ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка Web-сервісу для обліку абітурієнтів кафедри мовою C#»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Андрій РОМАНСЬКИЙ
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

Андрій РОМАНСЬКИЙ

Керівник: Віктор ГРЕБЕНЮК
доктор філософії (PhD)

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Романському Андрій Васильовичу

1. Тема кваліфікаційної роботи: «Розробка Web-сервісу для обліку абітурієнтів кафедри мовою C#»
керівник кваліфікаційної роботи доктор філософії (Phd), доцент кафедри ІПЗ Віктор ГРЕБЕНЮК,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: Матеріали переддипломної практики, процеси управління та адміністрування даних, методи роботи WEB-ресурсів, науково-технічна література

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

1. Огляд існуючих веб-сервісів для обліку абітурієнтів та їх аналіз
2. Проектування та розробка
3. Реалізація функціональності
4. Огляд готового продукт

5. Висновки

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Алгоритм роботи застосунку.
6. Діаграма класів.
7. Екранні форми.
8. Апробація результатів дослідження

6. Дата видачі завдання«28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Дослідження програмних засобів	14.03-28.03.2024	
4	Моделювання об’єкту проектування	29.03-10.04.2024	
5	Розробка функціоналу додатку	11.04-15.04.2024	
6	Вступ, висновки, реферат	16.04-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Андрій РОМАНСЬКИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Віктор ГРЕБЕНЮК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 60 стор., 2 табл., 6 рис., 19 джерел.

Мета роботи – спрощення процесу ведення обліку абітурієнтів, за рахунок використання програмного забезпечення з використанням мови C#.

Об’єкт дослідження – процес управління та адміністрування даних абітурієнтів.

Предмет дослідження – програмне забезпечення для віддаленого ведення обліку абітурієнтів.

Короткий зміст роботи: У даній роботі проведено аналіз методів і алгоритмів для обробки інформації у контексті задачі створення веб-застосунку для прийому абітурієнтів. Проаналізовано існуючі рішення для управління заявками та вимоги до програмного забезпечення для системи управління заявками абітурієнтів.

Розроблено алгоритм роботи застосунку та програмно реалізовані ключові функціональні можливості, зокрема: реєстрація абітурієнтів, завантаження резюме, перегляд і редагування даних абітурієнтів, формування таблиць з розподілом по групах, подача заявок на підтримку через форму зворотного зв’язку. Проведено функціональне та модульне тестування додатку. У роботі використано ASP.NET для розробки веб-застосунку, C# для програмної логіки, SQL для роботи з базами даних, HTML, CSS, та JavaScript для створення користувацького інтерфейсу.

Сферою використання застосунку є автоматизація процесу прийому абітурієнтів та управління заявками в освітніх закладах.

Галузь використання – облік абітурієнтів кафедри.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AJAX – Asynchronous JavaScript and XML

API – Application Programming Interface

MVC – model-view-controller architecture

XML – Extensible Markup Language

ПЗ – програмне забезпечення

ВСТУП

Сучасне суспільство виявляє зростаючий інтерес до використання онлайн-сервісів для комунікації, розваг та співпраці. З розширенням популярності та сервісів виникає необхідність у створенні нових та інноваційних інструментів задоволення різноманітних потреб користувачів.

Однією з важливих сфер, яка потребує нових рішень, є організація прийому абітурієнтів. Традиційні методи обробки заяв часто потребують фізичної присутності абітурієнтів, складної організації процесів та ведення обліку. Це може бути незручним та витратним за часом процесом, особливо якщо абітурієнти знаходяться далеко або мають обмежені можливості для особистих зустрічей.

У зв'язку з цим, розробка онлайн-сервісу для прийому абітурієнтів стає нагальною потребою. Такий сервіс здатен значно полегшити процес подання заявок, надати зручний інтерфейс для користувачів та автоматизувати ряд операцій, пов'язаних з обробкою заявок та веденням обліку.

Мета роботи – спрощення процесу ведення обліку абітурієнтів, за рахунок використання програмного забезпечення з використанням мови C#.

Об’єкт дослідження – процес управління та адміністрування даних абітурієнтів.

Предмет дослідження – програмне забезпечення для віддаленого ведення обліку абітурієнтів.

Методи дослідження – методи розробки та проєктування програмного забезпечення, методи обробки вхідних даних.

Опираючись на поставлену мету, були визначені наступні задачі:

- аналіз галузі управління абітурієнтами;
- аналіз готових рішень для галузі управління абітурієнтами;
- розробка та формування вимог до програмного забезпечення для системи обліку абітурієнтів;
- аналіз підходів до розробки веб-сайтів з використанням ASP.NET, C#, інших компонентів для реалізації веб-додатку.
- проектуватування та розробка веб-додаток для системи обліку абітурієнтів;
- розробка засобів для захисту даних користувача

1. ОГЛЯД ІСНУЮЧИХ ВЕБ-СЕРВІСІВ ДЛЯ ОБЛІКУ АБІТУРІЄНТІВ КАФЕДРИ

1.1 Поняття веб-сервісу для обліку абітурієнтів

Веб-сервіс для обліку абітурієнтів – це інтегрована онлайн-система, яка забезпечує автоматизацію процесу прийому заявок, обробки та управління інформацією про абітурієнтів. Такий веб-сервіс надає освітнім установам можливість ефективно організувати прийом абітурієнтів, зменшити адміністративне навантаження та забезпечити високу точність і безпеку даних.

Переваги веб-сервісу для обліку абітурієнтів:

- Ефективність: Зменшення часу та ресурсів, необхідних для обробки заявок.
- Зручність: Надання користувачам інтуїтивно зрозумілого інтерфейсу для подачі заявок та отримання інформації про їхній статус.
- Точність: Зменшення кількості помилок, пов'язаних з ручним введенням даних.
- Безпека: Надійний захист персональних даних абітурієнтів.
- Аналітика: Можливість отримання детальних звітів та аналітики для покращення процесу прийому абітурієнтів.

1.2 Аналіз існуючих веб-сервісів для обліку абітурієнтів

1.2.1 ApplyBoard

ApplyBoard – це канадська освітня технологічна компанія, заснована у 2015 році братами Мартіном, Масі та Метті Басірі. Платформа допомагає іноземним студентам подавати заявки до навчальних закладів у Канаді, США, Великій Британії та Австралії. ApplyBoard використовує платформу з підтримкою штучного інтелекту, яка містить понад 1500 навчальних закладів, дозволяючи

студентам та агентам подавати заявки напряму. Також ApplyBoard запустила ApplyProof – платформу для перевірки документів, таких як листи прийому та тести на знання англійської мови, що забезпечує достовірність заявок студентів.

Переваги:

- широкий вибір навчальних закладів;
- система дозволяє студентам подавати заявки онлайн, що значно спрощує процес.
- використання штучного інтелекту для покращення рекомендацій та процесу подачі заявок.

Недоліки:

- можуть бути випадки, коли рекомендації не завжди відповідають реальним потребам студентів.
- проблеми з платформою можуть вплинути на процес подачі заявок.

Приклад головного інтерфейсу зображений на рисунках 1.1, 1.2.

ApplyBoard виявився найбільш функціональним з цих трьох сайтів, оскільки має більше можливостей, таких як зберігання обраних програм та закладів, оцінки користувачів програм і закладів, а також наявність інформації про візи і стипендії. StudyLink, хоча має деякі функції, такі як пошук програм навчання і зв'язок з закладами, але відсутні функції, які є у ApplyBoard, такі як зберігання обраних програм і оцінки користувачів.

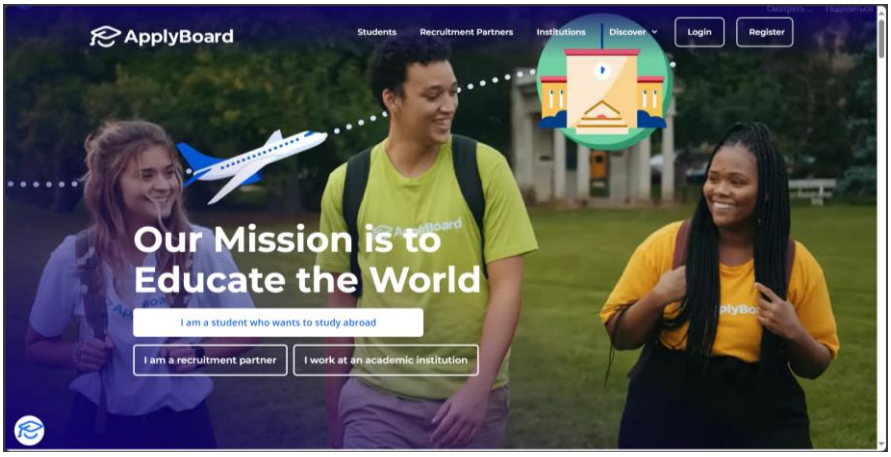


Рис. 1.1 Головна сторінка ApplyBoard

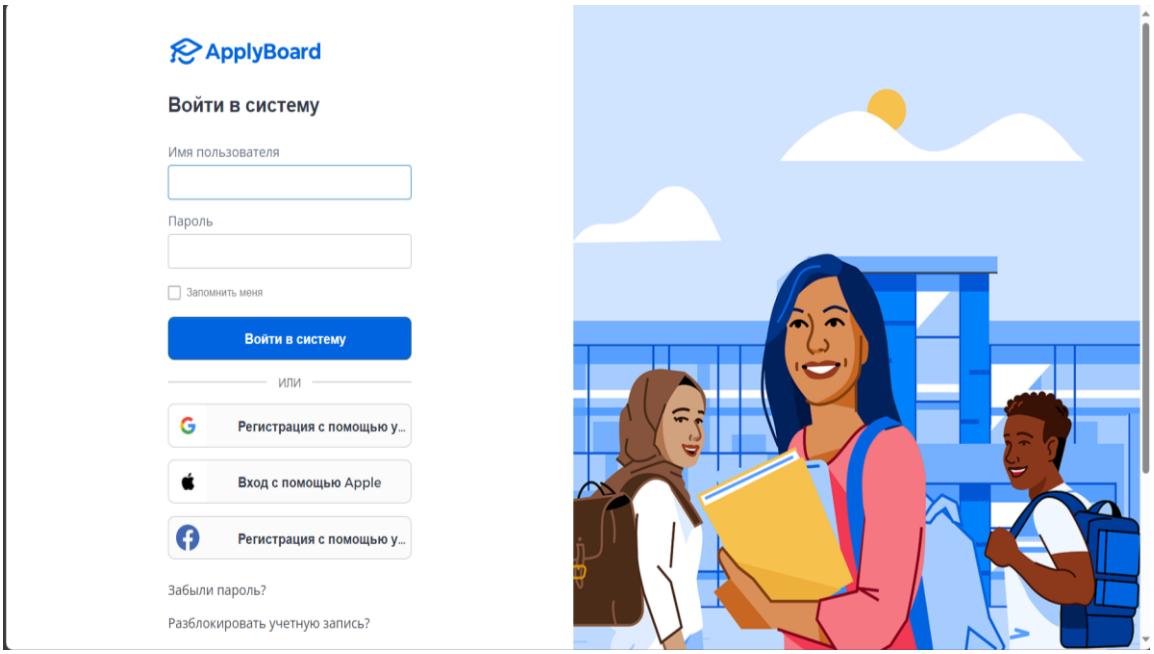


Рис. 1.2. Сторінка Вхожу користувача

1.2.2 Сервіс StudyLink

StudyLink – це онлайн-платформа, що допомагає студентам знаходити та подавати заявки на навчання в міжнародних університетах і коледжах. Платформа пропонує широкий вибір курсів і програм від закордонних навчальних закладів, надаючи інформацію про вимоги до вступу, вартість навчання, стипендії та інші можливості для студентів.

