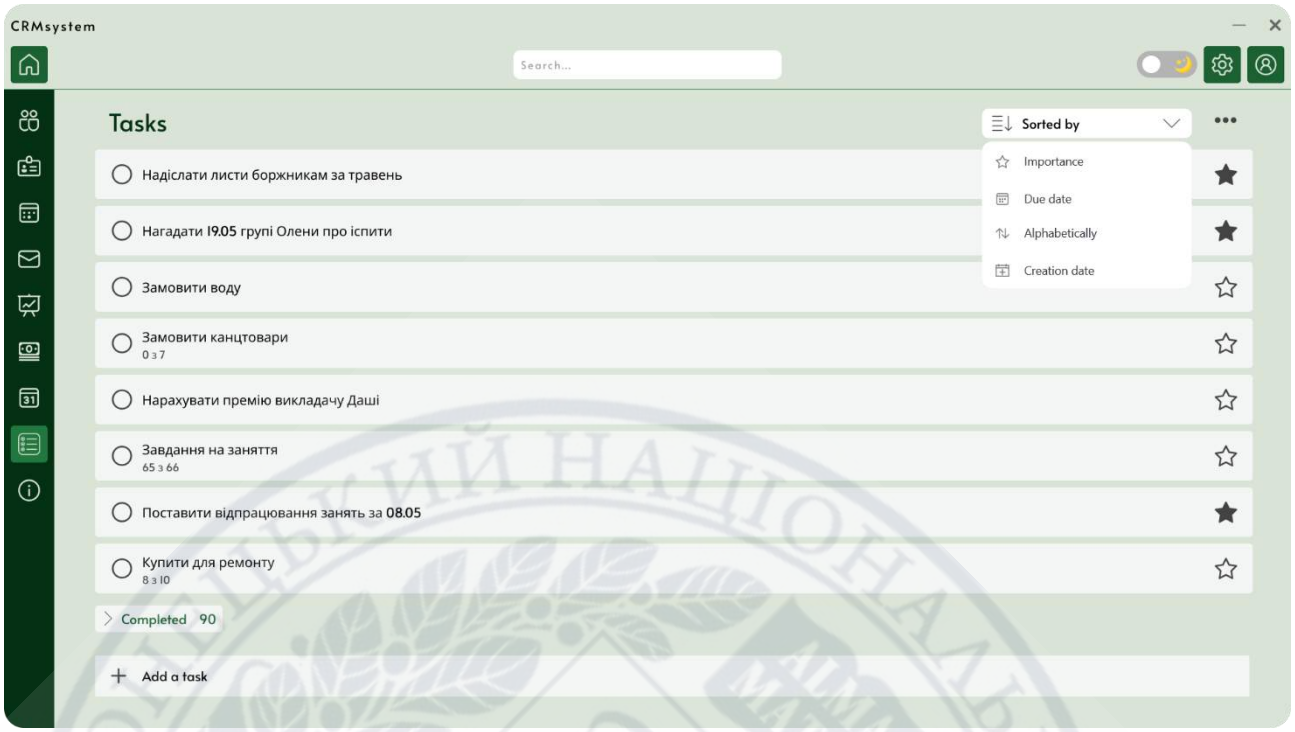

Рисунок 2.15 – Вікно для завдань

Остання сторінка «Про систему» (елемент вікна відображено на рис. 2.16) містить статичну інформацію: опис системи, інформація про розробників, версія застосунку, електронна пошта служби підтримки, посилання на документацію на сайті. Використано простий, легкий дизайн з акцентом на зручність читання. Також для уніфікації застосунку на сторінці міститься логотип системи, виконаний у зеленому кольорі із елементами, які асоціюються з призначенням системи.

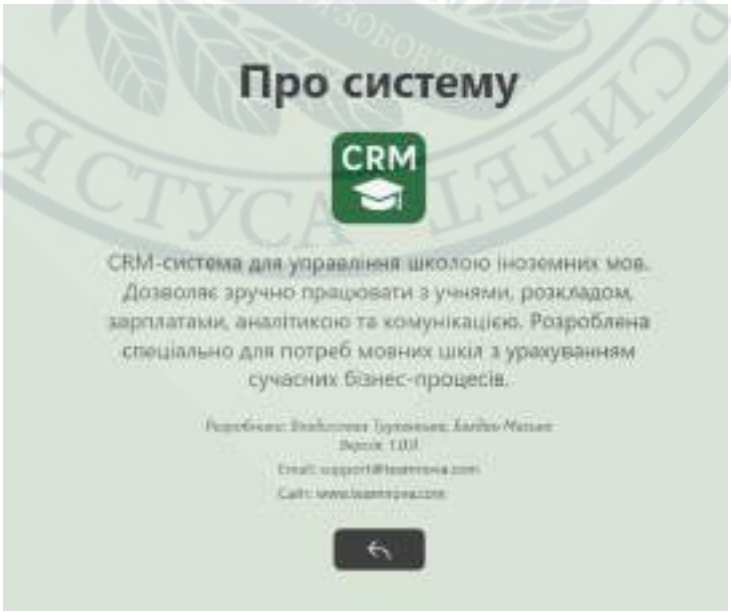

Рисунок 2.16 – Елемент вікна «Про систему»

Дизайн сторінок і вікон застосунку в темній темі представлено в додатку А.

Розроблений дизайн і логічна структура інтерфейсу CRM-системи відповідають сучасним вимогам корпоративного програмного забезпечення, що орієнтоване на щоденну взаємодію великої кількості користувачів. Розроблений у Figma прототип забезпечує візуальну чіткість, підтримує принципи адаптивності та закладає ефективну основу для подальшої реалізації у середовищі WPF із використанням Code Behind. Завдяки структурованій побудові вікон та інтуїтивній навігації система надає комфортне середовище для виконання повного спектра адміністративних, навчальних і управлінських функцій у межах школи іноземних мов.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ CRM-СИСТЕМИ

3.1 Загальна структура системи

Процес розробки клієнтської частини CRM-системи для школи іноземних не можливий без попереднього етапу проєктування структури застосунку, що включає визначення архітектурної моделі, основних функціональних блоків, навігаційної логіки, а також загального життєвого циклу взаємодії користувача з програмним забезпеченням.

Клієнтська частина CRM-системи реалізована як десктопний застосунок на платформі .NET з використанням WPF [37]. Загальна архітектура застосунку побудована за принципом багатошаровості, що забезпечує розподіл відповідальностей між інтерфейсом користувача, сервісною логікою та доступом до даних. Розробка інтерфейсу на основі WPF дозволяє створити гнучку, масштабовану і візуально привабливу систему з чітким розмежуванням візуальної частини (XAML) і логіки обробки подій (Code Behind на мові C#) [38]. Організація застосунку побудована так, щоб забезпечити достатню модульність і логічну ізольованість функціональних блоків, необхідних для подальшої підтримки і масштабування системи. Структура проєкту містить окремі каталоги для моделей даних (Entities/Models), сервісів (Services), ресурсів стилів (Styles, Themes) та представлень (Views), що ілюструє чітке розділення модулів за функціональним призначенням.

Загальна будова десктопного застосунку включає логіку переходу між основними вікнами, організацію вкладок у межах головного інтерфейсу та механізми запуску і завершення роботи програми. Життєвий цикл застосунку починається із запуску процесу, після чого завантажується вікно авторизації користувача. На цьому етапі користувач має можливість увійти в систему (ввести свої облікові дані) або пройти реєстрацію (створити новий обліковий запис). Авторизація перевіряє облікові дані через серверне API та, у разі успіху, ініціалізує роботу основного вікна застосунку. У випадку нового користувача –

реєстрація також здійснюється через відповідний запит до API, після чого можна виконати вхід. Тож, початкова стадія життєвого циклу охоплює авторизацію/реєстрацію користувача, що є необхідною передумовою доступу до основних можливостей системи.

Після успішної аутентифікації відбувається перехід до головного вікна, яке виступає центральною точкою навігації застосунку. Це вікно виступає основним контейнером інтерфейсу, яке побудовано з урахуванням принципів зручної навігації, і містить панель вкладок або кнопок, що відкривають відповідні підмодулі системи, тобто відображаються всі основні розділи CRM-системи. Користувачу доступне головне меню або панель навігації з вкладками, кожна з яких відповідає певному функціональному модулю системи. Зокрема, передбачено наступні вкладки основного вікна:

- Профіль акаунта, де відображається особиста інформація поточного користувача, статус ролі в системі (адміністратор, викладач, директор тощо), а також передбачена можливість редагування даних профілю та кнопка для розлогінення.
- Адміністрування студентами, яке дає змогу переглядати, додавати, редагувати та видаляти записи про студентів. Вікно реалізовано з використанням DataGrid, із підтримкою CRUD-операцій та валідації введених даних [39].
- Адміністрування працівниками, де аналогічним чином реалізовані функції керування інформацією про викладачів, адміністраторів і персонал.
- Розклад, де показано групові та індивідуальні заняття з можливістю додавання, редагування або видалення уроків, а також перегляду детальної інформації про заняття: дата, час, клас, викладач, номер групи.
- Повідомлення для перегляду історії листів адміністрації до студентів або викладачів школи. Передбачено реалізацію списку повідомлень із коротким описом та можливістю відкриття повного тексту повідомлення.
- Фінансова звітність – вкладка з можливістю перегляду ключових фінансових метрик: прибуток, витрати, кількість оплат за курси, динаміка прибутковості.

- Зарплатний лист – розділ обліку заробітної плати співробітників. Тут здійснюється розрахунок заробітної плати викладачів на основі визначених показників: ставки, кількості проведених занять тощо, та ведеться історія виплат. Зміни у цій вкладці можуть вносити лише адміністратори, а для інших користувачів системи доступний лише перегляд.
- Календар, який дозволяє відображати і вносити інформацію про важливі дати, події та нагадування – заплановані співбесіди, заходи школи, дедлайни завдань тощо. Цей модуль є узагальненою формою організації часу для всіх.
- Таск-менеджер – модуль управління завданнями та нагадуваннями. Користувач може створювати особисті нагадування або завдання, встановлювати дедлайни та пріоритети. Передбачено статуси виконання, дедлайни, нотатки та повідомлення-нагадування.
- «Про систему», що містить відомості про опис системи, інформацію про розробників, версію застосунку, електронну пошта служби підтримки, посилання на документацію на сайті. Цей розділ призначений для надання довідкової інформації про сам застосунок.

Всі перелічені вкладки доступні в межах головного вікна після авторизації. Користувач може в довільному порядку переключатися між цими розділами, виконувати необхідні операції у відповідних модулях. Інтерфейс забезпечує навігацію між вкладками без необхідності відкривати окремі вікна для кожного розділу – основні дії здійснюються в межах єдиного вікна, що спрощує роботу користувача та забезпечує цілісний досвід використання системи.

Життєвий цикл завершується, коли користувач завершує роботу із системою. Для цього в розділі «Профіль» передбачено кнопку «Вихід», яка ініціює процес виходу з облікового запису та закриття програми. При натисканні на цю кнопку застосунок виконує необхідні дії для завершення сеансу і закриває головне вікно. Тож, «Вихід» – фактично означає завершення роботи користувача із CRM-системою і є кінцевою точкою життєвого циклу застосунку. Після цього при наступному запуску програми цикл починається знову з етапу авторизації.

Увесь життєвий цикл зображено на рис. 3.1.

Рисунок 3.1 – Життєвий цикл програми

У логічному плані застосунок можна поділити на три основні підсистеми:

1. Підсистема автентифікації, що відповідає за безпеку входу та реєстрації. Дані користувачів передаються на сервер через RESTful API із використанням JWT-токенів, які зберігаються локально в оперативній пам’яті [41, 42].

2. Основна інтерфейсна підсистема, яка відображає відповідні сторінки після автентифікації. Кожне вікно реалізоване як окремий XAML-файл із відповідним класом Code Behind. Навігація здійснюється через механізми Frame та Page, що зберігає контекст користувача та його права доступу.

3. Підсистема взаємодії з сервером, яка реалізована через HTTP-запити із застосуванням бібліотеки HttpClient. Для кожної операції передбачено серіалізацію/десеріалізацію JSON-об’єктів через Newtonsoft.Json.