Переваги:

- StudyLink пропонує доступ до тисяч навчальних програм у різних країнах світу.
- Платформа дозволяє студентам легко знаходити і подавати заявки онлайн.
- StudyLink надає детальну інформацію про умови вступу, фінансові вимоги та інші важливі аспекти навчання за кордоном.

Недоліки:

- Платформа може не надавати повноцінної підтримки під час вступу, що іноді потребує додаткових консультацій.

- Деякі послуги можуть бути платними, що може збільшити загальну вартість процесу вступу.

Приклад графічного інтерфейсу StudyLink на рисунках 1.3.

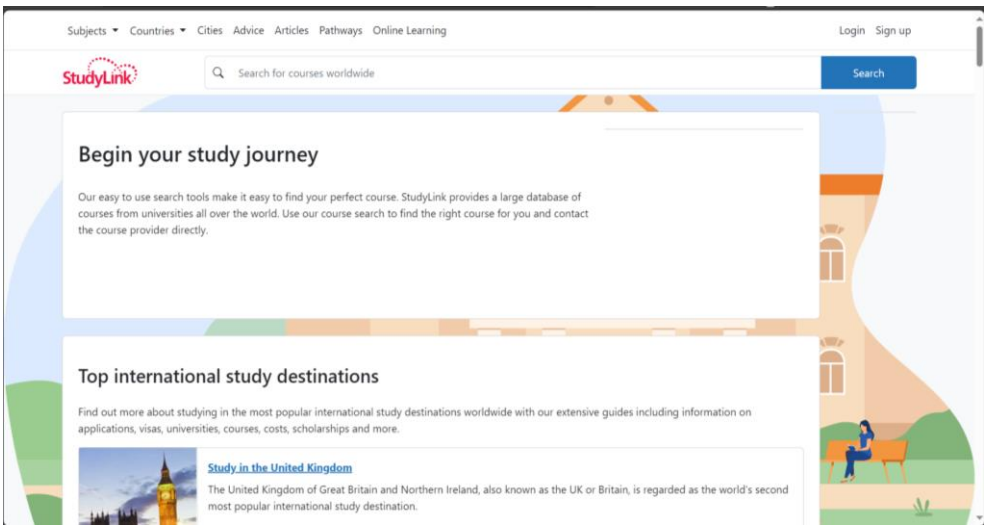


Рис. 1.3. Головна сторінка StudyLink

1.2.3 Сервіс EducationUSA

EducationUSA - це глобальна мережа адвізорів, яка надає інформацію та підтримку студентам, які цікавляться навчанням у США. Створена за ініціативою Державного департаменту США, мережа EducationUSA допомагає студентам у різних країнах зрозуміти процес вступу до американських університетів, вибрати найбільш підходящі навчальні заклади та отримати інформацію про стипендії та фінансову допомогу. Розрахована на абітурієнтів не тільки з США але і з інших країн.

Переваги:

- Сервіс надає об'єктивну та достовірну інформацію про навчання у США, що допомагає студентам зробити обдуманий вибір щодо своєї освіти. Надає данні про більшість вузів США
- EducationUSA надає широкий спектр послуг, включаючи консультації та інформаційну підтримку, що допомагає студентам успішно подолати процес вступу до американських університетів

- Сервіс надає доступ до безкоштовних та платних ресурсів для самостійного вивчення, що дозволяє студентам підготуватися до вступних випробувань та ознайомитися з процесом вступу.

Недоліки:

- У деяких країнах можуть бути обмежені можливості отримання доступу до послуг EducationUSA через політичні або соціальні обставини.
- Хоча сервіс надає загальну інформацію, він може не завжди враховувати індивідуальні потреби та ситуації кожного студента.
- Рівень доступності та якості послуг EducationUSA може варіюватися в залежності від регіону та країни, де надаються послуги.

EducationUSA є одним з найкращих сервісів на даний час він надає доступ для подання заявок для абітурієнтів з більшості країн світу. Це є значним плюсом адже усі інші веб-застосунки розраховані на абітурієнтів з певного кола країн, що обмежує їх доступ для більшої частини світу.

Приклад графічного інтерфейсу на рисунку 1.4.

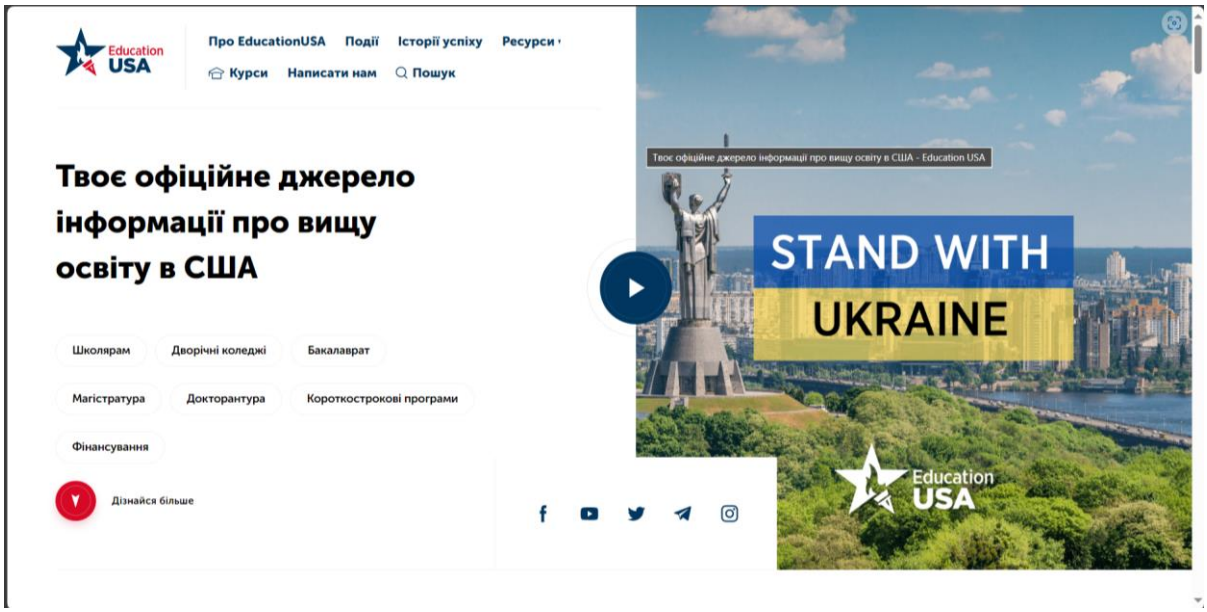


Рис. 1.4. Головна сторінка EducationUSA

Результати аналізу цих сайтів наведена у таблиці 1.1.

Таблиця 1.1

Аналіз сайтів

Показник	ApplyBoard	StudyLink	EducationUSA
Платформи	Web	Mobile,Web	Web
Реєстрація користувача	+	+	-
Зберігання обраних програм	+	-	-
Можливість зв'язку з закладом	+	+	+
Пошук програм навчання	+	+	+
Пошук навчальних закладів	+	+	+
Наявність інформації про курси	+	-	+
Наявність інформації про стипендії	+	-	+

2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ВЕБ-СЕРВІСУ

2.1 Моделювання вимог до програмного забезпечення

Діаграма варіантів використання демонструє, які можливості має система з точки зору її користувачів. Вона показує, які дії можуть виконувати користувачі системи (актори) та які функції доступні в системі (варіанти використання). Загальний вид діаграми варіантів використання сервісу для обліку абітурієнтів кафедри наведено на рисунку 2.1

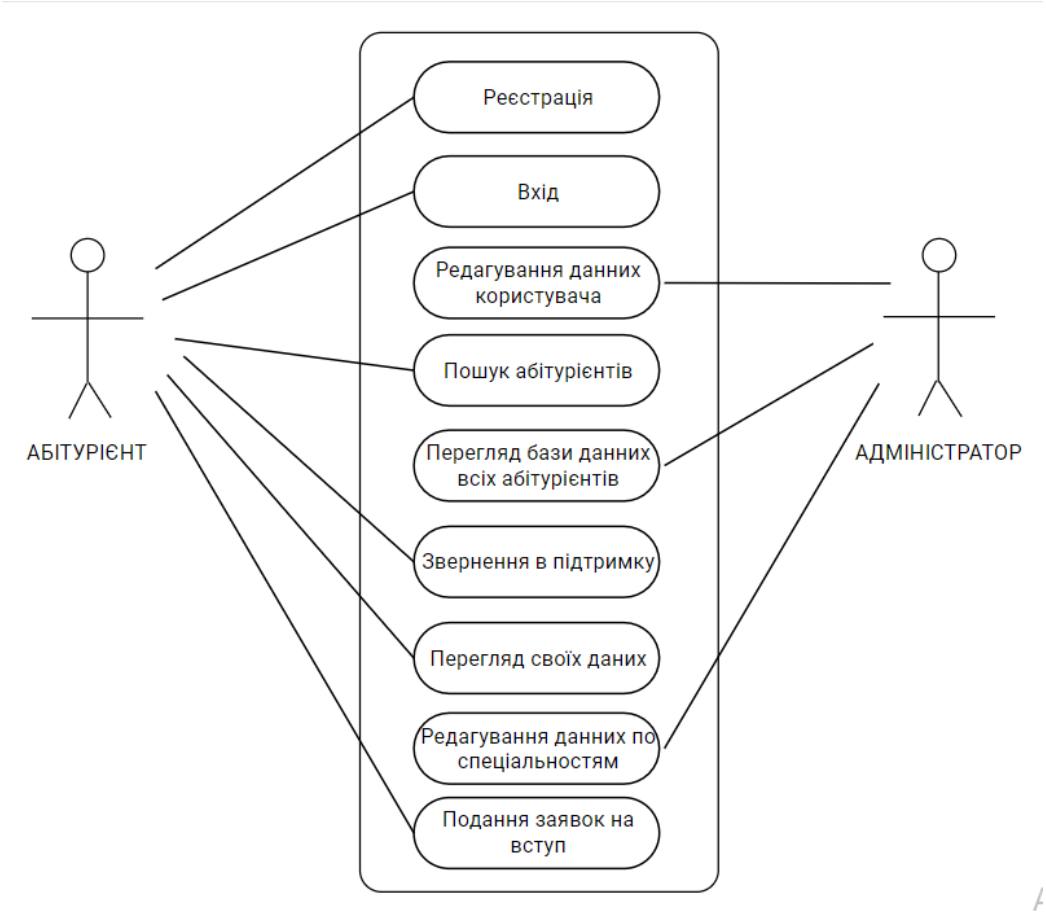


Рис. 2.1 Діаграма варіантів використання

Функціонал сервісу обліку абітурієнтів кафедри надає багато можливостей для усіх користувачів. Не обов’язково бути зареєстрованим на сервісі для доступу

до його основних можливостей. Це дає змогу новим користувачам без створення аккаунту отримати доступ до всіх функцій нашого сервісу

Користувач з правами адміністратора має доступ до редагування даних усіх користувачів та їх перегляд.