Отже, загальна структура десктопного застосунку передбачає послідовність етапів «Авторизація → Головне вікно з вкладками → Вихід», де головне вікно містить усі основні підсистеми для виконання бізнес-функцій CRM-системи школи. Такий підхід забезпечує зручність для користувача, оскільки всі інструменти доступні в одному місці, та підтримує логічну

впорядкованість роботи програми. Лістинг структури проєкту (папки Views, Models, Services тощо) підтверджує модульність системи, де кожен компонент відповідає за свою ділянку функціональності. Взаємодія між компонентами здійснюється через чітко визначені інтерфейси і методи, про які детальніше йтиметься в підрозділі 3.3.

3.2 Розробка інтерфейсу системи

Інтерфейс користувача (UI) десктопного застосунку розроблено з використанням технології WPF, що дозволяє створювати сучасні та візуально привабливі інтерфейси із застосуванням розмітки XAML. Основними вимогами до інтерфейсу є зручність користування, інтуїтивно зрозуміла навігація між розділами, уніфікований стиль оформлення та динамічне відображення інформації з бази даних (через виклики до API).

Загальна структура головного вікна містить елементи навігації, які реалізують переключення між вкладками розділів, перелічених у підрозділі 3.1. Розмітка HomeView.xaml визначає основний каркас інтерфейсу: верхню панель із заголовком застосунку та кнопками домашньої головної сторінки, зміни теми, налаштувань та профілю акаунта користувача, бічне меню вкладок, а також область вмісту, де відображається інформація обраної вкладки. Для реалізації вікон у проєкті використано компонент TabControl, кожна вкладка якого (TabItem) містить уміст відповідного модуля. Наприклад, один TabItem має заголовок «Профіль» і вміщує елементи управління профілем, інший – «Студенти» із вмістом StudentsView тощо.

Важливою частиною розробки головного вікна було забезпечення єдиного стилю для всіх вкладок. У проєкті визначено ресурсні словники XAML, в яких описані стилі для типових елементів управління – кнопок, полів введення, таблиць, тощо). Завдяки цьому всі вкладки мають спільні кольорові схеми, шрифти та поведінку елементів, наприклад, кнопки однакового розміру з єдиним стилем по всій програмі. Головне вікно підключає ці ресурси через атрибути ResourceDictionary безпосередньо в HomeView.xaml, тому стилі застосовуються

автоматично до всіх підпорядкованих елементів. Програмний код розробки інтерфейсу головного вікна та його елементів представлено в додатку Б.

Як приклад реалізації конкретного розділу інтерфейсу буде розглянуто модуль управління студентами, детальний програмний код реалізації якого міститься в додатку В.

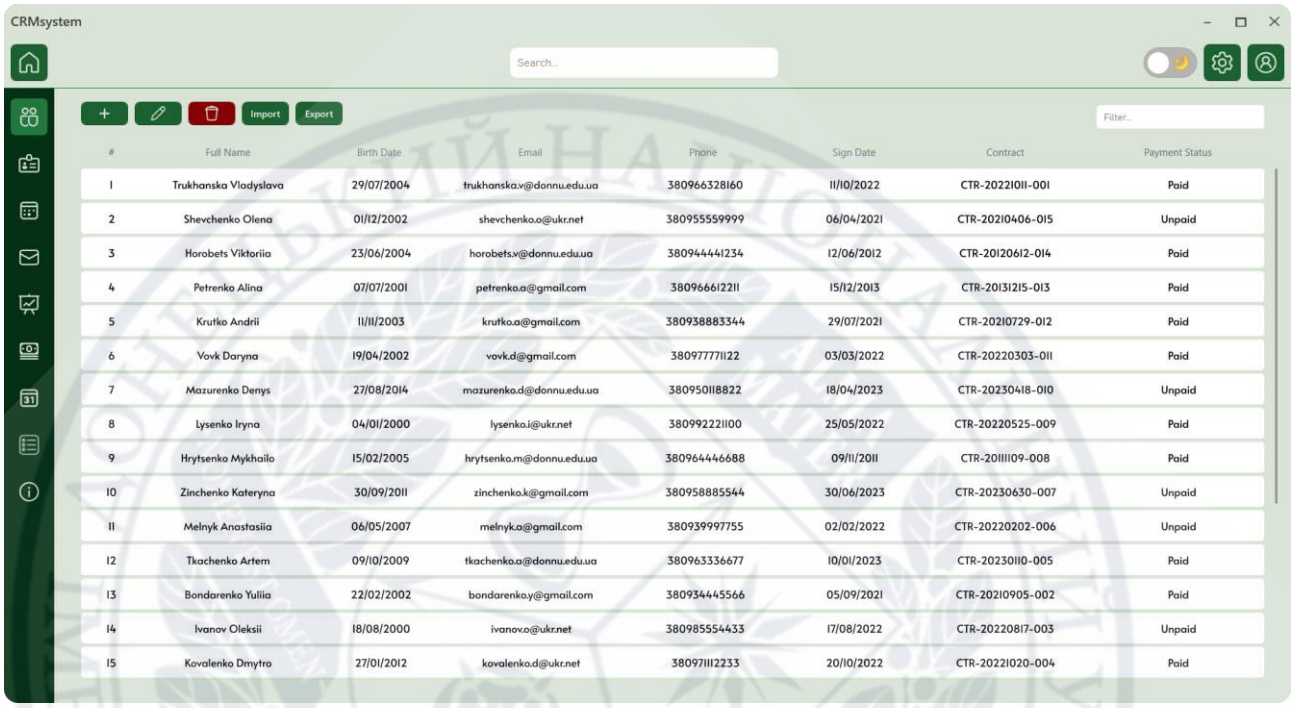

Рисунок 3.2 – Приклад екранної форми зі списком студентів

Тож, вікно StudentsView відображає список студентів та надає засоби для керування цими даними. Структура StudentsView.xaml умовно поділяється на дві області: область списку студентів та область деталей/форм редагування. Для відображення списку студентів використовується елемент DataGrid, що дозволяє показувати табличні дані з можливістю сортування і прокрутки. Кожен рядок DataGrid відповідає окремому студенту і відображає основні поля: ID або № запису, ім’я, прізвище, дату народження, контактні дані, приналежність до навчальної групи, рівень володіння мовою тощо. Стовпці DataGrid прив’язані до відповідних властивостей моделі даних студента. Прив’язка здійснюється через властивість ItemsSource, яка вказує на колекцію студентів, отриману з бази даних через сервіс StudentService. У результаті при завантаженні вкладки викликається метод, що отримує список студентів з API, заповнює ObservableCollection<Student> у ViewModel, а DataGrid автоматично відображає

ці дані. На рис. 3.2 показано приклад інтерфейсу зі списком студентів у системі (табличне представлення даних студента з декількома стовпцями), панеллю керування записами студентів та полем для фільтрації зверху.

Інтерфейс відображає таблицю студентів із такими полями, як: ім’я та прізвище, дата народження електронна пошта, номер телефону та ін. Як видно з наведеної екранної форми, над таблицею розташовані кнопки для додавання нових студентів, редагування чи видалення інформації про вже створених. Дані кнопки реалізують основні дії над списком:

- при натисканні на «Додати» відкривається форма для введення нових даних студента,
- «Редагувати» – дозволяє змінити дані обраного у списку студента,
- «Видалити» – запитує підтвердження та видаляє запис.

Також є кнопки, які передбачають можливість як імпорту, так і експорту даних з файлів з розширеннями .csv, .json і .excel, що буде детальніше розглянуто в підрозділі 3.3 про функціональні можливості.

У StudentsView.xaml ці кнопки реалізовані через елементи <Button> зі встановленими обробниками команд або подій, прив’язаними до команд у ViewModel (реалізовано через ICommand). Наприклад, кнопка «Додати студента» прив’язана до команди AddStudentCommand, яка визначена у StudentViewModel і при виконанні відкриває діалог додавання.

Форма додавання/редагування студента, як правило, реалізована у вигляді діалогового вікна StudentEditView, що відображається поверх основного та який відкривається із StudentsView при додаванні нового чи редагуванні вже створеного студента. Цей діалог містить набір полів вводу для властивостей студента: ім’я, прізвище, дата народження, контактний телефон, email, група тощо (з вибіркою через DatePicker). Кожне поле вводу прив’язане до тимчасової моделі, де зберігаються значення, які вводить користувач. Приклад вигляду такої форми наведено на рис. 3.3.

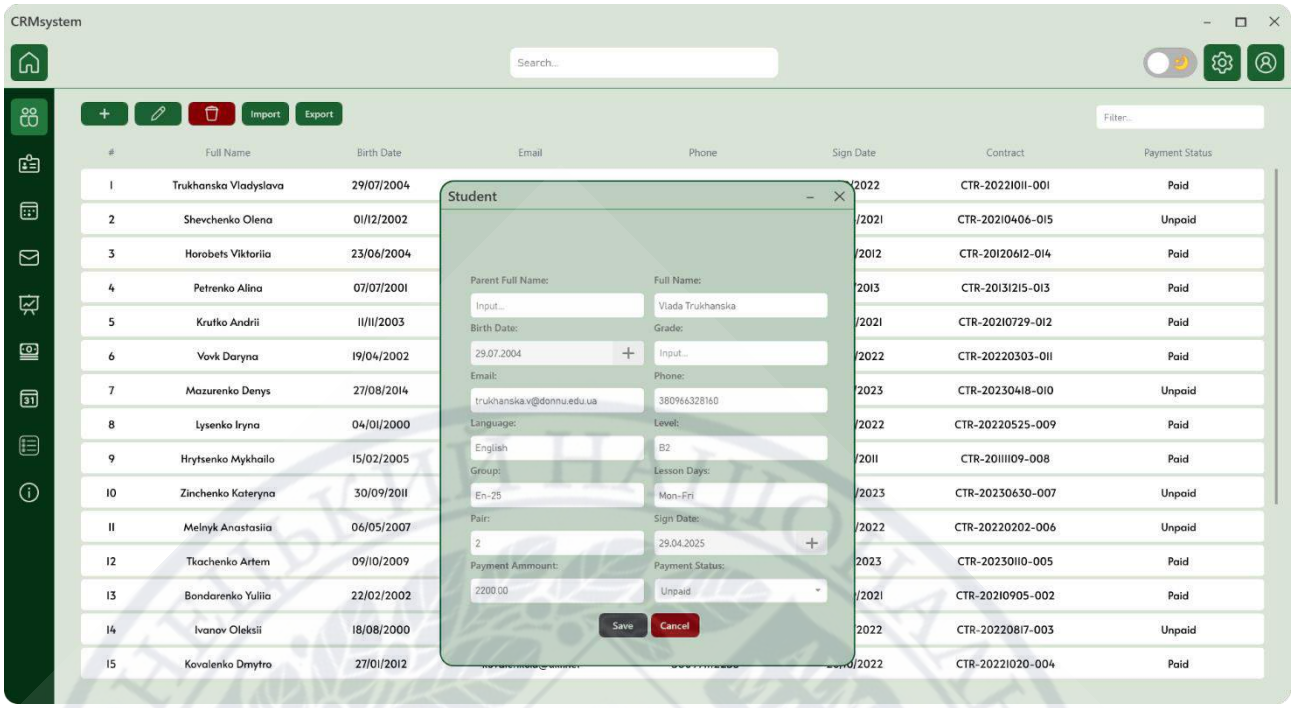

Рисунок 3.3 – Форма введення/редагування даних студента

У цьому діалоговому вікні реалізовано валідацію введених даних: напряму під час вводу або при спробі збереження програма перевіряє коректність даних. Наприклад, поля «Ім’я» та «Прізвище» не можуть бути порожніми – у XAML це задано через атрибути перевірки ValidationRules або через атрибути моделі DataAnnotations із відповідним відображенням помилки. Поле «Email» повинно відповідати формату електронної пошти; дата народження – бути логічною тощо. Якщо якесь із полів містить некоректні дані, інтерфейс виділяє це поле червоною рамкою і вказує на помилки. У результаті досягається логіка валідації на боці клієнта, що попереджає відправку заздалегідь некоректних даних на сервер.

Стилістично форма має такий самий дизайн, як і решта застосунку: використовуються фірмові кольори школи у світло- і темно-зелених відтінках, кнопки «Зберегти» та «Скасувати» виконані в єдиному стилі, узгодженому з кнопками на головному вікні. Завдяки застосуванню централізованих стилів, зміни у дизайні можуть бути внесені у файл ресурсів і автоматично будуть відображені на всіх формах включно з StudentsView та іншими.