2.2 Опис архітектури MVC

Для розроби сервісу по обліку абітурієнтів кафедри була обрана архітектура MVC. Архітектура MVC (Model-View-Controller) – це підхід до побудови програмного забезпечення, який розділяє програму на три основні компоненти: Модель (Model), Представлення (View) та Контролер (Controller). Це дозволяє організувати код у більш структурований і зрозумілий спосіб, що спрощує розробку, тестування та підтримку програмного забезпечення.

Графічний приклад архітектури MCV зображено на рисунку 2.2.

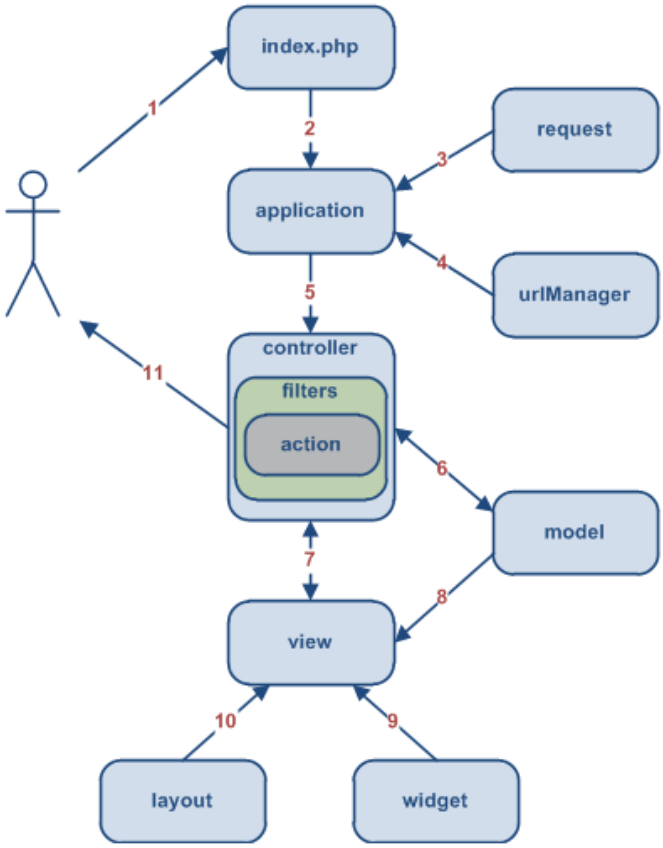


Рис. 2.2 Графічний вигляд архітектури MVC

2.2.1 Модель (Model) в архітектурі MVC

Модель (Model) в архітектурі MVC є центральним компонентом, який відповідає за управління даними та логікою бізнесу. Вона забезпечує доступ до даних, маніпуляцію цими даними і підтримує правила бізнес-логіки, необхідні для додатка. Модель працює незалежно від інших компонентів (Представлення і Контролера), що дозволяє забезпечити гнучкість та масштабованість системи.

Основні функції моделі:

Управління даними: Модель відповідає за доступ до даних, які можуть зберігатися в базах даних, файлах або інших джерелах. Вона включає методи для отримання, збереження, оновлення і видалення даних. Це дозволяє зосередити всі операції з даними в одному місці.

Бізнес-логіка: Модель містить бізнес-логіку, яка визначає, як дані обробляються і які правила застосовуються. Це може включати валідацію даних, обчислення, перетворення даних і інші операції, які є необхідними для підтримки бізнес-процесів.

Сповіщення: Модель може використовувати механізми сповіщення, такі як спостерігач (Observer pattern), для інформування інших компонентів про зміни в даних. Це забезпечує автоматичне оновлення представлень (Views) при зміні даних у моделі, підтримуючи актуальність відображуваної інформації.

Роль моделі в MVC:

Модель виконує роль "чистого" компонента в MVC, оскільки вона не знає нічого про інтерфейс користувача чи контролери. Вона зосереджується лише на управлінні даними і бізнес-логіці. Це розділення відповідальностей дозволяє легко змінювати і підтримувати кожен компонент системи без впливу на інші частини.

Переваги використання моделі:

Гнучкість: Оскільки модель є незалежною від представлення і контролера, зміни в бізнес-логіці або структурі даних не впливають на користувацький інтерфейс. Повторне використання:

Модель можна використовувати в різних контекстах і додатках, забезпечуючи єдину точку доступу до даних і логіки.

Тестування: Окреме тестування моделі полегшується, оскільки вона не залежить від інших компонентів системи. Це дозволяє проводити юніт-тести для перевірки правильності бізнес-логіки і обробки даних.

Архітектура MVC забезпечує модульний і структурований підхід до розробки програмного забезпечення, що робить її однією з найпопулярніших і найефективніших методологій для створення складних і масштабованих додатків.

2.2.2 Представлення (View) в архітектурі MVC

Представлення (View) в архітектурі MVC відповідає за відображення даних користувачеві. Це компонент, який користувач безпосередньо бачить і з яким взаємодіє. Представлення отримує дані з моделі і виводить їх у зручному і зрозумілому вигляді.

Основні функції представлення:

Відображення даних: Основна функція представлення полягає у відображенні даних, отриманих з моделі. Це можуть бути форми, таблиці, графіки або інші елементи інтерфейсу, які презентують дані користувачеві.

Взаємодія з користувачем: Представлення забезпечує інтерфейс для взаємодії з користувачем, дозволяючи йому вводити, редагувати і видаляти дані. Воно включає такі елементи, як кнопки, поля вводу, меню і інші інтерактивні елементи.

Оновлення даних: Представлення може динамічно оновлюватися при зміні даних у моделі. Це може відбуватися через сповіщення від моделі або через виклики з контролера, який оновлює представлення у відповідь на дії користувача.

Роль представлення в MVC:

Представлення є компонентом, який зосереджений виключно на презентації даних і взаємодії з користувачем. Воно не повинно містити бізнес-логіки або логіки управління даними, яка відноситься до моделі. Це розділення дозволяє створювати

більш гнучкі та підтримувані системи, де кожен компонент відповідає за свою конкретну роль.

2.2.3 Контролер (Controller) в архітектурі MVC

Контролер (Controller) в архітектурі MVC виступає посередником між моделлю (Model) та представленням (View). Він відповідає за обробку вхідних запитів користувачів, взаємодію з моделлю для отримання або зміни даних і вибір відповідного представлення для відображення даних користувачеві.

Основні функції контролера:

Обробка вхідних запитів: Контролер отримує вхідні запити від користувачів (наприклад, через URL-адреси), обробляє їх та визначає, які дії потрібно виконати. Це може включати обробку даних форми, маршрутизацію запитів і виконання відповідних методів моделі.

Взаємодія з моделлю: Контролер звертається до моделі для отримання необхідних даних або для виконання дій, які змінюють стан даних. Він викликає методи моделі для отримання, збереження, оновлення або видалення даних у відповідь на запити користувачів.

Вибір представлення: Після обробки запиту і взаємодії з моделлю, контролер визначає, яке представлення слід використовувати для відображення результату користувачеві. Він передає дані від моделі до представлення для відображення.

Роль контролера в MVC:

Контролер виступає в ролі координатора, який забезпечує узгоджену роботу між моделлю і представленням. Він не повинен містити складної бізнес-логіки або логіки відображення – ці аспекти розподіляються між моделлю і представленням відповідно. Контролер зосереджується на обробці запитів та управлінні потоком даних у додатку.

Переваги використання контролера:

Розділення обов'язків: Контролер розділяє обов'язки між моделлю та представленням, що сприяє чистішій і більш структурованій архітектурі. Це дозволяє кожному компоненту зосередитися на своїй конкретній ролі. Зручність у

підтримці: Оскільки контролер координує взаємодію між іншими компонентами, зміни у вимогах або функціональності можуть бути впроваджені локально, без необхідності змін у всіх компонентах системи. Покращене тестування: Окреме тестування контролера дозволяє перевіряти логіку обробки запитів і взаємодію з моделлю та представленням, що полегшує виявлення та виправлення помилок.

2.2.4 Переваги архітектури MVC

Архітектура MVC (Model-View-Controller) є популярною завдяки своїй здатності розділяти додаток на три незалежні компоненти, кожен з яких відповідає за різні аспекти функціональності. Це розділення сприяє більш ефективному управлінню, підтримці і масштабованості програмного забезпечення. Давайте детально розглянемо основні переваги цієї архітектури.

Розділення обов'язків

Однією з головних переваг MVC є чітке розділення обов'язків між трьома компонентами: Модель, Представлення і Контролер. Кожен з цих компонентів має свою чітко визначену роль:

Модель управляє даними та бізнес-логікою.

Представлення відповідає за відображення даних користувачеві.

Контролер обробляє вхідні запити та координує взаємодію між Моделлю та Представленням. Це розділення дозволяє розробникам зосередитися на окремих аспектах додатка, що спрощує процес розробки і підтримки.

Підвищена гнучкість і масштабованість.

Завдяки розділенню на три незалежні компоненти, MVC забезпечує високу гнучкість. Зміни в одному компоненті, наприклад, у Представленні, не впливають на інші компоненти. Це полегшує масштабування додатка і додавання нових функцій без необхідності переписувати інші частини коду.

Покращене тестування

MVC сприяє покращеному тестуванню додатків. Кожен компонент можна тестувати окремо:

Модель можна тестувати для перевірки правильності бізнес-логіки і обробки даних.

Представлення можна тестувати для перевірки коректності відображення даних і взаємодії з користувачем.

Контролер можна тестувати для перевірки логіки обробки запитів і взаємодії з моделлю і представленням.

Це полегшує виявлення і виправлення помилок, що підвищує якість кінцевого продукту.

Спрощена підтримка і розвиток.

Чітке розділення обов'язків сприяє спрощеній підтримці і розвитку додатка. Коли кожен компонент відповідає за свій набір завдань, зміни можна вносити локально, без впливу на всю систему. Це робить процес підтримки і додавання нових функцій швидшим і менш ризикованим.

Підвищена повторне використання коду.