Інтерфейс кожного модуля системи насичений різними елементами управління WPF, такими як поля введення TextBox, PasswordBox для паролів, мітки Label/TextBlock для відображення тексту, кнопки, списки, що випадають

ComboBox для вибору, перемикачі CheckBox або RadioButton та інші. Розташування елементів здійснюється за допомогою гнучких панелей компоновки WPF – Grid, StackPanel, DockPanel тощо. У StudentsView використано сіткове розташування з двома рядками: верхній рядок – DataGrid зі списком, нижній – панель з кнопками. У формі редагування студентів застосовано Grid з кількома рядками і двома стовпцями для вирівнювання міток та полів вводу в стовпчики.

Прив’язка даних (Data Binding) – ключовий механізм, що з’єднує логіку та інтерфейс. У XAML-файлах задаються зв’язки між властивостями елементів UI і властивостями моделей або ViewModel. Наприклад, <TextBox Text=«{Binding FirstName}»/> у формі редагування зв’язує текстове поле з властивістю FirstName об’єкта Student у ViewModel. Якщо користувач змінює текст у полі, властивість FirstName у моделі одразу оновлюється (за умови встановлення Mode=TwoWay для двостороннього зв’язку). Навпаки, якщо програмно змінити значення у властивості, інтерфейс автоматично відобразить цю зміну. Таке двостороннє зв’язування активно використовується для відображення даних, що приходять з сервера, та збору введених користувачем даних для відправки на сервер.

Кожне вікно має пов’язані команди або обробники подій, які визначають поведінку при взаємодії користувача. У ViewModel оголошуються команди типу DelegateCommand/RelayCommand, які прив’язуються до кнопок. Наприклад, команда DeleteStudentCommand буде викликана при натисканні кнопки «Видалити студента» і виконає вилучення запису, викликавши метод сервісу StudentService для видалення на сервері. Якщо ж не використовувати команди, можливе використання подій code-behind: у StudentsView.xaml.cs (файл back-коду вікна) можна обробляти кліки на кнопках і викликати відповідні методи. У розробленому застосунку застосовано комбінований підхід: для простих дій використовуються команди, для більш складних сценаріїв або навігації – події в code-behind з перевіркою стану.

Окрім взаємодії «користувач–елемент», інтерактивність UI проявляється у динамічних змінах інтерфейсу. Наприклад, після додавання нового студента у

формі (рис. 3.3) користувач натискає «Зберегти» програма передає дані на сервер і після успішного додавання автоматично оновлює список студентів у головному вікні StudentsView. Це досягається викликом оновлення колекції студентів, тобто отримання нового списку або додавання нового елемента до ObservableCollection, що через data binding миттєво відображає нового студента у DataGrid без необхідності ручного перезавантаження інтерфейсу користувачем.

За аналогією з модулем студентів розроблено інтерфейси інших вкладок. Вкладка *Адміністрування працівників* має схожу структуру: таблиця співробітників з полями (ПІБ, посада, ставка, тощо) та панель дій для додавання/редагування/видалення працівника. Форму редагування працівника побудовано подібно до форми студента, але з іншими полями (інформація про посаду, заробітну плату, графік роботи тощо). Вкладка *Розклад* відображає таблицю занять, а саме сітка днів тижня і занять, де заповнюються назви груп, які виділені кольором присвоєному кожному з викладачів. При натисканні на конкретну комірку, інформацію в ній можна відредагувати, додати або очистити (за умови відповідних прав). *Календар подій* реалізований з використанням стандартного WPF Calendar, доповненого можливістю позначати дати, які мають заходи. Для візуалізації подій на конкретний день може використовуватися список подій під календарем або підсвічування дат у самому календарі. *Таск-менеджер* представлений у вигляді списку завдань, що відображає назви завдань, їх статус та важливість. *Зарплатний лист* відображає фінансові дані у вигляді таблиць та графіків: таблиця виплат по тижнях для всіх працівників. Інтерфейс цього модуля також уніфікований – використано таблиці для списку співробітників з колонками «Місяць», «Відпрацьовано годин», «Нараховано зарплати», «Виплачено» та ін.

Хоча десктопний застосунок не потребує мобільної адаптивності, важливо забезпечити коректне відображення інтерфейсу на різних розмірах екрану та при різних роздільностях. Для цього WPF надає гнучкі макети, що автоматично підлаштовуються: елементи у Grid можуть розтягуватися (HorizontalAlignment=«Stretch») чи заповнювати доступний простір з

використанням *-розмірів рядків і стовпців. У проєкті головне вікно зроблено масштабованим – якщо користувач розгортає його на весь екран або змінює розмір, табличні частини збільшуються, щоб показати більше записів, а форми вводу центровані або розтягуються відповідно до встановлених правил компоновки. Текст та елементи управління залишаються читабельними завдяки використанню відносних розмірів шрифтів і можливості Windows масштабувати інтерфейс.

Отже, клієнтський інтерфейс десктопного застосунку побудований за сучасними принципами UI/UX, який визначає єдиний стиль і структура, чітка навігація через вкладки, використання табличного відображення для зручного перегляду списків, інтуїтивні форми введення даних із підказками та перевіркою, інтерактивність без перезапуску. Усі основні вікна реалізовані аналогічним чином до розглянутих прикладів StudentsView та форм редагування, що забезпечує однаковий підхід до роботи з різними сутностями і полегшує підтримку коду через багаторазове перевикористання одних і тих самих рішень.

3.3 Функціональні можливості системи

Клієнтський застосунок CRM-системи забезпечує широкий спектр функціональних можливостей, необхідних для ефективного управління школою іноземних мов. У цьому підрозділі буде розглянуто реалізацію ключових функцій: авторизація та управління сеансом користувача, операції зі студентами та працівниками (створення, редагування, видалення, імпорт/експорт), планування розкладу, робота із задачами та нагадуваннями, облік заробітної плати, система повідомлень та формування звітності. Окрім того буде пояснено як саме десктопний застосунок взаємодіє з серверною частиною Web API через шар сервісів, що забезпечує збереження та оновлення інформації в базі даних.

Застосунок реалізує патерн «клієнт-сервер», де всі дані зберігаються на сервері, а клієнт звертається до них через HTTP-запити до Web API. Для спрощення цих викликів у клієнті створено окремі сервісні класи, зокрема: ApiService, AuthService, StudentService, AuthGuardService та ін. Клас ApiService

відповідає за низькорівневу роботу з HTTP – встановлює базову URL сервера, надсилає запити (GET, POST, PUT, DELETE) та обробляє відповіді (статуси, помилки). На його основі побудовані більш спеціалізовані сервіси. AuthService містить методи для авторизації та реєстрації: Login(username, password) формує запит до API /auth/login, а Register(userInfo) – до /auth/register. StudentService інкапсулює виклики, пов'язані зі студентами: GetAllStudents(), AddStudent(student), UpdateStudent(student), DeleteStudent(id), ImportStudents(file) тощо, які звертаються до відповідних кінцевих точок API. Аналогічно реалізовано EmployeeService для операцій над працівниками, ScheduleService для розкладу, TaskService для задач. Програмний код для взаємодії front-end елементів для адміністрування студентів у вікні програми з back-end; для підключення до back-end для отримання повного функціоналу програми; для взаємодії front-end елементів у вікні авторизації програми з back-end наведено у додатках Г, Ґ і Д відповідно.

Для контролю доступу на боці клієнта введено клас AuthGuardService. Його завдання – перевіряти, чи має право користувач доступ до певного функціоналу, чи виконано вхід в систему. AuthGuardService, використовується перед відкриттям головного вікна або при спробі виклику захищеного API. Якщо користувач не авторизований, AuthGuardService може блокувати виконання дії і перенаправляти користувача до вікна входу. Цей сервіс грає роль додаткового рівня безпеки і зручності, щоб запобігти некоректному використанню інтерфейсу.

При реалізації функціональності клієнта важливо правильно організувати кожну операцію: UI викликає відповідний метод сервісу, де сервіс надсилає запит на сервер і обробляє результат, після чого UI відображає оновлені дані або повідомлення про результат операції. Нижче буде розглянуто основні функції системи та як вони реалізовані.

Під час запуску програми користувачу показується форма входу, де вводиться логін і пароль. Ці дані передаються в AuthService.Login(), який викликає API авторизації. Якщо сервер підтверджує облікові дані, він повертає

токен сесії та індикатор успішного входу. Клієнт зберігає цей токен у пам’яті, а при потребі в захищеному сховищі чи реєстрі Windows для автологіну, та відкриває головне вікно. У разі невдалого входу – на формі відображається повідомлення про помилку авторизації. Реєстрація нового користувача працює аналогічно: у формі реєстрації вводяться необхідні дані, які через AuthService.Register() надсилаються на API. Якщо реєстрація успішна, зазвичай новому користувачу також видається токен або його перенаправляють до форми входу для авторизації. Важливим аспектом є перевірка сесії, де після входу AuthService зберігає отриманий токен і додає його в заголовки кожного наступного запиту. AuthGuardService контролює строк дії токена – якщо сервер відповідає відмовою через недійсний/протермінований токен, клієнт виконує процедуру автоматичного виходу – закриває головне вікно, повертає до екрану входу і робить запит на повторну авторизацію. У результаті гарантується безперервність та безпека сеансу, коли користувач не мусить повторно входити при кожній дії, але якщо застосунок закрито або токен протерміновано, необхідна повторна авторизація.

Модуль студентів підтримує повний набір CRUD-операцій (Create, Read, Update, Delete) [40]. *Створення інформації про нового студента*: після заповнення форми (рис. 3.3) і натискання «Зберегти» викликається метод StudentService.AddStudent(studentModel), який перетворює дані студента у формат DTO, очікуваний API і надсилає HTTP POST запит на endpoint сервера. У разі успіху сервер поверне створений об’єкт студента або підтвердження; клієнт закриває форму і оновлює локальний список студентів. *Редагування інформації про студента:* при натисканні «Редагувати» програма відкриває форму з уже заповненими полями, після внесення змін і збереження викликається StudentService.UpdateStudent(editedStudent), який відправляє HTTP PUT (або PATCH) запит на сервер, де передає змінені дані. Якщо відповідь успішна, на клієнті одразу оновлюється відображення в списку – в ObservableCollection відповідний об’єкт змінюється, завдяки чому DataGrid показує оновлені дані. *Видалення інформації про студента*: при підтвердженні

видалення викликається StudentService.DeleteStudent(id), що відправляє HTTP DELETE запит на сервер. У разі успіху – запис видаляється і з інтерфейсу. Усі операції супроводжуються повідомленнями про результат: якщо сталася помилка, то користувач побачить відповідне повідомлення

Система передбачає можливість імпорту списку студентів з файлу, які мають розширення CSV, Excel або JSON та експорту наявних даних у файл для зовнішнього використання. В інтерфейсі це реалізовано через діалоги вибору файлу: кнопка «Імпорт» відкриває провідник, де користувач вибирає файл, далі шлях цього файлу передається до StudentService.ImportStudents(filePath). Сервіс може або самостійно зчитувати файл та надсилати дані порціями на сервер, або надіслати файл на сервер. У проєкті для імпорту спрощено зчитування CSV у клієнті та виклик AddStudent для кожного запису. Експорт працює навпаки: StudentService.ExportStudents(format) може запитати у сервера сформувати файл через GET або сам клієнт отримує список студентів і зберігає його у файл локально.

Модуль працівників функціонує аналогічно до модуля студентів. Він також підтримує створення нових записів, редагування даних та видалення працівників. У EmployeeService реалізовані методи AddEmployee, UpdateEmployee, DeleteEmployee, і вони звертаються до відповідних endpoint API. В інтерфейсі адміністратора, встановлені певні обмеження: звичайний викладач не може додавати працівників, ця вкладка доступна лише користувачу з роллю адміністратора. Такі обмеження контролюються або на рівні відображення вкладки, де AuthGuardService може приховувати вкладку «Працівники» для тих, хто не адміністратор, або на рівні спроб виконати дію, якщо викладач спробує відкрити форму – йому буде відмовлено. Функція імпорту/експорту також реалізована і для працівників.

Функціональність розкладу є важливою для координації занять у школі. У системі ведеться список навчальних груп, кожна з яких має свій розклад уроків з днями тижня, годинами занять та класами. У вкладці «Розклад» викладачі можуть переглядати розклад уроків для кожної групи, а редагування доступне

лише адміністратору: додати нові записи, змінити час/дату наявного заняття або скасувати його. Ці дії реалізовані через ScheduleService, який звертається до API розкладу. Візуально розклад представлений у вигляді таблиці, розбитої на класи, дні тижня і години. Для зручності інтегровано календар подій, де у вкладці «Календар» показано загальний календар місяця з позначками на датах, коли є певні події і заходи. Клікнувши на дату, користувач бачить деталі подій цього дня.

У вкладці «Завдання» (таск-менеджер) користувач може створювати персональні завдання та нагадування. Ця функція дуже допомагає користувачам у плануванні та організації. Інтерфейс цього модуля складається зі списку завдань, кожне з яких має статус, назву, опис. Для додавання нового завдання передбачена кнопка «Додати завдання». Дані завдання після підтвердження зберігаються через TaskService.AddTask(task) на сервері. Список завдань завантажується через TaskService.GetTasks() при відкритті вкладки і оновлюється при додаванні/редагуванні. Користувач може відмічати завдання як виконані – тоді в інтерфейсі рядок «ховається» у «виконані» завдання, а сервіс викликає API для оновлення статусу. Нагадування про завдання може дублюватися в календарі: якщо на поточну дату є невиконане завдання, система може відображати сповіщення.

У вкладці «Зарплатний лист» реалізовано функції розрахунку та обліку заробітної плати викладачів та інших працівників. Система накопичує дані про проведені заняття та ставки викладачів, на основі чого можна обчислити суму, яку слід виплатити за певний період. Наприклад, при виборі викладача і місяця система рахує кількість проведених ним уроків за цей місяць та множить на ставку за урок, відповідно отримує суму. Ця інформація відображається в таблиці в окремому полі. Модуль зарплати тісно пов’язаний з даними співробітників і розкладу, тому він використовує і EmployeeService, і ScheduleService. У результаті, керівництво школи може легко отримати інформацію про заробітну плату кожного викладача і контролювати фінансові витрати.

Вкладка «Звітність» надає інструментарій для генерації аналітичних звітів по різних аспектах діяльності школи. Користувач у ролі адміністратора може переглянути тип звіту зі списку: наприклад, «Успішність студентів», «Відвідуваність занять», «Фінансовий звіт за місяць», «Звіт за новими студентами» тощо. Деякі звіти, як фінансові, представлені діаграмами у співвідношеннях доходів і витрат. Ця функціональність полегшує аналіз роботи школи та прийняття рішень на основі даних.

Вкладка «Повідомлення» слугує внутрішньою системою комунікації. Тут відображаються повідомлення, надіслані користувачу, та надається можливість відправляти нові. Наприклад, адміністратор може розсилати оголошення всім студентам певної групи, наприклад, про зміни у розкладі або заходи. Реалізація цієї функції передбачає модель повідомлення Message з полями: від кого, кому, тема, текст, дата. Клієнт отримує список доступних повідомлень через MessageService.GetMessages(userId), де API повертає або всі повідомлення для користувача, або фільтрує за його роллю. Повідомлення відображаються списком, а вибравши конкретне, можна побачити деталі. Для створення нового повідомлення є форма «Нове повідомлення»: поля «Одержувач» (вибір зі списку користувачів або груп), «Тема», «Текст», і кнопка «Надіслати». Натиснувши її, викликається MessageService.SendMessage(msg) -> POST на сервер. Цей модуль підвищує ефективність сповіщення усіх учасників навчального процесу без необхідності сторонніх засобів зв’язку.

Вкладка «Профіль», окрім відображення інформації про користувача, дозволяє змінити пароль. Це реалізовано через форму введення старого і нового пароля та підтвердження дії через AuthService.ChangePassword(oldPwd, newPwd) – PUT запит на API. Також профіль може дозволяти редагувати власні контактні дані (телефон, email) – аналогічно через відповідний сервіс. У «Про систему» жодної складної логіки немає – статична описова інформація про застосунок, з номером версії застосунку та даними про розробників.

Загалом, клієнтська частина CRM-системи забезпечує комплексну підтримку функціональності, необхідної для повноцінного управління

навчальним процесом у школі іноземних мов. Реалізовані механізми дозволяють не лише створювати, редагувати й видаляти записи про студентів, а й забезпечувати їх експорт, імпорт, фільтрацію та перевірку. Підтримка авторизації дозволяє гарантувати безпеку доступу до критичних даних, тоді як централізована робота з API у вигляді сервісів спрощує масштабування та супровід системи. З точки зору архітектурної побудови, використання WPF у поєднанні з Code Behind дозволило досягти прямої взаємодії між компонентами інтерфейсу та логікою програми, що значно пришвидшує етапи розробки в умовах обмежених термінів. Зокрема, структура компонентів у вигляді окремих UserControl забезпечує модульність, а винесення логіки HTTP-викликів у сервіси гарантує повторне використання коду та кращу підтримуваність.

ВИСНОВКИ

У результаті проведеного дослідження було розроблено клієнтську частину CRM-системи для школи іноземних мов, що дозволило не лише підтвердити актуальність і важливість теми, а й окреслити практичні шляхи її реалізації з урахуванням сучасних технологій та вимог користувачів. Сьогодні автоматизація бізнес-процесів у сфері освіти виступає одним із ключових факторів підвищення конкурентоспроможності та якості наданих послуг. У такому контексті CRM-система, як інструмент управління взаємовідносинами з клієнтами, забезпечує не лише ефективне ведення обліку, а й оптимізацію взаємодії між адміністрацією закладу, викладачами та студентами.

Проведений аналіз показав, що наявні CRM-рішення не завжди відповідають потребам невеликих освітніх закладів. Основні проблеми полягають у надлишковій функціональності, складному інтерфейсі, відсутності адаптації до специфіки навчального процесу, а також у переважній орієнтації на веб або мобільні платформи. Для ряду шкіл іноземних мов, де робота значною мірою здійснюється локально, залишається актуальною потреба в автономному десктопному застосунку. Це обумовило доцільність розробки спеціалізованої CRM-системи, яка враховує саме такі умови функціонування.

У рамках бакалаврської роботи було розглянуто особливості побудови клієнтської частини CRM-системи як десктопного застосунку. Визначено функціональні вимоги до інтерфейсу, проведено огляд сучасних інструментів front-end розробки для настільних програм, зокрема Windows Presentation Foundation (WPF), який було обрано як технологічну основу реалізації. Окрема увага приділялася створенню прототипу та проєктуванню інтерфейсу за допомогою Figma, що дозволило забезпечити відповідність інтерфейсу принципам UI/UX-дизайну.

Результатом дослідження стала реалізована клієнтська частина CRM-системи, яка охоплює основні функціональні модулі: реєстрацію та облік клієнтів, відображення навчального розкладу, облік оплати, формування

звітності та інші сервіси, що є необхідними для ефективної діяльності мовної школи. Завдяки впровадженим рішенням вдалося спростити процеси адміністрування, зменшити кількість помилок у роботі з даними, підвищити прозорість комунікацій та якість обслуговування клієнтів. Особливу увагу було приділено принципам зручності у взаємодії користувача з інтерфейсом, а саме: простоті навігації, логічній структурі вікон, інтуїтивно зрозумілим елементам керування, адаптації під типові сценарії роботи персоналу. Це забезпечує швидке засвоєння інтерфейсу новими користувачами та мінімізує потребу в додатковому навчанні персоналу. Виконане дослідження має значення для подальшого розвитку підходів до розробки прикладного програмного забезпечення у сфері освіти. Зокрема, результати роботи можуть бути використані для розробки схожих систем в інших типах навчальних закладів або для створення більш масштабних освітніх платформ із локальною підтримкою клієнтів.

Отже, виконане дослідження підтвердило доцільність розробки спеціалізованих CRM-систем для шкіл іноземних мов, орієнтованих на десктопні застосунки. Практична реалізація клієнтської частини показала, що за умови правильного вибору інструментів і врахування специфіки предметної області можна створити ефективне, зручне та гнучке рішення, здатне покращити освітній сервіс і підвищити якість взаємодії із клієнтами. Перспективи подальшого розвитку включають інтеграцію із серверною частиною системи, розширення функціональності та адаптацію інтерфейсу під різні ролі користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Жосан Г. В., Кириченко Н. В. Управління цифровізацією бізнес-процесів діяльності підприємства. Economic Synergy. 2022. No 4. С. 82–91. URL: https://doi.org/10.53920/ES-2022-4-6

2. Олешко, Т., Попик, Н., & Бронський, О. (2025). ХАРАКТЕРИСТИКА ТА АНАЛІЗ МОДУЛІВ ERP-СИСТЕМ. *Економіка та суспільство*, (72). URL: https://doi.org/10.32782/2524-0072/2025-72-37

3. Жигалкевич, Ж. М., & Залуцький, Р. О. (2023). Improving the quality of business processes of enterprises based on digitization. *Journal of Strategic Economic Research*, (2), 84–93. URL: https://doi.org/10.30857/2786-5398.2023.2.9

4. Янко, А. С., & Шахно, В. О. (2022). Аспект інформаційної безпеки в сучасних CRM-системах в епоху діджиталізації економіки та бізнесу. *Таврійський науковий вісник. Серія: Технічні науки*, (4), С. 28-33. URL: https://doi.org/10.32851/tnv-tech.2022.4.4

5. Pasztaleniec, M., & Skublewska-Paszkowska, M. (2020). Comparative analysis of Windows Presentation Foundation and Windows Forms. *Journal of Computer Sciences Institute*, *14*, 26–30. URL: https://doi.org/10.35784/jcsi.1571

6. Troelsen, A., Japikse, P. (2020). Introducing Windows Presentation Foundation and XAML. In: Pro C# 8 with .NET Core 3. Apress, Berkeley, CA. URL: https://doi.org/10.1007/978-1-4842-5756-2_24

7. Filipova-Petrakieva S., Shopov S. Educational Windows Presentation Foundation and XAML Application for Information Protection based on the Cryptographic Methods – part II. *2021 13th Electrical Engineering Faculty Conference (BulEF)*, Varna, Bulgaria, 8–11 September 2021. URL: https://doi.org/10.1109/bulef53491.2021.9690842

8. Belenesi D.-T., Gabor G., Moisi E. V. Comparative study on WPF and UWP Frameworks used in RSS Application. *2021 13th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, Pitesti, Romania, 1–3 July 2021. URL: https://doi.org/10.1109/ecai52376.2021.9515066

9. Taylor, A.G. (2016). The Universal Windows Platform (UWP). In: Develop Microsoft HoloLens Apps Now. Apress, Berkeley, CA. URL: https://doi.org/10.1007/978-1-4842-2202-7_3

10. Andrade L. M., Domingues P., Frade M. Keeping track of UWP application changes for digital forensic purposes. *2021 Telecoms Conference (ConfTELE)*, Leiria, Portugal, 11–12 February 2021. URL: https://doi.org/10.1109/conftele50222.2021.9435530

11. S. K., M G. Web Application Using HTML, CSS, Java Script and Java. *International Journal of Innovative Research in Engineering*. 2023. P. 124–127. URL: https://doi.org/10.59256/ijire.2023040363

12. Nicoara, R. (2023). HTML and CSS. In: How to be a Web Developer. Apress, Berkeley, CA. URL: https://doi.org/10.1007/978-1-4842-9663-9_3

13. Karka N. R. Best Practices for Building Scalable Single Page Applications (SPAS). *International journal of information technology and management information systems*. 2025. Vol.16, no.1. P.1219-1241. URL: https://doi.org/10.34218/ijitmis_16_01_087

14. Durvas Jayaraman, Kumaresan & Kumar, Avneesh. (2024). Optimizing Single Page Applications (SPA) Through Angular Framework Innovations. 12. 17.

15. Горський, М. П., Огірко, М. О., Солтис, І. В., Дуболазов, О. В., Ушенко, О. Г., Морфлюк-Щур, В. В., & Слоцька, Л. С. (2024). Сучасні фреймворки для створення користувацького інтерфейсу в технологіях електронних видань. *Технологія і техніка друкарства*, (3(85), С. 93–100. URL: https://doi.org/10.20535/2077-7264.3(85).2024.293209

16. Mikita Piastou. Comprehensive Performance and Scalability Assessment of Front-End Frameworks: React, Angular, and Vue.js. *World Journal of Advanced Engineering Technology and Sciences*. 2023. Vol. 9, no. 2. P. 366–376. URL: https://doi.org/10.30574/wjaets.2023.9.2.0153

17. Białecki, G., & Pańczyk, B. (2021). Performance analysis of Svelte and Angular applications. *Journal of Computer Sciences Institute*, *19*, С. 139–143. URL: https://doi.org/10.35784/jcsi.2633

18. Zou D., Darus M. Y. A Comparative Analysis of Cross-Platform Mobile Development Frameworks. *2024 IEEE 6th Symposium on Computers & Informatics (ISCI)*, Kuala Lumpur, Malaysia, 10 August 2024. P. 84–90. URL: https://doi.org/10.1109/isci62787.2024.10667693

19. Hvenfelt, L. (2023). Survey on the state of cross-platform mobile development frameworks (Dissertation). URL: https://urn.kb.se/resolve?urn=urn:nbn:se:lnu:diva-122042

20. Hemasundara Reddy Lanka. Optimizing Java applications with advanced functional programming: a comparative Analysis of Java, Scala, and Kotlin. *Journal of Informatics Education and Research*. 2025. Vol. 5, no. 2. URL: https://doi.org/10.52783/jier.v5i2.2466

21. Goli V. R. Cross-Platform Mobile Development: Comparing React Native and Flutter, and Accessibility in React Native. *International Journal of Innovative Research in Computer and Communication Engineering*. 2023. Vol. 11, no. 03. URL: https://doi.org/10.15680/ijircce.2023.1103002

22. Kurapati, L. (2024). Micro Frontend Architecture in Web Applications. International Journal for Multidisciplinary Research.