Компоненти MVC можна легко повторно використовувати в різних частинах додатка або в інших додатках. Наприклад, модель, що управляє даними і логікою, можна використовувати з різними представленнями. Це підвищує ефективність розробки і знижує вартість створення нових функцій.

Підтримка різних представлень.

MVC дозволяє використовувати різні представлення для однієї і тієї ж моделі. Наприклад, дані можуть бути представлені у вигляді веб-сторінки, мобільного додатка або API. Це особливо корисно в сучасних додатках, де необхідно підтримувати різні платформи і пристрої.

Полегшена співпраця в команді.

У великих командах розробників MVC сприяє полегшенню співпраці. Різні команди можуть працювати над різними компонентами додатка незалежно один від одного. Наприклад, одна команда може займатися розробкою моделей і бізнес-логіки, тоді як інша – створенням представлень і інтерфейсу користувача.

2.2.5 Недоліки архітектури MVC

Архітектура MVC (Model-View-Controller) має безліч переваг, але, як і будь-який інший підхід до розробки програмного забезпечення, вона має і свої недоліки. Розглянемо основні з них:

Складність розробки:

Одним із значних недоліків MVC є підвищена складність розробки в порівнянні з традиційними підходами. Це пов'язано з тим, що розробникам потрібно мати глибоке розуміння того, як взаємодіють між собою компоненти Моделі, Представлення і Контролера. Розподіл функціональності між трьома компонентами може бути складним завданням, особливо для новачків.

Крива навчання: Нові розробники можуть зіткнутися з високою кривою навчання, оскільки вони повинні навчитися працювати з трьома окремими компонентами і розуміти, як вони взаємодіють між собою.

Складність структури: Структура MVC може бути більш складною у порівнянні з простішими архітектурами, що ускладнює розробку і налагодження.

Надмірне розділення:

Хоча розділення обов'язків є великою перевагою, воно може також стати недоліком, якщо не впроваджувати його належним чином. Надмірне розділення може призвести до збільшення кількості класів і методів, що ускладнює розробку та підтримку коду.

Ускладнення коду: Велика кількість класів і методів може ускладнити навігацію по коду і зрозуміння, як система працює в цілому.

Підтримка синхронізації: Забезпечення синхронізації між компонентами може бути складним завданням, оскільки зміни в одному компоненті можуть вимагати змін в інших.

Витрати на підтримку:

MVC може вимагати більше ресурсів для підтримки і розвитку в порівнянні з іншими архітектурами. Це пов'язано з необхідністю підтримки окремих компонентів і забезпечення їхньої узгодженої роботи.

Час і зусилля: Підтримка трьох окремих компонентів може вимагати більше часу і зусиль, ніж підтримка монолітної структури.

Витрати на навчання: Розробники повинні бути добре ознайомлені з концепціями MVC, що може збільшити витрати на навчання та підготовку персоналу.

Проблеми з продуктивністю:

У деяких випадках використання MVC може призводити до проблем з продуктивністю, особливо в додатках з високими вимогами до швидкості обробки даних.

Перевантаження контролера: Контролер може стати вузьким місцем, оскільки він обробляє всі вхідні запити і координує взаємодію між моделлю та представленням.

Додаткові витрати: Розділення обов'язків може призводити до додаткових витрат на обробку і передачу даних між компонентами.

Можливість недоузгодження:

Однією з проблем може бути недоузгодження між компонентами. Якщо модель, представлення і контролер не добре узгоджені, це може призвести до помилок і некоректної роботи системи.

Неповна інтеграція: Незабезпечення належної інтеграції між компонентами може призвести до проблем з передачею даних і обробкою запитів.

Відсутність координації: Недостатня координація між розробниками, які працюють над різними компонентами, може призвести до виникнення конфліктів і помилок.

2.3 Розробка архітектури сервісу

Розробка інформаційної системи для обліку абітурієнтів є важливим завданням, яке сприяє ефективній організації та управлінню процесом вступу. Дана система повинна забезпечувати зручний та прозорий механізм для подання документів, обробки заявок, зберігання інформації про абітурієнтів, а також

надавати аналітичні та звітні функції для адміністраторів. Основною метою цієї роботи є створення архітектури сервісу обліку абітурієнтів кафедри на основі патерну проектування MVC (Model-View-Controller), що дозволить забезпечити структурованість, гнучкість і легкість підтримки системи.

Аналіз вимог

На етапі аналізу вимог були визначені основні функціональні і нефункціональні вимоги до системи. Основні функціональні вимоги включають:

Реєстрація та авторизація користувачів. Абітурієнти повинні мати можливість створювати облікові записи, входити в систему та відновлювати пароль.

Подання заявок. Система повинна дозволяти абітурієнтам подавати заявки на вступ, завантажувати необхідні документи та відстежувати статус їх обробки. Управління заявками. Адміністратори повинні мати можливість переглядати, обробляти та редагувати заявки абітурієнтів.

Зберігання даних. Система повинна забезпечувати надійне зберігання даних про абітурієнтів, їхні заявки та супутні документи.

Звіти та аналітика. Система повинна надавати можливість формування звітів та проведення аналізу даних.

Нефункціональні вимоги включають забезпечення безпеки даних, високу продуктивність, масштабованість та зручність користування.

Вибір архітектури

Для розробки сервісу обліку абітурієнтів було обрано архітектуру MVC (Model-View-Controller), яка забезпечує розділення обов’язків між трьома основними компонентами: Модель, Представлення і Контролер.

Модель (Model): відповідає за управління даними та бізнес-логікою. У нашій системі модель буде відповідати за зберігання інформації про абітурієнтів, їхні заявки та інші супутні дані.

Представлення (View): відповідає за відображення даних користувачеві. Представлення буде включати інтерфейси для реєстрації, подання заявок, перегляду статусу заявок та інші форми взаємодії з користувачем.

Контролер (Controller): виступає посередником між Моделлю та Представленням. Контролер буде обробляти вхідні запити від користувачів, взаємодіяти з моделлю для отримання або зміни даних і передавати результати до представлення для відображення.

2.4 Опис інструментів розробки

2.4.1 C#

C# (читається "сі-шарп") є сучасною, об'єктно-орієнтованою мовою програмування, розробленою компанією Microsoft як частина їхньої .NET платформи. Вона поєднує в собі простоту та потужність, що робить її ідеальним вибором для розробки широкого спектру застосунків, включаючи веб-сервіси. У цій частині ми розглянемо основні характеристики мови C#, її переваги та причини, чому саме C# було обрано для розробки веб-сервісу обліку абітурієнтів кафедри.

Основні характеристики мови C#

- **Об'єктно-орієнтована природа**: C# підтримує основні принципи об'єктно-орієнтованого програмування (ООП) — інкапсуляцію, наслідування, поліморфізм та абстракцію. Це дозволяє розробникам створювати модульний, зручний для супроводу та повторного використання код.
- **Типізація**: C# є сильно типізованою мовою, що забезпечує виявлення помилок на ранніх етапах розробки. Це сприяє підвищенню надійності та якості коду.
- **Платформа .NET**: C# тісно інтегрована з платформою .NET, яка надає великий набір бібліотек і фреймворків для розробки різноманітних застосунків, від десктопних до веб- та мобільних.
- **Асинхронне програмування**: C# підтримує асинхронне програмування за допомогою ключових слів async і await, що дозволяє створювати високопродуктивні та масштабовані застосунки.

- **LINQ (Language Integrated Query)**: C# має вбудовану підтримку LINQ, що спрощує роботу з колекціями даних та забезпечує зручний спосіб написання запитів до баз даних.

Переваги використання C# для веб-сервісу

- **Інтеграція з ASP.NET Core**: Однією з головних причин вибору C# для розробки веб-сервісу обліку абітурієнтів є тісна інтеграція з фреймворком ASP.NET Core. ASP.NET Core — це потужний, високопродуктивний фреймворк для створення сучасних веб-додатків та API, який підтримує кросплатформеність.
- **Безпека**: C# і .NET платформа забезпечують високий рівень безпеки завдяки вбудованим засобам для обробки помилок, аутентифікації та авторизації, шифрування даних та захисту від загроз, таких як SQL-ін'єкції та XSS-атаки.
- **Висока продуктивність**: Завдяки оптимізації платформи .NET та можливостям асинхронного програмування, веб-сервіси на C# демонструють високу продуктивність та здатність обробляти велику кількість запитів одночасно.
- **Масштабованість**: C# дозволяє легко масштабувати веб-сервіси для обслуговування великої кількості користувачів і даних. Це досягається завдяки можливостям розподіленого зберігання даних і асинхронної обробки запитів.
- **Зручність розробки та підтримки**: C# має зрозумілий і читабельний синтаксис, що спрощує процес розробки і підтримки коду. Інтеграція з середовищем розробки Visual Studio забезпечує зручність написання, налагодження і тестування коду.
- **Ком'юніті та підтримка**: C# має велику спільноту розробників і сильну підтримку від Microsoft. Це забезпечує доступ до численних ресурсів, таких як документація, навчальні матеріали та форуми, де можна знайти відповіді на будь-які питання.

Вибір мови програмування C# для розробки веб-сервісу обліку абітурієнтів кафедри обумовлений її потужними можливостями, високою продуктивністю, надійністю та зручністю розробки. Використання C# у поєднанні з фреймворком ASP.NET Core дозволяє створити сучасний, масштабований і безпечний веб-сервіс, що відповідає всім вимогам сучасних інформаційних систем. Цей вибір також

забезпечує легкість підтримки та розвитку системи в майбутньому, що є критично важливим для успішного функціонування і розвитку кафедри.

2.4.2 Visual Studio

Для розробки нашого веб-сервісу обліку абітурієнтів кафедри ми обрали Visual Studio з кількох важливих причин, які роблять це середовище розробки ідеальним для створення складних та функціональних веб-додатків.

Перш за все, Visual Studio пропонує потужний і зручний інтегрований інтерфейс розробки (IDE), який підтримує всі етапи розробки програмного забезпечення — від написання коду до його тестування та налагодження. Це значно підвищує продуктивність розробників, оскільки всі необхідні інструменти зібрані в одному місці, що дозволяє ефективно керувати проектами та швидко знаходити і виправляти помилки.

Visual Studio має розширену підтримку мови програмування C# та платформи .NET, що є ключовими для нашого проекту. Інтеграція з .NET Core, зокрема ASP.NET Core, забезпечує створення високопродуктивних, масштабованих і безпечних веб-додатків. Це середовище розробки пропонує широкий набір шаблонів і засобів для швидкого старту проектів, включаючи готові шаблони для веб-додатків, API, а також підтримку сучасних веб-технологій, таких як HTML, CSS і JavaScript.

Однією з найважливіших переваг Visual Studio є його потужні засоби для налагодження та тестування. Visual Studio надає інструменти для інтерактивного налагодження, що дозволяють розробникам відслідковувати виконання коду в реальному часі, встановлювати точки перериву, переглядати змінні та виконувати детальний аналіз поведінки програми. Це значно спрощує виявлення і виправлення помилок, що, в свою чергу, підвищує надійність і якість кінцевого продукту.