23. Necula, S. (2024). Exploring The Model-View-Controller (MVC) Architecture: A Broad Analysis of Market and Technological Applications. URL: https://doi.org/10.20944/preprints202404.1860.v1

24. Smith, A., & Gonzalez, M. (2023). Modernizing Web Applications with MVP and MVC Patterns. ACM Computing Surveys.

25. García, R.F. (2023). MVP: Model–View–Presenter. In: iOS Architecture Patterns. Apress, Berkeley, CA. URL: https://doi.org/10.1007/978-1-4842-9069-9_3

26. Sudip Chakraborty, & P. S. Aithal. (2023). MVVM Demonstration Using C# WPF. International journal of applied engineering and management letters (IJAEML), 7(1), 1–14. URL: https://doi.org/10.5281/zenodo.7538711

27. Code-Behind and XAML in WPF. URL: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/advanced/code-behind-and-xaml-in-wpf

28. ASP.NET Code-behind model overview. URL: https://learn.microsoft.com/en-us/troubleshoot/developer/webapps/aspnet/development/code-behind-model

29. A Review Paper on: UI/UX Design in the Digital Era: Trends, Challenges and Educational Gaps. (2025). *International Research Journal on Advanced Engineering Hub (IRJAEH)*, *3*(05), 2341-2346. URL: https://doi.org/10.47392/IRJAEH.2025.0346

30. 10 Usability Heuristics for User Interface Design. URL: https://www.nngroup.com/articles/ten-usability-heuristics/

31. Abdallah R. Take advantage of Adobe XD software to support the customer's experience in selecting product specifications. *Journal of Design Sciences and Applied Arts*. 2022. Vol. 3, no. 1. P. 295–306. URL: https://doi.org/10.21608/jdsaa.2021.95610.1128

32. Провідні програми з розробки графічного дизайну. URL: https://www.adobe.com/ua/creativecloud/design.html

33. Kowalczyk, E., Glinka, A., & Szymczyk, T. (2022). Comparative analysis of interface sketch design tools in the context of User Experience. *Journal of Computer Sciences Institute*, *22*, 51–58. URL: https://doi.org/10.35784/jcsi.2803

34. Staiano, F. (2022). *Designing and Prototyping Interfaces with Figma: Learn essential UX/UI design principles by creating interactive prototypes for mobile, tablet, and desktop*. Packt Publishing Ltd.

35. Figma. URL: https://www.figma.com

36. Compare Figma vs. Lunacy vs. Penpot. URL: https://slashdot.org/software/comparison/Figma-vs-Lunacy-vs-penpot/

37. Golmohammadi, A., Zhang, M. & Arcuri, A. .NET/C# instrumentation for search-based software testing. *Software Qual J* 31, p. 1439-1465 (2023). URL: https://doi.org/10.1007/s11219-023-09645-1

38. Голубничий Д. Ю., Колесник М. Ю. Використання платформи .NET для розробки застосунків. Поліграфічні, мультимедійні та web-технології : тези доп. IX Міжнар. наук.-техн. конф., 14-18 травня 2024 р. Т. 1. Харків: ТОВ

«Друкарня Мадрид», 2024. С. 81-82. URL: https://openarchive.nure.ua/handle/document/26683

39. Chakraborty, Sudip and Aithal, P. S., CRUD Operation on WordPress Database Using C# And REST API (November 23, 2023). International Journal of Applied Engineering and Management Letters (IJAEML), 7(4), 130-138, 2023; ISSN: 2581-7000. URL: http://dx.doi.org/10.2139/ssrn.4729159

40. Bonteanu A. M., Tudose C. Performance Analysis and Improvement for CRUD Operations in Relational Databases from Java Programs Using JPA, Hibernate, Spring Data JPA. *Applied Sciences*. 2024. Vol. 14, no. 7. P. 2743. URL: https://doi.org/10.3390/app14072743

41. Aravinda A Kumar, Divya TL. Security measures implemented in RESTful API Development. *Open Access Research Journal of Engineering and Technology*. 2024. Vol. 7, no. 1. P. 105–112. URL: https://doi.org/10.53022/oarjet.2024.7.1.0042

42. Shivam, Gupta M. Secure API Gateway with Rate Limiting and JWT Authentication. *International Journal for Research in Applied Science and Engineering Technology*. 2025. Vol. 13, no. 4. P. 3559–3562. URL: https://doi.org/10.22214/ijraset.2025.68953

ДОДАТКИ

ДОДАТОК А

Дизайн сторінок і вікон застосунку в темній темі

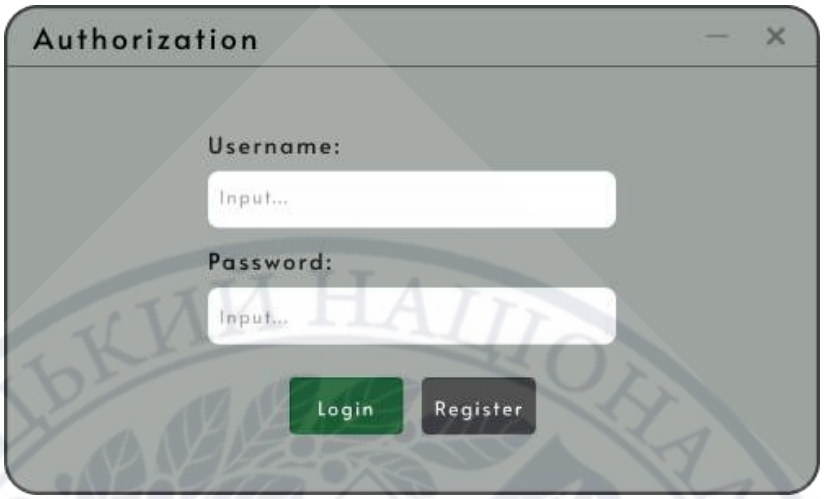

Рисунок А.1

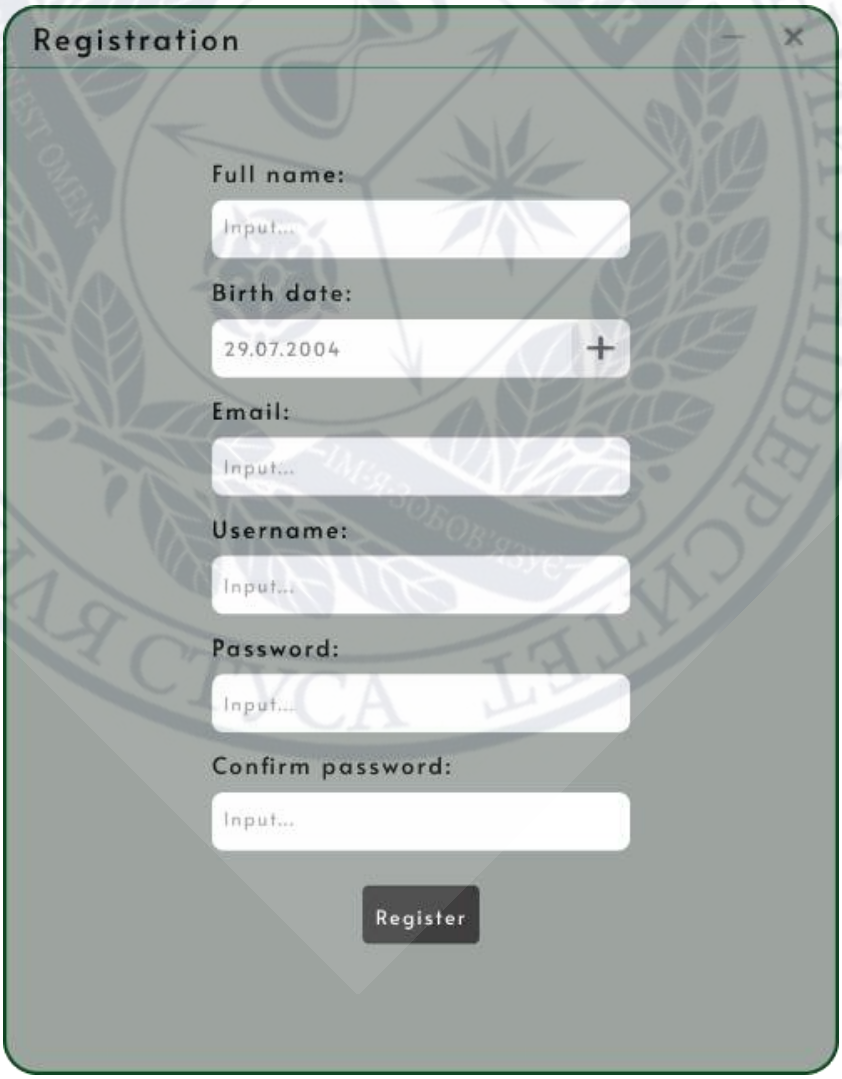

Рисунок А.2

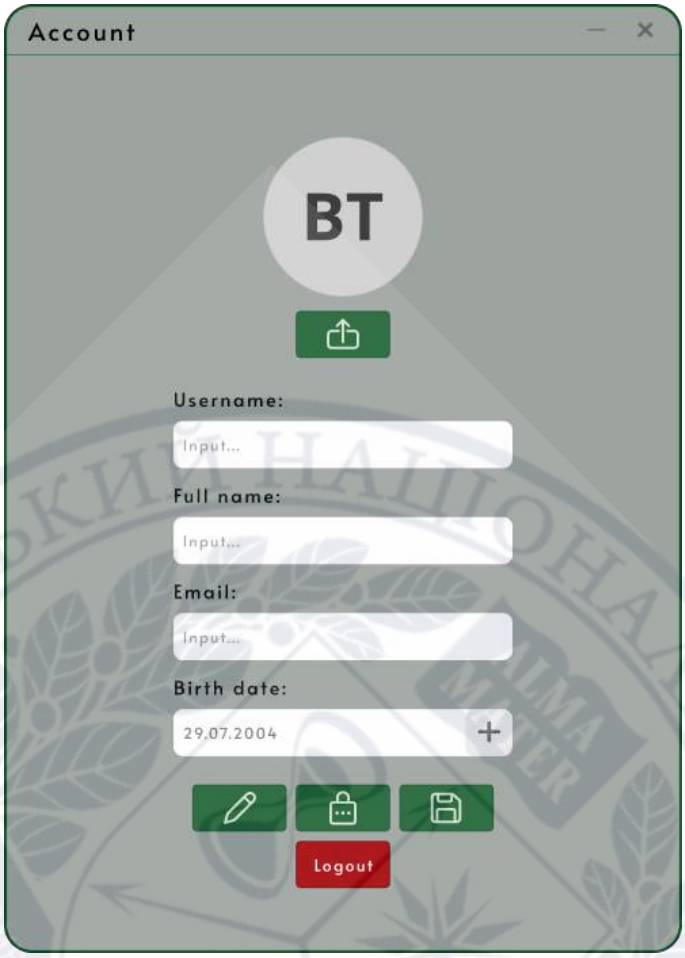

Рисунок А.3

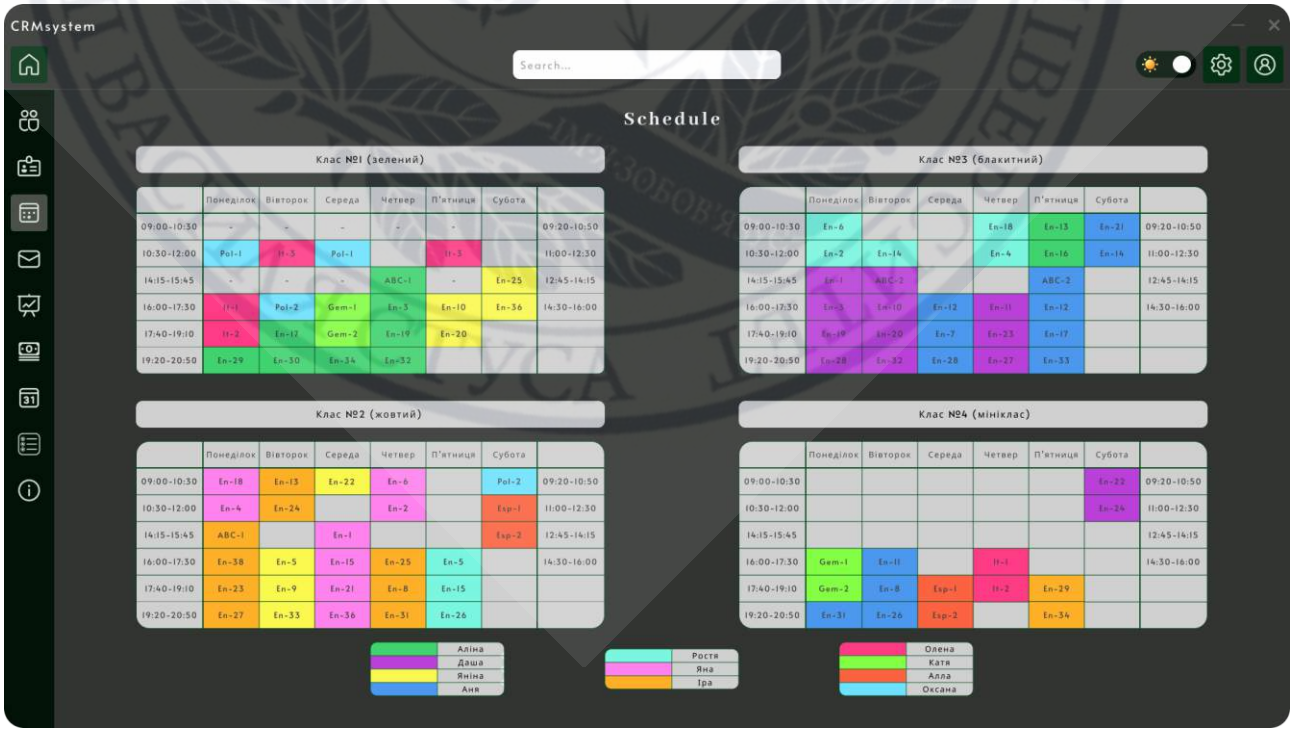

Рисунок А.4

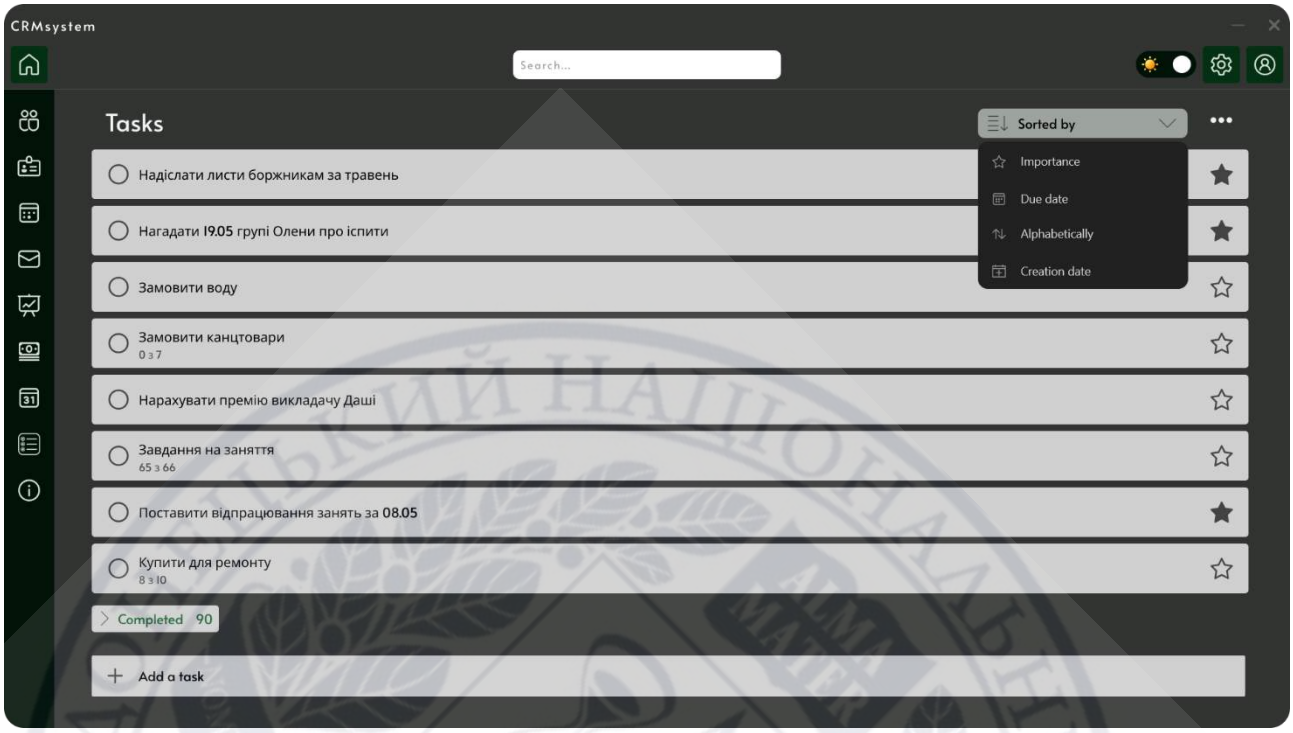

Рисунок А.5

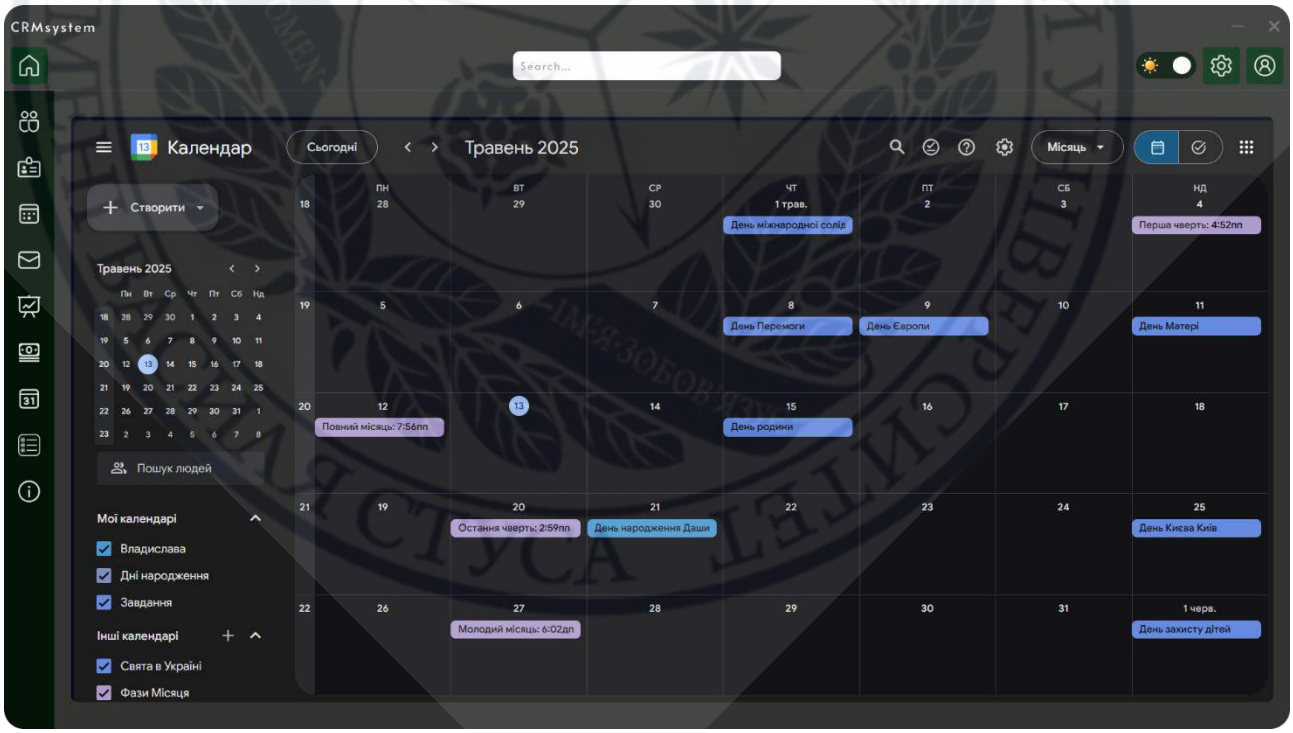

Рисунок А.6

ДОДАТОК Б

Програмний код реалізації інтерфейсу головного вікна та його елементів у застосунку

```
<UserControl x:Class=«TestApp.Views.HomeView»
xmlns=«http://schemas.microsoft.com/winfx/2006/xaml/presentation»
xmlns:x=«http://schemas.microsoft.com/winfx/2006/xaml»
xmlns:mc=«http://schemas.openxmlformats.org/markup-
compatibility/2006»
xmlns:d=«http://schemas.microsoft.com/expression/blend/2008»
             xmlns:local=«clr-namespace:TestApp.Views»
             xmlns:lvc=«clr-
namespace:LiveCharts.Wpf;assembly=LiveCharts.Wpf»
             xmlns:local1=«clr-namespace:TestApp.Custom»
             xmlns:local2=«clr-namespace:TestApp.DTOs»
             mc:Ignorable=«d»
             Height=«auto»
             Width=«auto»
             Background=«Transparent»
             BorderThickness=«0»
             SnapsToDevicePixels=«True»>

    <UserControl.Resources>
        <local1:HeightLimitConverter x:Key=«HeightLimitConverter»/>
    </UserControl.Resources>

    <!--// Home View \\-->
    <Grid
        Margin=«30»
        Background=«Transparent»
        SnapsToDevicePixels=«True»>

        <Grid.RowDefinitions>
            <RowDefinition Height=«Auto»/>
            <RowDefinition Height=«Auto»/>
            <RowDefinition Height=«Auto»/>
            <RowDefinition Height=«*»/>
        </Grid.RowDefinitions>

        <!--// Title Text \\-->
        <StackPanel
            Orientation=«Vertical»
            VerticalAlignment=«Center»
            HorizontalAlignment=«Center»
            Margin=«0,0,0,15»>

            <TextBlock
                Text=«Dashboard»
                FontSize=«26»
                FontWeight=«Bold»
                HorizontalAlignment=«Center»
```

```
                Style=«{StaticResource InfoTextBlockStyle}»/>

        </StackPanel>
<!--// Report Cards \\-->
        <StackPanel
            Grid.Row=«1»
            Orientation=«Horizontal»
            HorizontalAlignment=«Center»
            Margin=«0,0,0,30»>

            <!--// Students Card \\-->
            <Border     Background=«#22783E»     CornerRadius=«10»
Padding=«20» Margin=«10» Width=«200» Height=«100»>
                <StackPanel          HorizontalAlignment=«Center»
VerticalAlignment=«Center»>
                    <TextBlock                    Text=«Students»
Foreground=«{DynamicResource        RegisterLoginForegroundBrush}»
FontSize=«18» HorizontalAlignment=«Center»/>
                    <TextBlock          x:Name=«StudentCountText»
Foreground=«{DynamicResource        RegisterLoginForegroundBrush}»
FontSize=«26» FontWeight=«Bold» HorizontalAlignment=«Center»/>
                </StackPanel>
            </Border>

            <!--// Employees Card \\-->
            <Border     Background=«#064D1E»     CornerRadius=«10»
Padding=«20» Margin=«10» Width=«200» Height=«100»>
                <StackPanel          HorizontalAlignment=«Center»
VerticalAlignment=«Center»>
                    <TextBlock                    Text=«Employees»
Foreground=«{DynamicResource        RegisterLoginForegroundBrush}»
FontSize=«18» HorizontalAlignment=«Center»/>
                    <TextBlock          x:Name=«EmployeeCountText»
Foreground=«{DynamicResource        RegisterLoginForegroundBrush}»
FontSize=«26» FontWeight=«Bold» HorizontalAlignment=«Center»/>
                </StackPanel>
            </Border>

            <!--// Net Profit Card \\-->
            <Border     Background=«#47525E»     CornerRadius=«10»
Padding=«20» Margin=«10» Width=«200» Height=«100»>
                <StackPanel          HorizontalAlignment=«Center»
VerticalAlignment=«Center»>
                    <TextBlock          Text=«Net          Profit»
Foreground=«{DynamicResource        RegisterLoginForegroundBrush}»
FontSize=«18» HorizontalAlignment=«Center»/>

                    <TextBlock              x:Name=«NetProfitText»
Foreground=«{DynamicResource        RegisterLoginForegroundBrush}»
FontSize=«26» FontWeight=«Bold» HorizontalAlignment=«Center»/>
                </StackPanel>
            </Border>
```

```
        </StackPanel>

        <!--// Student Trend Diagram \\-->
        <Border
            Grid.Row=«2»
            Background=«{DynamicResource InputTextForegroundBrush}»
            CornerRadius=«10»
            Padding=«20»
            Margin=«0,0,0,30»>

            <StackPanel>

                <!--// Title \\-->
                <TextBlock
                    Text=«Monthly Student Trend»
                    FontSize=«20»
                    FontWeight=«Bold»
                    Margin=«0,0,0,10»
                    Foreground=«{DynamicResource
RegisterLoginForegroundBrush}»/>

                <!--// Charts \\-->
                <lvc:CartesianChart x:Name=«StudentTrendChart»
                    Series=«{Binding StudentTrendSeries}»
                    Height=«200»
                    LegendLocation=«None»
                    Background=«Transparent»
                    Foreground=«{DynamicResource
RegisterLoginForegroundBrush}»>

                    <lvc:CartesianChart.DataTooltip>

                        <lvc:DefaultTooltip
                            Foreground=«{DynamicResource
RegisterLoginBackgroundBrush}»
                            FontSize=«14»/>

                    </lvc:CartesianChart.DataTooltip>
                    <lvc:CartesianChart.AxisX>

                        <lvc:Axis
                            Labels=«{Binding StudentTrendLabels}»
                            Foreground=«{DynamicResource
RegisterLoginForegroundBrush}»
                            FontSize=«14»/>

                    </lvc:CartesianChart.AxisX>

                    <lvc:CartesianChart.AxisY>

                        <lvc:Axis
                            LabelFormatter=«{Binding
TrendFormatter}»
```

```
                            Foreground=«{DynamicResource
RegisterLoginForegroundBrush}»
                            FontSize=«14»/>

                    </lvc:CartesianChart.AxisY>

                </lvc:CartesianChart>

            </StackPanel>

        </Border>
<!--// Events \\-->
        <Border
            Grid.Row=«3»
            Background=«{DynamicResource InputTextForegroundBrush}»
            CornerRadius=«10»
            Padding=«20»
            Margin=«0,0,0,0»
            SnapsToDevicePixels=«True»>

            <StackPanel>

                <!--// Title \\-->
                <TextBlock
                    Text=«Upcoming Events»
                    FontSize=«20»
                    FontWeight=«Bold»
                    Foreground=«{DynamicResource
RegisterLoginForegroundBrush}»
                    Margin=«0,0,0,20»/>

                <!--// Scrollable List of Events \\-->
                <ScrollViewer
                    MaxHeight=«{Binding
RelativeSource={RelativeSource                    Mode=FindAncestor,
AncestorType=Grid},  Path=ActualHeight,  Converter={StaticResource
HeightLimitConverter}}»
                    Style=«{StaticResource ScrollViewerStyle}»>