Крім того, Visual Studio підтримує інтеграцію з системами контролю версій, такими як Git, що дозволяє ефективно керувати змінами в коді, працювати в команді та забезпечувати контроль за версіями проекту. Це особливо важливо для

великих проектів, де багато розробників можуть одночасно працювати над різними частинами системи.

Visual Studio також забезпечує підтримку DevOps процесів через інтеграцію з Azure DevOps та іншими сервісами CI/CD (безперервної інтеграції та безперервного розгортання). Це дозволяє автоматизувати процеси тестування і розгортання, що сприяє швидкому впровадженню змін і оновлень у виробниче середовище.

Завдяки своїй потужності, гнучкості та широкому спектру інструментів, Visual Studio стала незамінним інструментом для розробки нашого веб-сервісу обліку абітурієнтів кафедри. Вона забезпечує ефективний процес розробки, високу якість коду та зручність роботи для розробників, що є ключовими факторами успішного створення і підтримки складних програмних рішень.

2.4.3 HTML

HTML (HyperText Markup Language) — це основна мова розмітки, що використовується для створення та структурування веб-сторінок у Інтернеті. Вона є невід'ємною частиною веб-технологій, поряд з CSS і JavaScript. HTML описує структуру веб-сторінки за допомогою різноманітних тегів та атрибутів, що визначають різні елементи сторінки, такі як заголовки, абзаци, посилання, зображення, таблиці та інші компоненти.

Одна з ключових причин використання HTML для створення веб-сервісу обліку абітурієнтів кафедри полягає в його простоті та універсальності. HTML є стандартом, який підтримується всіма веб-браузерами, що гарантує, що веб-сторінки будуть правильно відображатися на будь-якому пристрої, незалежно від операційної системи чи веб-браузера. Це забезпечує доступність веб-сервісу для широкого кола користувачів.

HTML дозволяє розробникам створювати структуру веб-сторінок за допомогою семантичних тегів, таких як <header>, <footer>, <article>, <section> та інших. Ці теги не лише покращують читабельність коду, але й сприяють кращій

оптимізації для пошукових систем (SEO), що полегшує індексацію контенту веб-сервісу пошуковими роботами.

Однією з важливих особливостей HTML є його гнучкість та можливість інтеграції з іншими веб-технологіями. HTML дозволяє включати CSS для оформлення та стилізації сторінок, що робить їх візуально привабливими та зручними для користувачів. Крім того, HTML підтримує інтеграцію з JavaScript, що дозволяє додавати інтерактивні елементи та динамічний контент на веб-сторінки, забезпечуючи більш багатий користувацький досвід.

HTML також підтримує формати для введення даних, такі як форми, що дозволяє створювати інтерфейси для збору інформації від користувачів. Це особливо важливо для нашого веб-сервісу обліку абітурієнтів, оскільки форми використовуються для подання заявок, введення особистих даних та інших взаємодій з користувачами. Ще однією важливою перевагою HTML є його постійний розвиток та підтримка міжнародними організаціями, такими як World Wide Web Consortium (W3C). Останні версії HTML, включаючи HTML5, забезпечують розширені можливості, такі як вбудовані мультимедійні елементи (відео та аудіо), локальне зберігання даних, підтримка веб-аплікацій та багато іншого. Це дозволяє створювати сучасні та функціональні веб-додатки.

Використання HTML для створення нашого веб-сервісу обліку абітурієнтів кафедри забезпечує простоту, універсальність та доступність. Це основний будівельний блок для створення веб-сторінок, який у поєднанні з іншими веб-технологіями дозволяє створювати потужні, інтерактивні та ефективні веб-додатки. HTML забезпечує структурованість контенту, зручність взаємодії користувачів із сервісом та широкі можливості для інтеграції з іншими технологіями, що робить його незамінним інструментом для веб-розробки.

2.4.4 CSS

CSS (Cascading Style Sheets) — це мова стилів, яка використовується для опису вигляду та форматування документів, написаних мовами розмітки, такими як HTML. CSS визначає, як елементи HTML повинні відображатися на екрані,

папері або в інших медіа. Завдяки CSS розробники можуть створювати привабливі, зручні для користувачів веб-сторінки з консистентним дизайном і макетом.

Основна роль CSS полягає в тому, щоб відокремити зміст веб-сторінки (структуру, визначену HTML) від її оформлення. Це дозволяє розробникам створювати і змінювати вигляд веб-сторінок без необхідності змінювати їхній зміст. Використання CSS забезпечує ряд переваг, що робить його незамінним інструментом для створення веб-додатків, таких як наш веб-сервіс обліку абітурієнтів кафедри.

Однією з ключових переваг CSS є можливість централізованого управління стилями. CSS дозволяє визначити стилі в одному місці (наприклад, у зовнішньому файлі стилів) і застосовувати їх до багатьох елементів на різних сторінках. Це значно спрощує підтримку та оновлення стилів. Наприклад, зміна кольору фону або шрифту на всьому сайті може бути зроблена лише в одному місці, що економить час і зусилля розробників.

CSS надає широкий спектр інструментів для оформлення елементів веб-сторінки. Це включає налаштування кольорів, шрифтів, відступів, рамок, розмірів, розташування та багато іншого. Завдяки таким можливостям розробники можуть створювати візуально привабливі та зручні для користувачів інтерфейси. Крім того, CSS підтримує медіа-запити, що дозволяють створювати адаптивний дизайн, який автоматично підлаштовується під різні розміри екранів і пристроїв, що є критично важливим в епоху мобільних технологій.

CSS також підтримує концепцію каскадування стилів. Це означає, що стилі можуть успадковуватися від батьківських елементів до дочірніх, що спрощує управління стилями і забезпечує більш консистентний вигляд сторінки. Каскадування дозволяє комбінувати стилі з різних джерел (наприклад, користувацькі стилі, стилі за замовчуванням браузера та зовнішні стилі) та визначати, які з них мають пріоритет.

Ще однією важливою особливістю CSS є підтримка псевдокласів і псевдоелементів, які дозволяють застосовувати стилі до специфічних станів елементів або частин контенту. Наприклад, псевдоклас :hover можна

використовувати для зміни стилю елемента, коли курсор миші знаходиться над ним, що підвищує інтуїтивність і інтерактивність інтерфейсу користувача.

CSS також підтримує анімацію та переходи, що дозволяє створювати динамічні ефекти без використання JavaScript. Це може включати плавні зміни кольорів, розмірів або положення елементів, що робить взаємодію з веб-сторінкою більш привабливою та інтерактивною.

Для нашого веб-сервісу обліку абітурієнтів кафедри використання CSS є критично важливим. Воно дозволяє створити сучасний, естетично привабливий та зручний для користувачів інтерфейс, який відповідає вимогам сучасних веб-дизайн стандартів. CSS забезпечує гнучкість і зручність в управлінні стилями, дозволяє створювати адаптивний дизайн, який підлаштовується під різні пристрої, і надає можливості для створення інтерактивних елементів і анімацій.

Загалом, CSS є потужним інструментом для веб-розробників, що дозволяє відокремити зміст від оформлення, забезпечити гнучкість і зручність у розробці, та створювати привабливі, сучасні та функціональні веб-сторінки. Це робить CSS невід'ємною частиною нашого веб-сервісу обліку абітурієнтів кафедри та важливим компонентом у створенні якісних веб-додатків.

2.4.5 SQL Server

SQL Server — це система управління реляційними базами даних (РСУБД), розроблена компанією Microsoft. Вона використовується для зберігання та управління великими обсягами даних, забезпечуючи високий рівень продуктивності, безпеки і масштабованості. SQL Server є потужним інструментом для роботи з даними, що надає розробникам і адміністраторам баз даних широкий спектр функціональних можливостей для створення та управління базами даних.

SQL Server підтримує стандартну мову SQL (Structured Query Language), яка використовується для виконання запитів до бази даних. Це дозволяє розробникам легко створювати, читати, оновлювати і видаляти дані в базі даних за допомогою зрозумілих і інтуїтивно зрозумілих команд. SQL Server також підтримує складні

запити, транзакції і процедури, що дозволяє ефективно обробляти дані і забезпечувати цілісність і консистентність бази даних.

Однією з головних причин використання SQL Server для нашого веб-сервісу обліку абітурієнтів кафедри є його надійність і продуктивність. SQL Server здатний обробляти великі обсяги даних і виконувати складні запити з високою швидкістю, що є критично важливим для ефективного функціонування системи обліку. Крім того, SQL Server надає потужні засоби для оптимізації продуктивності, такі як індексація, кешування і розподілене зберігання даних, що дозволяє забезпечити швидкий доступ до даних і мінімізувати час виконання запитів.

SQL Server також забезпечує високий рівень безпеки даних. Він надає широкий спектр засобів для аутентифікації і авторизації користувачів, шифрування даних і захисту від несанкціонованого доступу. Це дозволяє гарантувати безпеку конфіденційних даних абітурієнтів і забезпечити відповідність вимогам щодо захисту персональних даних.

Ще однією важливою перевагою SQL Server є його інтеграція з іншими продуктами Microsoft і підтримка широкого спектру інструментів для розробки і управління базами даних. SQL Server добре інтегрується з середовищем розробки Visual Studio, що дозволяє розробникам легко створювати і тестувати запити до бази даних, використовувати зручні інструменти для налагодження і оптимізації коду. Крім того, SQL Server підтримує інструменти для резервного копіювання і відновлення даних, що забезпечує надійність і доступність даних у випадку збоїв або аварій.

SQL Server також підтримує можливості для аналізу даних і створення звітів. Вбудовані інструменти для бізнес-аналітики, такі як SQL Server Analysis Services (SSAS) і SQL Server Reporting Services (SSRS), дозволяють аналізувати дані, створювати складні звіти і візуалізувати результати аналізу. Це особливо корисно для прийняття обґрунтованих рішень на основі даних, зібраних у процесі обліку абітурієнтів.

Для нашого веб-сервісу обліку абітурієнтів кафедри використання SQL Server забезпечує надійне зберігання, швидкий доступ і високий рівень безпеки

даних. Завдяки його продуктивності, масштабованості і інтеграції з іншими інструментами Microsoft, SQL Server є оптимальним вибором для створення потужної та ефективної системи управління базами даних. Це дозволяє розробникам і адміністраторам ефективно управляти даними, забезпечувати безпеку і доступність інформації, а також аналізувати і використовувати дані для прийняття стратегічних рішень.

2.4.6 JavaScript

JavaScript — це високорівнева, динамічна мова програмування, яка використовується переважно для створення інтерактивних та динамічних веб-сторінок. Спочатку розроблена як мова сценаріїв для браузерів, JavaScript значно еволюціонувала і тепер використовується як на стороні клієнта, так і на стороні сервера. JavaScript є невід'ємною частиною сучасного веб-розробки, що дозволяє створювати багатофункціональні, інтерактивні веб-додатки.