                    <ItemsControl>

                        <ItemsControl.ItemsPanel>
                            <ItemsPanelTemplate>
                                <StackPanel/>
                            </ItemsPanelTemplate>
                        </ItemsControl.ItemsPanel>

                        <ItemsControl.ItemTemplate>
                            <DataTemplate>

                                <Border
                                    Background=«{DynamicResource
RegisterLoginBackgroundBrush}»
```

```
                                        CornerRadius="8"
                                        Padding="10"
                                        Margin="0,0,0,10">

                                        <StackPanel>

                                            <TextBlock
                                                Text="{Binding Title}"
                                                FontWeight="SemiBold"
Foreground="{DynamicResource RegisterLoginForegroundBrush}"
                                                FontSize="16"/>

                                            <TextBlock
                                                Text="{Binding Date}"
Foreground="{DynamicResource RegisterLoginForegroundBrush}"
                                                FontSize="14"
                                                Opacity="0.7"/>

                                        </StackPanel>

                                    </Border>

                                </DataTemplate>
                            </ItemsControl.ItemTemplate>
                            <ItemsControl.Items>
                                <local2:EventModel Title="• Event 1"
Date="April 30, 2025"/>
                                <local2:EventModel Title="• Event 2"
Date="May 10, 2025"/>
                                <local2:EventModel Title="• Event 3"
Date="May 10, 2025"/>
                                <local2:EventModel Title="• Event 4"
Date="May 10, 2025"/>
                                <local2:EventModel Title="• Event 5"
Date="May 10, 2025"/>
                            </ItemsControl.Items>

                        </ItemsControl>

                    </ScrollViewer>

                </StackPanel>

            </Border>

        </Grid>

</UserControl>
```

ДОДАТОК В

Програмний код реалізації інтерфейсу вкладки з інформацією про студентів у застосунку

```
<UserControl x:Class=«TestApp.Views.StudentsView»

xmlns=«http://schemas.microsoft.com/winfx/2006/xaml/presentation»

xmlns:x=«http://schemas.microsoft.com/winfx/2006/xaml»
             xmlns:mc=«http://schemas.openxmlformats.org/markup-
compatibility/2006»

xmlns:d=«http://schemas.microsoft.com/expression/blend/2008»
             xmlns:local=«clr-namespace:TestApp.Views»
             mc:Ignorable=«d»
             d:DesignHeight=«450»
             d:DesignWidth=«800»>

    <!--// Students View \\-->
    <Grid
        Background=«Transparent»
        SnapsToDevicePixels=«True»>

        <!--// Main Content \\-->
        <DockPanel
            Background=«Transparent»
            Margin=«20»>

            <Grid>

                <Grid.RowDefinitions>
                    <RowDefinition Height=«Auto»/>
                    <RowDefinition Height=«*»/>
                </Grid.RowDefinitions>

                <!--// Control Operation Panel \\-->
                <StackPanel
                    Grid.Row=«0»
                    Orientation=«Horizontal»
                    DockPanel.Dock=«Top»
                    Margin=«20,0,0,10»>

                    <Button
                        Margin=«0,0,10,0»
                        Style=«{StaticResource AddStyle}»
                        Click=«AddStudent_Click»/>

                    <Button
                        Margin=«0,0,10,0»
                        Style=«{StaticResource EditStyle}»
                        Click=«EditStudent_Click»/>
```

```
                    <Button
                        Margin=«0,0,10,0»
                        Style=«{StaticResource RemoveStyle}»
                        Click=«DeleteStudent_Click»/>

                    <Button
                        Margin=«0,0,10,0»
                        Style=«{StaticResource ImportStyle}»
                        Click=«ImportStudents_Click»/>

                    <Button
                        Margin=«0,0,10,0»
                        Style=«{StaticResource ExportStyle}»
                        Click=«ExportStudents_Click»/>

                </StackPanel>

                <!--// Filter Panel \\-->
                <StackPanel
                    Grid.Row=«0»
                    Orientation=«Horizontal»
                    HorizontalAlignment=«Right»
                    Margin=«0,0,20,0»>

                    <TextBox x:Name=«StudentFilterTextBox»
                        Style=«{StaticResource SearchStyle}»

TextChanged=«StudentFilterTextBox_TextChanged»/>

                </StackPanel>

<!--// Students Table \\-->
                <DataGrid x:Name=«StudentsDataGrid»
                    Grid.Row=«1»
                    ItemsSource=«{Binding Employees}»
                    SelectedItem=«{Binding       SelectedEmployee,
Mode=TwoWay}»
                    Style=«{StaticResource DataGridStyle}»
                    CellStyle=«{StaticResource DataGridCellStyle}»
                    RowStyle=«{StaticResource DataGridRowStyle}»
                    ColumnHeaderStyle=«{StaticResource
DataGridColumnHeaderStyle}»

SelectionChanged=«StudentsDataGrid_SelectionChanged»

MouseDoubleClick=«StudentsDataGrid_MouseDoubleClick»>

                    <DataGrid.Resources>
                        <Style               TargetType=«ScrollBar»
BasedOn=«{StaticResource CustomScrollBarStyle}»/>
                    </DataGrid.Resources>
                    <DataGrid.Columns>
```

```
                        <DataGridTextColumn
                            Header=«#»
                            Binding=«{Binding RowNumber}»
                            SortMemberPath=«RowNumber»
                            Width=«0.7*»
                            IsReadOnly=«True»
                            CanUserResize=«False»
                            CanUserSort=«True»
                            ElementStyle=«{StaticResource
CenteredTextCellStyle}»/>

                        <DataGridTextColumn   Header=«Full   Name»
Binding=«{Binding FullName}» Width=«2*» />
                        <DataGridTextColumn   Header=«Birth   Date»
Binding=«{Binding      BirthDate,      StringFormat='dd/MM/yyyy'}»
Width=«1.5*» />
                        <DataGridTextColumn         Header=«Email»
Binding=«{Binding Email}» Width=«2*» />
                        <DataGridTextColumn         Header=«Phone»
Binding=«{Binding Phone}» Width=«2*» />
                        <DataGridTextColumn   Header=«Sign   Date»
Binding=«{Binding      SignDate,      StringFormat='dd/MM/yyyy'}»
Width=«1.5*» />
                        <DataGridTextColumn      Header=«Contract»
Binding=«{Binding ContractNumber}» Width=«2*» />
                        <DataGridTextColumn Header=«Payment Status»
Binding=«{Binding PaymentStatus}» Width=«2*» />

                    </DataGrid.Columns>

                </DataGrid>

            </Grid>

        </DockPanel>

    </Grid>

</UserControl>
```