Одна з головних причин використання JavaScript для нашого веб-сервісу обліку абітурієнтів кафедри полягає в його здатності додавати інтерактивність і динамічність до веб-сторінок. За допомогою JavaScript можна реагувати на дії користувачів в режимі реального часу, наприклад, обробляти кліки, вводити дані у форми, динамічно змінювати контент сторінки без її перезавантаження. Це значно покращує користувацький досвід і робить веб-додаток більш інтерактивним та зручним.

JavaScript підтримує широкий спектр функціональних можливостей, що дозволяє створювати складні логічні структури і реалізовувати бізнес-логіку безпосередньо в браузері користувача. Наприклад, можна здійснювати валідацію даних на стороні клієнта перед їх відправкою на сервер, що допомагає зменшити навантаження на сервер і покращує ефективність роботи веб-додатка.

Завдяки своїй гнучкості та універсальності, JavaScript підтримує численні бібліотеки та фреймворки, такі як React, Angular, Vue.js та інші. Ці інструменти значно спрощують розробку сучасних веб-додатків, надаючи готові компоненти та рішення для типових задач. Використання цих бібліотек дозволяє розробникам

швидше створювати функціональні та масштабовані веб-додатки з меншою кількістю коду і більш високою якістю.

JavaScript також дозволяє здійснювати асинхронні операції за допомогою технологій, таких як AJAX (Asynchronous JavaScript and XML), Fetch API та WebSockets. Це дає можливість оновлювати контент веб-сторінок в режимі реального часу без необхідності перезавантаження сторінки. Такі можливості особливо корисні для створення інтерактивних форм, чату, живих оновлень даних та інших динамічних елементів веб-додатка.

Ще однією важливою перевагою JavaScript є його популярність та підтримка великою спільнотою розробників. Існує велика кількість ресурсів, документації, навчальних матеріалів та інструментів, що робить процес навчання і розробки на JavaScript зручним і ефективним. Крім того, завдяки активній спільноті постійно з'являються нові бібліотеки, фреймворки та інструменти, що дозволяють розробникам вирішувати нові задачі та вдосконалювати свої проекти.

JavaScript також підтримується всіма сучасними веб-браузерами, що забезпечує його сумісність та доступність на різних платформах і пристроях. Це означає, що веб-додатки, розроблені з використанням JavaScript, можуть працювати на будь-якому пристрої, включаючи настільні комп'ютери, ноутбуки, планшети та смартфони, забезпечуючи широкий доступ до веб-сервісу обліку абітурієнтів кафедри.

Для нашого веб-сервісу обліку абітурієнтів кафедри використання JavaScript дозволяє створювати інтерактивні, динамічні та зручні для користувачів веб-сторінки. Це забезпечує високий рівень взаємодії з користувачем, покращує користувацький досвід і робить веб-додаток більш привабливим та функціональним. Завдяки своїм потужним можливостям, гнучкості та підтримці сучасних веб-технологій, JavaScript є ключовим інструментом для створення ефективного та сучасного веб-сервісу.

2.4.7 Bootstrap

Bootstrap — це популярний фреймворк для розробки інтерфейсів користувача, який надає набір інструментів для створення адаптивних і зручних веб-сторінок. Він включає в себе готові CSS та JavaScript компоненти, що значно спрощують розробку і економлять час.

Основні причини використання Bootstrap для нашого веб-сервісу обліку абітурієнтів кафедри включають його адаптивний дизайн, який автоматично підлаштовується під різні розміри екранів і пристроїв. Це забезпечує доступність і зручність використання веб-сервісу на настільних комп'ютерах, планшетах і смартфонах.

Bootstrap також пропонує багатий набір готових стилів і компонентів, таких як кнопки, форми, навігаційні панелі, модальні вікна та інші, що дозволяє швидко створювати професійний і консистентний дизайн. Крім того, Bootstrap легко налаштовується, що дозволяє адаптувати його під конкретні вимоги проекту.

Завдяки широкій підтримці спільноти, Bootstrap має велику кількість документації, прикладів і ресурсів, що полегшує процес навчання і використання фреймворку. Це робить Bootstrap ідеальним вибором для швидкої та ефективної розробки сучасних веб-додатків.

2.4.8 ASP.NET

ASP.NET — це фреймворк для розробки веб-додатків, створений компанією Microsoft. Він дозволяє розробникам створювати динамічні, інтерактивні веб-сайти і веб-сервіси, використовуючи мови програмування, такі як C# та VB.NET. ASP.NET є частиною платформи .NET, що забезпечує широкий спектр інструментів і бібліотек для розробки.

Одна з головних причин використання ASP.NET для нашого веб-сервісу обліку абітурієнтів кафедри — це його продуктивність і масштабованість. ASP.NET підтримує оптимізацію продуктивності за рахунок технологій, таких як

кешування, управління станом і компіляція на стороні сервера, що дозволяє ефективно обробляти великі обсяги даних і забезпечувати швидкий доступ до них.

ASP.NET також забезпечує високий рівень безпеки завдяки вбудованим засобам для аутентифікації і авторизації користувачів, шифрування даних і захисту від загроз, таких як SQL-ін'єкції та XSS-атаки. Це критично важливо для захисту конфіденційних даних абітурієнтів.

Однією з ключових особливостей ASP.NET є його інтеграція з іншими продуктами Microsoft, такими як Visual Studio, SQL Server і Azure. Це забезпечує зручне середовище для розробки, тестування та розгортання веб-додатків. Visual Studio надає потужні інструменти для написання коду, відлагодження і управління проектами, що значно спрощує процес розробки.

ASP.NET також підтримує модель MVC (Model-View-Controller), що сприяє розділенню логіки програми, інтерфейсу користувача і управління даними. Це робить код більш організованим, легшим для підтримки та масштабування.

Завдяки своїм можливостям і інтеграції з іншими технологіями, ASP.NET є потужним інструментом для створення надійних і безпечних веб-додатків. Він дозволяє ефективно управляти даними, забезпечувати безпеку і надавати користувачам зручний і інтуїтивний інтерфейс. Це робить ASP.NET ідеальним вибором для розробки нашого веб-сервісу обліку абітурієнтів кафедри.

3. РЕАЛІЗАЦІЯ ФУНКЦІОНАЛЬНОСТІ ВЕБ-СЕРВІСУ

3.1 Розробка основних функцій веб-сервісу

Створення веб-сервісу обліку абітурієнтів кафедри є складним і багатофункціональним процесом, який вимагає детального планування та реалізації. Основні функції цього сервісу повинні забезпечувати зручний інтерфейс для взаємодії абітурієнтів із системою, полегшуючи процес подачі заявок, управління даними та комунікації з адміністрацією. Нижче наведено детальний опис кожної з основних функцій цього веб-сервісу.

3.1.1 Реєстрація користувача(абітурієнта)

Реєстрація користувача є першим і найважливішим кроком у взаємодії абітурієнтів із веб-сервісом. Ця функція дозволяє новим абітурієнтам створити свій обліковий запис у системі, що забезпечує персоналізований доступ до функцій веб-сервісу. Реєстрація включає введення особистих даних, таких як ім'я, прізвище, електронна адреса, номер телефону та пароль. Під час реєстрації система перевіряє введені дані на валідність, щоб уникнути помилок і забезпечити правильність збереженої інформації. Після успішної реєстрації на електронну адресу користувача надсилається лист для підтвердження, що гарантує справжність введених даних і активує обліковий запис.

Ця функція є необхідною, оскільки забезпечує безпечний і персоналізований доступ до системи. Реєстрація дозволяє створити унікальні облікові записи для кожного абітурієнта, що спрощує процес подачі заявок, редагування даних та комунікації з адміністрацією кафедри. Крім того, підтвердження електронної адреси підвищує рівень безпеки та запобігає створенню фальшивих облікових записів.

В наведеному нижче коді показано як я реалізував цю функцію у себе на веб-сервісі:

```
@{
ViewData["Title"] = "Реєстрація";
}

<div class="form-container">
<h2>@ViewData["Title"]</h2>
<form method="post">
<div class="form-group">
<label for="Email">Пошта</label>
<input type="email" class="form-control" id="Email" name="Email" required>
</div>
<div class="form-group">
<label for="Password">Пароль</label>
<input type="password" class="form-control" id="Password" name="Password" required>
</div>
<div class="form-group">
<label for="ConfirmPassword">Підтвердіть пароль</label>
<input type="password" class="form-control" id="ConfirmPassword" name="ConfirmPassword"
required>
</div>
<button type="submit" class="btn btn-primary">Зареєструватися</button>
</form>
</div>
```

Цей код створює веб-сторінку для реєстрації користувачів. Користувачі можуть ввести свою адресу електронної пошти та пароль, щоб створити обліковий запис на веб-сайті. Після заповнення форми користувачі натискають кнопку "Зареєструватися", щоб надіслати свої дані.

3.1.2 Вхід користувача(абітурієнта)

Функція входу користувача (абітурієнта) на веб-сервісі обліку абітурієнтів кафедри є однією з ключових функцій, яка забезпечує персоналізований доступ до системи.

Після того, як абітурієнт зареєструється на веб-сервісі, йому потрібно увійти в свій особистий кабінет, щоб мати можливість користуватися всіма функціями сервісу. Процес входу дуже простий і зручний. Абітурієнт вводить свою електронну адресу та пароль, які він зазначив під час реєстрації. Ця інформація перевіряється системою для підтвердження, що це дійсно зареєстрований користувач. Якщо введені дані правильні, абітурієнт отримує доступ до свого особистого кабінету.

Особистий кабінет абітурієнта є важливим елементом сервісу, оскільки саме тут він може заповнювати свої особисті дані, подавати заявки, редагувати інформацію та спілкуватися з адміністрацією. Вхід до особистого кабінету забезпечує захист особистої інформації абітурієнтів, оскільки тільки вони мають доступ до своїх даних. Це допомагає підтримувати конфіденційність та безпеку інформації, що є надзвичайно важливим у процесі обліку та подачі заявок.

Отже, функція входу користувача на веб-сервісі є не тільки зручною, але й критично важливою для забезпечення безпеки та конфіденційності даних абітурієнтів. Вона дозволяє користувачам легко отримувати доступ до всіх можливостей сервісу і ефективно взаємодіяти з адміністрацією кафедри.

Приклад коду цієї функції:

```
@{
ViewData["Title"] = "Увійти";
}

<div class="form-container">
<h2>@ViewData["Title"]</h2>
<form asp-route-returnurl="@ViewData["ReturnUrl"]" method="post">
<div class="form-group">
<label for="Email">Пошта</label>
<input type="email" class="form-control" id="Email" name="Email" required>
</div>
<div class="form-group">
<label for="Password">Пароль</label>
<input type="password" class="form-control" id="Password" name="Password" required>
</div>
<div class="form-group form-check">
<input type="checkbox" class="form-check-input" id="RememberMe" name="RememberMe">
<label class="form-check-label" for="RememberMe">Залишатися в системі</label>
</div>
<button type="submit" class="btn btn-primary">Увійти</button>
</form>
</div>
```

Цей код відображає форму для входу користувача на веб-сайті. Користувач повинен ввести свою електронну адресу (Email) і пароль (Password). Є також чекбокс, за допомогою якого користувач може вибрати опцію "Залишатися в системі" (RememberMe), щоб залишитися автономною після закриття веб-сайту. Після натискання кнопки "Увійти" (Sign In), введені дані будуть відправлені на

сервер для перевірки, і користувач буде авторизований у системі, якщо введені дані вірні.

3.1.3 Заповнення даних необхідних для подачі за’явки

Функція "Заповнення даних для подачі заявки" на нашому веб-сервісі обліку абітурієнтів кафедри дозволяє користувачам вводити інформацію, необхідну для подання заявки на вступ до навчального закладу. Ця функція допомагає абітурієнтам зручно та ефективно надсилати свої дані, спрощуючи процес вступу.

Після входу в особистий кабінет користувач може перейти до розділу "Подати заявку" або подібного, де буде доступна форма для заповнення. Форма може включати поля для особистих даних, контактної інформації, академічного запису (наприклад, попередній досвід навчання), інших важливих деталей, що вимагаються для розгляду заявки.

Користувач заповнює ці дані відповідно до вимог, після чого може переглянути та відредагувати інформацію перед відправленням. Після надсилання заявки або збереження її як чернетку, користувач може отримати підтвердження про успішне подання і перевірити статус заявки у своєму особистому кабінеті.

Ця функція дозволяє абітурієнтам зручно та швидко надсилати свої дані для вступу, спрощуючи процес подачі заявки та забезпечуючи надійну передачу інформації.

Приклад коду для реалізації функції:

```
@model AdmissionSystem.Models.Applicant

@{
ViewData["Title"] = "Details";
}

<h2>Деталі абітурієнта</h2>

<div>
<h4>Абітурієнт</h4>
<hr />
<dl class="row">
<dt class="col-sm-2">
```

```
Ім'я
</dt>
<dd class="col-sm-10">
@Html.DisplayFor(model => model.FirstName)
</dd>
<dt class="col-sm-2">
Прізвище
</dt>
<dd class="col-sm-10">
@Html.DisplayFor(model => model.LastName)
</dd>
<dt class="col-sm-2">
Email
</dt>
<dd class="col-sm-10">
@Html.DisplayFor(model => model.Email)
</dd>
<dt class="col-sm-2">
Номер телефону
</dt>
<dd class="col-sm-10">
@Html.DisplayFor(model => model.PhoneNumber)
</dd>
<dt class="col-sm-2">
Результат НМТ
</dt>
<dd class="col-sm-10">
@Html.DisplayFor(model => model.Group)
</dd>
</dl>
</div>

<div>
<h4>По вашим результатам НМТ ви можете подати заявку на такі Спеціальності:</h4>
<div style="display: flex; align-items: center;">
<select id="specialtyDropdown" class="form-control" style="width: auto; display: inline-block;
margin-right: 10px;">
</select>
<button class="btn btn-primary" onclick="applyForSpecialty()">Подати заявку на цю
спеціальність</button>
</div>
</div>

<div style="margin-top: 100px;">
<a asp-action="Edit" asp-route-id="@Model.Id" class="btn btn-warning">Редагувати</a> |
<a asp-action="Index" class="btn btn-secondary">Назад до списку</a>
</div>

@section Scripts {
<script>
document.addEventListener("DOMContentLoaded", function () {
var nmtScore = @Model.Group;
```

```
var specialties = [];

if (nmtScore >= 170) {
specialties = [
"Програмна інженерія",
"Комп'ютерна наука",
"Інформаційні системи та технології",
"Кібербезпека",
"Інженерія даних",
"Розробка програмного забезпечення",
"Мережеві технології",
"Веб-розробка",
"Мобільна розробка",
"Штучний інтелект"
];
} else if (nmtScore >= 150) {
specialties = [
"Програмна інженерія",
"Комп'ютерна наука",
"Інформаційні системи та технології",
"Кібербезпека"
];
} else if (nmtScore >= 120) {
specialties = [
"Програмна інженерія",
"Комп'ютерна наука"
];
}

var dropdown = document.getElementById("specialtyDropdown");
specialties.forEach(function (specialty) {
var option = document.createElement("option");
option.text = specialty;
option.value = specialty;
dropdown.add(option);
});
});

function applyForSpecialty() {
var selectedSpecialty = document.getElementById("specialtyDropdown").value;
alert("Ви подали заявку на спеціальність: " + selectedSpecialty);
}
</script>
}
```

Цей код є частиною веб-сторінки для відображення деталей про абітурієнта на веб-сервісі обліку абітурієнтів кафедри. Давайте розберемо, що робить кожна частина коду.

@model AdmissionSystem.Models.Applicant: Ця строка вказує, що модель для цієї сторінки є об'єктом класу Applicant з простору імен AdmissionSystem.Models.

@{ ViewData["Title"] = "Details"; }: Встановлює заголовок сторінки як "Деталі".

<h2>Деталі абітурієнта</h2>: Відображує заголовок "Деталі абітурієнта" на сторінці.

... : Виводить основну інформацію про абітурієнта, таку як ім'я, прізвище, email, номер телефону та результати НМТ.

... : Показує список спеціальностей, на які можна подати заявку, залежно від результатів НМТ.

... : Виводить посилання для редагування даних абітурієнта та повернення до списку абітурієнтів.

@section Scripts { ... }: В цьому розділі розміщено JavaScript-код, який виконується на сторінці. В даному випадку скрипт генерує список спеціальностей залежно від результатів НМТ і дозволяє користувачеві подати заявку на обрану спеціальність.

Цей код створює інтерактивну сторінку, яка дозволяє абітурієнтам переглядати свої дані та подавати заявки на обрані спеціальності залежно від їх результатів НМТ.

3.1.4 Редагування даних

Функція "Редагування даних" на нашому веб-сервісі обліку абітурієнтів кафедри дозволяє користувачам змінювати свої особисті дані, які вони ввели під час реєстрації або пізніше. Ця функція є важливою для забезпечення точності та актуальності інформації, яку мають наші користувачі у системі.

Після входу в особистий кабінет користувач може перейти до розділу "Редагувати дані" або подібного, де буде доступна форма для редагування особистих даних. Форма може містити поля для редагування імені, прізвища, електронної пошти, номера телефону та іншої інформації, яка є доступною для редагування.

Після внесення змін користувач може переглянути та підтвердити нові дані перед збереженням. Після збереження система оновить дані користувача і повідомить про успішне оновлення. Користувач також може скасувати зміни і повернутися до попереднього стану своїх даних.

Ця функція дозволяє користувачам управляти своєю особистою інформацією на сайті, забезпечуючи їм можливість змінювати та оновлювати дані в будь-який зручний момент.

Реалізація цієї функції наведена в коді нижче:

```
@model AdmissionSystem.Models.Applicant

@{
ViewData["Title"] = "Details";
}

<h2>Деталі абітурієнта</h2>

<div>
<h4>Абітурієнт</h4>
<hr />
<dl class="row">
<dt class="col-sm-2">
Ім'я
</dt>
<dd class="col-sm-10">
@Html.DisplayFor(model => model.FirstName)
</dd>
<dt class="col-sm-2">
Прізвище
</dt>
<dd class="col-sm-10">
@Html.DisplayFor(model => model.LastName)
</dd>
<dt class="col-sm-2">
Email
</dt>
<dd class="col-sm-10">
@Html.DisplayFor(model => model.Email)
</dd>
<dt class="col-sm-2">
Номер телефону
</dt>
<dd class="col-sm-10">
@Html.DisplayFor(model => model.PhoneNumber)
```

```
</dd>
<dt class="col-sm-2">
Результат НМТ
</dt>
<dd class="col-sm-10">
@Html.DisplayFor(model => model.Group)
</dd>
</dl>
</div>

<div>
<h4>По вашим результатам НМТ ви можете подати заявку на такі Спеціальності:</h4>
<div style="display: flex; align-items: center;">
<select id="specialtyDropdown" class="form-control" style="width: auto; display: inline-block;
margin-right: 10px;">
</select>
<button class="btn btn-primary" onclick="applyForSpecialty()">Подати заявку на цю
спеціальність</button>
</div>
</div>

<div style="margin-top: 100px;">
<a asp-action="Edit" asp-route-id="@Model.Id" class="btn btn-warning">Редагувати</a> |
<a asp-action="Index" class="btn btn-secondary">Назад до списку</a>
</div>

@section Scripts {
<script>
document.addEventListener("DOMContentLoaded", function () {
var nmtScore = @Model.Group;
var specialties = [];

if (nmtScore >= 170) {
specialties = [
"Програмна інженерія",
"Комп'ютерна наука",
"Інформаційні системи та технології",
"Кібербезпека",
"Інженерія даних",
"Розробка програмного забезпечення",
"Мережеві технології",
"Веб-розробка",
"Мобільна розробка",
"Штучний інтелект"
];
} else if (nmtScore >= 150) {
specialties = [
"Програмна інженерія",
```

```
"Комп'ютерна наука",
"Інформаційні системи та технології",
"Кібербезпека"
];
} else if (nmtScore >= 120) {
specialties = [
"Програмна інженерія",
"Комп'ютерна наука"
];
}

var dropdown = document.getElementById("specialtyDropdown");
specialties.forEach(function (specialty) {
var option = document.createElement("option");
option.text = specialty;
option.value = specialty;
dropdown.add(option);
});
});

function applyForSpecialty() {
var selectedSpecialty = document.getElementById("specialtyDropdown").value;
alert("Ви подали заявку на спеціальність: " + selectedSpecialty);
}
</script>
}
```

Цей код відображає деталі абітурієнта на веб-сайті та дозволяє користувачам подавати заявки на спеціальності в залежності від їх результатів НМТ. Давайте розглянемо, як це працює крок за кроком:

@model AdmissionSystem.Models.Applicant: Цей рядок вказує на тип моделі, яка використовується для відображення даних абітурієнта на сторінці.

@{ ViewData["Title"] = "Details"; }: Встановлює заголовок сторінки.

<h2>Деталі абітурієнта</h2>: Виводить заголовок "Деталі абітурієнта" на сторінці.

<dl class="row">...</dl>: Відображає основну інформацію про абітурієнта, таку як ім'я, прізвище, email, номер телефону та результат НМТ.

... : Показує список спеціальностей, на які користувач може подати заявку в залежності від результату НМТ.

... : Надає можливість редагувати дані абітурієнта та повернутися до списку абітурієнтів.

@section Scripts { ... }: У цьому розділі міститься JavaScript-код, який виконується на сторінці. Скрипт генерує список спеціальностей, на які можна подати заявку, в залежності від результатів НМТ, та дозволяє користувачеві подати заявку на обрану спеціальність.