ДОДАТОК Г

Програмний код для взаємодії front-end елементів для адміністрування студентів у вікні програми з back-end

```
using System.Globalization;
using System.Windows;
using System.Windows.Controls;
using TestApp.Extensions;
using TestApp.Models;

namespace TestApp.Views
{
    public partial class StudentCreateEditView : UserControl
    {
        public Student Student { get; private set; }

        public StudentCreateEditView()
        {
            InitializeComponent();
Student = new Student();
            StudentCreateEditBirthDateSelector.SelectedDate       =
DateTime.UtcNow;
            StudentCreateEditSignDateSelector.SelectedDate        =
DateTime.UtcNow;
        }

        public  StudentCreateEditView(Student  existingStudent)  :
this()
        {
            Student = existingStudent;

            StudentCreateEditParentFullNameTextBox.Text           =
Student.ParentFullName ?? string.Empty;
            StudentCreateEditFullNameTextBox.Text                 =
Student.FullName;
            StudentCreateEditBirthDateSelector.SelectedDate       =
Student.BirthDate;
            StudentCreateEditGradeTextBox.Text                    =
Student.Grade.HasValue ? Student.Grade.ToString() : string.Empty;
            StudentCreateEditEmailTextBox.Text = Student.Email;
            StudentCreateEditPhoneTextBox.Text = Student.Phone;
            StudentCreateEditLanguageTextBox.Text                 =
Student.LanguageName;
            StudentCreateEditLevelTextBox.Text = Student.Level;
            StudentCreateEditGroupTextBox.Text = Student.GroupName;
            StudentCreateEditLessonDaysTextBox.Text               =
Student.LessonDays;
            StudentCreateEditPairTextBox.Text                     =
Student.PairNumber.ToString();
            StudentCreateEditSignDateSelector.SelectedDate        =
Student.SignDate;
```

```
            StudentCreateEditPaymentAmmountTextBox.Text           =
Student.PaymentAmount.ToString(CultureInfo.InvariantCulture);

            StudentCreateEditPaymentStatusComboBox.SelectedIndex  =
Student.IsPaid ? 0 : 1;
        }

        #region ACTIONS

        private void Save_Click(object sender, RoutedEventArgs e)
        {
            if
(string.IsNullOrWhiteSpace(StudentCreateEditFullNameTextBox.Text)

string.IsNullOrWhiteSpace(StudentCreateEditEmailTextBox.Text)

string.IsNullOrWhiteSpace(StudentCreateEditPhoneTextBox.Text)

string.IsNullOrWhiteSpace(StudentCreateEditPaymentAmmountTextBox.T
ext)

!StudentCreateEditBirthDateSelector.SelectedDate.HasValue

!StudentCreateEditSignDateSelector.SelectedDate.HasValue

StudentCreateEditPaymentStatusComboBox.SelectedItem == null)
            {
                MessageBox.Show(«Please  fill  all  the  fields.»,
«Validation Error», MessageBoxButton.OK);
                return;
            }

            if (StudentCreateEditBirthDateSelector.SelectedDate >=
DateTime.Today)
            {
                MessageBox.Show(«'Birth Date' must be in the past.»,
«Validation Error», MessageBoxButton.OK);
                return;
            }

            if
(!decimal.TryParse(StudentCreateEditPaymentAmmountTextBox.Text,
NumberStyles.Any,     CultureInfo.InvariantCulture,     out     var
paymentAmmount))
            {
                MessageBox.Show(«Invalid     payment     format.»,
«Validation Error», MessageBoxButton.OK);
                return;
            }

            var                    paymentStatus                   =
(StudentCreateEditPaymentStatusComboBox.SelectedItem            as
ComboBoxItem)?.Content.ToString()?.ToLower();
```

```
            if (paymentStatus is not («paid» or «unpaid»))
            {
                MessageBox.Show(«Payment Status must be either
'Paid' or 'Unpaid'.», «Validation Error», MessageBoxButton.OK);
                return;
            }

            string                parentFullName                =
StudentCreateEditParentFullNameTextBox.Text.Trim();
            if (string.IsNullOrWhiteSpace(parentFullName))
            {
                Student.ParentFullName = null;
                Student.IsParentRegister = false;
            }
            else
            {
                Student.ParentFullName = parentFullName;
                Student.IsParentRegister = true;
            }
string gradeText = StudentCreateEditGradeTextBox.Text.Trim();
            if (string.IsNullOrWhiteSpace(gradeText))
                Student.Grade = null;
            else if (int.TryParse(gradeText, out int parsedGrade))
                Student.Grade = parsedGrade;
            else
            {
                MessageBox.Show(«Grade must be a valid number or
left empty.», «Validation Error», MessageBoxButton.OK);
                return;
            }

            Student.FullName                                      =
StudentCreateEditFullNameTextBox.Text.Trim();
            Student.BirthDate                                     =
StudentCreateEditBirthDateSelector.SelectedDate.Value.ToUtcDate();
            Student.Email                                         =
StudentCreateEditEmailTextBox.Text.Trim();
            Student.Phone                                         =
StudentCreateEditPhoneTextBox.Text.Trim();
            Student.LanguageName                                  =
StudentCreateEditLanguageTextBox.Text.Trim();
            Student.Level                                         =
StudentCreateEditLevelTextBox.Text.Trim();
            Student.GroupName                                     =
StudentCreateEditGroupTextBox.Text.Trim();
            Student.LessonDays                                    =
StudentCreateEditLessonDaysTextBox.Text.Trim();
            Student.PairNumber                                    =
int.TryParse(StudentCreateEditPairTextBox.Text.Trim(),   out   int
pair) ? pair : 0;
            Student.SignDate                                      =
StudentCreateEditSignDateSelector.SelectedDate.Value.ToUtcDate();
            Student.PaymentAmount = paymentAmmount;
```

```
            Student.IsPaid = paymentStatus == «paid»;
            Student.SignDate = DateTime.UtcNow;

            Window.GetWindow(this)!.DialogResult = true;
            Window.GetWindow(this)!.Close();
        }

        private void Cancel_Click(object sender, RoutedEventArgs e)
        {
            Window.GetWindow(this)!.DialogResult = false;
            Window.GetWindow(this)!.Close();
        }

        #endregion
    }
}
```

ДОДАТОК Ґ

Програмний код для підключення до back-end, щоб отримати повний функціонал програми

```
using System.Net.Http;
using System.Net;
using System.Text.Json;
using System.Text;

namespace TestApp.Services
{
    public static class ApiService
    {
        private static readonly HttpClientHandler handler = new
HttpClientHandler
        {
            UseCookies = true,
            CookieContainer = new CookieContainer(),
            AllowAutoRedirect = true
        };

        private static readonly HttpClient client = new
HttpClient(handler)
        {
            BaseAddress = new Uri(«https://localhost:7200/»)
        };

        private static async Task<HttpResponseMessage>
SendRequestAsync(Func<Task<HttpResponseMessage>> request)
        {
            var response = await request();
if (response.StatusCode == HttpStatusCode.Unauthorized
response.StatusCode == HttpStatusCode.Forbidden)
            {
                AuthGuardService.SetAuthenticated(false);
                throw new UnauthorizedAccessException(«Session
expired. Redirecting to login...»);
            }

            if (!response.IsSuccessStatusCode)
            {
                var error = await
response.Content.ReadAsStringAsync();
                throw new Exception($»API Error:
{response.StatusCode}, Message: {error}»);
            }

            return response;
        }

        public static async Task<T> GetAsync<T>(string url)
```

```
        {
            var response = await SendRequestAsync(() =>
client.GetAsync(url));
            var json = await response.Content.ReadAsStringAsync();

            return JsonSerializer.Deserialize<T>(json, new
JsonSerializerOptions { PropertyNameCaseInsensitive = true });
        }

        public static async Task<HttpResponseMessage>
PostAsync<T>(string url, T data)
        {
            var json = JsonSerializer.Serialize(data);
            var content = new StringContent(json, Encoding.UTF8,
«application/json»);

            return await SendRequestAsync(() =>
client.PostAsync(url, content));
        }

        public static async Task<HttpResponseMessage>
PostFormAsync(string url, MultipartFormDataContent content)
        {
            return await SendRequestAsync(() =>
client.PostAsync(url, content));
        }

        public static async Task<HttpResponseMessage>
PutAsync<T>(string url, T data)
        {
            var json = JsonSerializer.Serialize(data);
            var content = new StringContent(json, Encoding.UTF8,
«application/json»);

            return await SendRequestAsync(() =>
client.PutAsync(url, content));
        }

        public static async Task<HttpResponseMessage>
PatchAsync<T>(string url, T data)
        {
            var json = JsonSerializer.Serialize(data);
            var content = new StringContent(json, Encoding.UTF8,
«application/json»);
            var request = new HttpRequestMessage(new
HttpMethod(«PATCH»), url)
            {
                Content = content
            };

            return await SendRequestAsync(() =>
client.SendAsync(request));
        }
```

```
        public static async Task<HttpResponseMessage>
DeleteAsync(string url)
        {
            return await SendRequestAsync(() =>
client.DeleteAsync(url));
        }
    }
}
using System.Net.Http;
using System.Text.Json;
using TestApp.Models;

namespace TestApp.Services
{
    public static class StudentService
    {
        public static async Task<IEnumerable<Student>>
GetAllStudentsAsync()
        {
            return await
ApiService.GetAsync<IEnumerable<Student>>(«api/student-
registration»);
        }
        public static async Task<Student> GetStudentByIdAsync(Guid
id)
        {
            return await
ApiService.GetAsync<Student>($»api/student-registration/{id}»);
        }
        public static async Task<bool> CreateStudentAsync(Student
student)
        {
            FixStudentBeforeSend(student);

            var response = await ApiService.PostAsync(«api/student-
registration», student);
            return response.IsSuccessStatusCode;
        }
        public static async Task<bool> UpdateStudentAsync(Guid id,
Student student)
        {
            FixStudentBeforeSend(student);

            var response = await ApiService.PutAsync($»api/student-
registration/{id}», student);
            return response.IsSuccessStatusCode;
        }
        public static async Task<bool> DeleteStudentAsync(Guid id)
        {
            var response = await
ApiService.DeleteAsync($»api/student-registration/{id}»);
            return response.IsSuccessStatusCode;
```

```
        }

        public static async Task<IEnumerable<Student>>
ImportStudentsAsync(MultipartFormDataContent formData)
        {
            var response = await
ApiService.PostFormAsync(«api/student-registration/import-file»,
formData);
            var json = await response.Content.Rea
dAsStringAsync();

            return
JsonSerializer.Deserialize<IEnumerable<Student>>(json, new
JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
            }) ?? [];
        }
        public static async Task<byte[]> ExportStudentsAsync(string
fileName)
        {
            var students = await GetAllStudentsAsync();

            foreach (var student in students)
                FixStudentBeforeSend(student);

            var json = JsonSerializer.Serialize(students, new
JsonSerializerOptions
            {
                PropertyNamingPolicy = JsonNamingPolicy.CamelCase,
                WriteIndented = true
            });
            var content = new StringContent(json);
            content.Headers.ContentType = new
System.Net.Http.Headers.MediaTypeHeaderValue(«application/json»);

            var response = await
ApiService.PostAsync($»api/student-registration/export-
file?fileName={fileName}», content);
            return await response.Content.ReadAsByteArrayAsync();
        }
        private static void FixStudentBeforeSend(Student student)
        {
            if (student.Grade is < 0)
                student.Grade = null;

            student.IsParentRegister =
!string.IsNullOrWhiteSpace(student.ParentFullName);
            student.ParentFullName = student.IsParentRegister ?
student.ParentFullName?.Trim() : null;
        }
    }
}
```

ДОДАТОК Д

Програмний для взаємодії front-end елементів у вікні авторизації програми з back-end

```
namespace TestApp.Services
{
    public static class AuthGuardService
    {
        private static bool _isAuthenticated = false;

        public static bool IsAuthenticated => _isAuthenticated;

        public static void SetAuthenticated(bool value)
        {
            _isAuthenticated = value;
        }

        public static async Task<bool>
ValidateAuthenticationAsync()
        {
            try
            {
                var user = await UserService.GetCurrentUserAsync();
                _isAuthenticated = user != null;
            }
            catch
            {
                _isAuthenticated = false;
            }

            return _isAuthenticated;
        }
    }
}

namespace TestApp.Services
{
    public static class AuthService
    {
        public static async Task<bool> LoginAsync(string username,
string password)
        {
            var user = new
            {
                Username = username,
                Password = password
            };

            var response = await
ApiService.PostAsync(«api/auth/sign-in», user);
```

```
            if (response.IsSuccessStatusCode)
            {
                AuthGuardService.SetAuthenticated(true);
                return true;
            }

            return false;
        }

        public static async Task<bool> RegisterAsync(string fullName, DateTime birthDate, string email, string username, string password, string confirmPassword)
        {
            var utcBirthDate = DateTime.SpecifyKind(birthDate.Date, DateTimeKind.Utc);

            var user = new
            {
                FullName = fullName,
                BirthDate = utcBirthDate,
                Email = email,
                Username = username,
                Password = password,
                ConfirmPassword = confirmPassword
            };
            var response = await ApiService.PostAsync(«api/auth/sign-up», user);
            return response.IsSuccessStatusCode;
        }
        public static async Task<bool> LogoutAsync()
        {
            var response = await ApiService.PostAsync(«api/auth/sign-out», new { });
            return response.IsSuccessStatusCode;
        }
    }
}
using System.Windows;
using TestApp.Services;

namespace TestApp.Views
{
    public partial class LoginWindow : Window
    {
        public LoginWindow()
        {
            InitializeComponent();
        }

        #region BASE METHODS

        private void MinimizeWindow_Click(object sender, RoutedEventArgs e) => WindowState = WindowState.Minimized;
```

```
private void CloseWindow_Click(object sender, RoutedEventArgs e) =>
Close();

        #endregion

        #region ACTIONS

        private   async   void   LoginButton_Click(object   sender,
RoutedEventArgs e)
        {
            try
            {
                if
(string.IsNullOrWhiteSpace(LoginUsernameTextBox.Text)
string.IsNullOrWhiteSpace(LoginPasswordTextBox.Password))
                {
                    MessageBox.Show(«Please  enter  'Username'  and
'Password'.», «Incorrect Data», MessageBoxButton.OK);
                    return;
                }
                var          success          =          await
AuthService.LoginAsync(LoginUsernameTextBox.Text,
LoginPasswordTextBox.Password);

                if (success)
                {
                    var main = new MainWindow();
                    Application.Current.MainWindow = main;

                    main.Show();
                    Close();
                }
                else MessageBox.Show(«Login failed.»);
            }
            catch (Exception)
            {
                MessageBox.Show(«'Username'   or   'Password'   is
incorrect.», «Incorrect Data», MessageBoxButton.OK);
            }
        }
        private   void   RegistrationButton_Click(object   sender,
RoutedEventArgs e)
        {
            var register = new RegisterWindow();

            register.Show();
            Close();
        }

        #endregion
    }
}
```