Цей код створює інтерактивну сторінку, яка відображає інформацію про абітурієнта та дозволяє подавати заявки на обрані спеціальності в залежності від їх результатів НМТ.

3.1.5 Дозволи користувачів

На сайті з обліку абітурієнтів кафедри можна реалізувати різні рівні дозволів для користувачів, такі як Адміністратор і Користувач. Ось як можна розглянути ці ролі:

Адміністратор: Цей користувач має найвищий рівень доступу і може мати доступ до всієї інформації та функціоналу на сайті. Деякі можливі можливості для адміністратора:

Перегляд, редагування та видалення будь-яких даних або записів.

Управління ролями та доступом інших користувачів на сайті.

Додавання нових користувачів або видалення існуючих.

Модерація контенту на сайті, така як затвердження коментарів або відгуків.

Користувач: Цей рівень доступу призначений для звичайних користувачів, які мають обмежений доступ до функціоналу сайту. Деякі можливі можливості для користувача:

Перегляд власних даних та редагування їх.

Подача заявок на обрані спеціальності.

Перегляд статусу поданих заявок та іншої власної інформації.

Реалізація цих рівнів доступу може здійснюватися за допомогою системи авторизації та аутентифікації, наприклад, використовуючи ролі ASP.NET або інші механізми контролю доступу веб-сайту.

3.1.6 Форма зворотнього зв’язку

Форма зворотнього зв'язку на сайті з обліку абітурієнтів кафедри може бути використана для того, щоб користувачі могли звертатися до адміністраторів або підтримки з питань, які виникають під час використання сайту. Ось деякі особливості, які можуть бути включені до такої форми:

Поле для введення повідомлення: Користувач може ввести своє повідомлення, де він може описати свою проблему або запитання.

Поля для контактних даних: Користувач може ввести свою електронну адресу або номер телефону, якщо він очікує відповідь на своє повідомлення.

Кнопка "Надіслати": Після заповнення форми користувач натискає кнопку "Надіслати", щоб надіслати своє повідомлення адміністраторам або підтримці сайту.

Повідомлення про успішну відправку: Після надсилання форми користувач може отримати повідомлення про успішну відправку, щоб підтвердити, що його повідомлення було отримано.

Повідомлення про помилку: Якщо користувач введе невірні дані або не заповнить обов'язкові поля, він може отримати повідомлення про помилку та вказівки щодо виправлення.

Ця форма дозволяє користувачам легко звертатися до адміністраторів або підтримки сайту та отримувати відповіді на свої питання або проблеми.

Код реалізації наведений нижче.

```
@{
ViewData["Title"] = "Підтримка";
}

<div class="container">
<h2>Підтримка</h2>
@if (ViewBag.Message != null)
{
<div class="alert alert-info">
@ViewBag.Message
</div>
}
<form method="post" action="/Support/SendSupportRequest">
```

```
<div class="form-group">
<label for="name">Ім'я</label>
<input type="text" class="form-control" id="name" name="name" required>
</div>
<div class="form-group">
<label for="email">Електронна пошта</label>
<input type="email" class="form-control" id="email" name="email" required>
</div>
<div class="form-group">
<label for="message">Повідомлення</label>
<textarea class="form-control" id="message" name="message" rows="5" required></textarea>
</div>
<button type="submit" class="btn btn-primary">Надіслати</button>
</form>
</div>
```

Цей код створює сторінку підтримки на веб-сайті, яка дозволяє користувачам зв'язатися з адміністрацією. Він починається з встановлення заголовка сторінки як "Підтримка". Весь вміст форми розміщено всередині контейнера, що допомагає структуризувати та стилізувати сторінку. На сторінці є заголовок "Підтримка", який інформує користувачів про призначення цієї сторінки. Якщо у ViewBag є повідомлення, воно відображається як інформаційне повідомлення, яке може використовуватися для інформування користувача про успішне надсилання форми або інші важливі повідомлення.

Форма зворотного зв'язку відправляє дані методом POST на маршрут /Support/SendSupportRequest, де вони будуть оброблені. Вона містить кілька обов'язкових полів для введення: поле для введення імені, де користувач вводить своє ім'я, поле для електронної пошти, яке забезпечує валідацію введених даних і текстове поле для введення повідомлення, що дозволяє користувачеві написати своє повідомлення. Наприкінці форми є кнопка, натиснувши на яку, користувач відправляє свої дані для обробки адміністрацією сайту.

3.2 Розробка бази даних

Розробка бази даних для веб-сервісу обліку абітурієнтів кафедри є ключовим етапом у створенні системи, яка забезпечить зберігання, організацію та доступ до всієї необхідної інформації про абітурієнтів. База даних повинна бути спроектована

таким чином, щоб забезпечити ефективне зберігання даних, швидкий доступ до них та можливість обробки великих обсягів інформації.

Перше, що потрібно зробити при розробці бази даних, це визначити всі необхідні сутності (таблиці) та їх взаємозв'язки. Основними сутностями є "Абітурієнти", "Користувачі", "Заявки", "Спеціальності", "Адміністратори" та інші.

Таблиця "Абітурієнти" містить основну інформацію про кожного абітурієнта, таку як ім'я, прізвище, електронна пошта, номер телефону, результати НМТ та інші дані, що можуть бути необхідними для подачі заявки. Важливо, щоб кожен запис у цій таблиці мав унікальний ідентифікатор (ID), який дозволяє однозначно ідентифікувати кожного абітурієнта.

Таблиця "Користувачі" містить інформацію про всіх користувачів системи, включаючи як абітурієнтів, так і адміністраторів. У цій таблиці можуть бути поля для зберігання імені користувача, паролю, ролі (адміністратор чи абітурієнт) та інших даних, необхідних для аутентифікації та авторизації користувачів.

Таблиця "Заявки" містить інформацію про всі заявки, подані абітурієнтами. Вона може включати посилання на таблицю "Абітурієнти" для визначення, хто подав заявку, та на таблицю "Спеціальності" для визначення, на яку спеціальність подана заявка.

Таблиця "Спеціальності" містить перелік усіх доступних спеціальностей, на які можуть подавати заявки абітурієнти. Вона містить такі поля, як назва спеціальності, опис, вимоги та інші дані, що можуть бути корисними для абітурієнтів.

Таблиця "Адміністратори" містить інформацію про всіх адміністраторів системи. Ця таблиця може бути пов'язана з таблицею "Користувачі", але може містити додаткові поля, специфічні для адміністраторів, такі як права доступу або додаткові контактні дані.

Розробка бази даних включає також визначення взаємозв'язків між таблицями. Наприклад, таблиця "Заявки" має зовнішні ключі, що посилаються на таблиці "Абітурієнти" та "Спеціальності". Це дозволяє забезпечити цілісність даних та уникнути дублювання інформації.

Після визначення структури бази даних необхідно створити SQL-запити для створення таблиць та встановлення взаємозв'язків між ними. Крім того, потрібно визначити індекси для поліпшення швидкості запитів, які часто використовуються, а також розробити процедури для резервного копіювання та відновлення бази даних.

У результаті, добре спроектована база даних забезпечить надійне зберігання даних, швидкий доступ до них та можливість ефективного управління інформацією про абітурієнтів, що є критично важливим для успішної роботи веб-сервісу обліку абітурієнтів кафедри.

3.3 Розробка та функції моделей

Моделі (Models) є центральним поняттям у веб-розробці та об'єктно-орієнтованому програмуванні. Модель – це абстракція, що представляє структуру даних для певного об'єкта або сутності у вашому додатку чи системі. Модель визначає властивості (атрибути) та поведінку (методи) певного об'єкта. Вона інкапсулює логіку та дані, пов'язані з конкретною концепцією або сутністю вашого додатку.

Основні функції моделей:

1. Структурування даних - моделі визначають, які дані повинні зберігатися та як вони пов'язані між собою.
2. Інкапсуляція логіки - моделі містять правила та логіку, пов'язані з даними, які вони інкапсулюють.
3. Перевірка валідності - моделі можуть визначати правила валідації для забезпечення коректності та цілісності даних.
4. Взаємодія з базою даних - моделі часто використовуються для доступу до даних у базах даних, запису, оновлення та видалення записів.
5. Незалежність від представлення - моделі відокремлені від користувацьких інтерфейсів та інших компонентів відображення.

Використання моделей забезпечує кращу організацію коду, дотримання принципу єдиної відповідальності та полегшує тестування і підтримку коду. У багатьох сучасних веб-фреймворках і шаблонах проектування використання моделей є невід'ємною частиною розробки.

У моєму веб-додатку для обліку абітурієнтів кафедри я використовую кілька основних моделей:

1. Applicant (Абітурієнт)Ця модель зберігає ключову інформацію про абітурієнта - ім'я, прізвище, дату народження, контактні дані тощо. Вона є центральною сутністю системи, оскільки всі заяви та результати іспитів пов'язані з конкретним абітурієнтом.

2. ErrorViewModel Це допоміжна модель для передачі даних про помилки та виключні ситуації, які можуть виникати в додатку. Вона допомагає структурувати та транспортувати інформацію про помилки на сторінки із зрозумілим для користувача форматуванням.

3. LoginViewModel Ця модель використовується на сторінці входу в систему. Вона містить поля для введення облікових даних користувача (логіна та пароля) та може мати додаткові властивості, пов'язані з авторизацією, наприклад, прапорець "Запам'ятати мене".

4. RegisterViewModel Модель реєстрації нового абітурієнта в системі. Тут зберігаються дані, необхідні для створення нового облікового запису - логін, пароль, ім'я, контактна інформація тощо. Після успішної реєстрації ці дані переносяться в модель Applicant.

5. SupportFormModelЦя модель призначена для форми зворотного зв'язку або підтримки користувачів. У ній можуть міститися поля для імені відвідувача, його електронної пошти, теми звернення та повідомлення. Ці дані використовуються для обробки запитів від абітурієнтів.

Усі ці моделі відіграють важливу роль у моєму веб-додатку. Вони інкапсулюють відповідні дані, забезпечують валідацію, визначають взаємозв'язки між сутностями та спрощують роботу з даними в різних частинах додатку.

3.4 Розробка контролерів

Контролер - це частина програми, яка керує роботою додатку і взаємодією між його компонентами.

Контролер діє як проміжна ланка між користувачем, даними і візуальним представленням додатку. Він обробляє запити користувача, визначає, що потрібно зробити, звертається до відповідних моделей даних, отримує потрібні дані, передає їх у відповідне представлення для відображення користувачеві.

Контролери використовуються в багатьох видах програм і фреймворків, особливо у веб-розробці. Популярні веб-фреймворки, такі як Ruby on Rails, Laravel, ASP.NET MVC, Angular використовують архітектурний шаблон Model-View-Controller (MVC), де контролер відіграє ключову роль у керуванні потоком даних і логікою додатку.

Таким чином, контролер є важливим компонентом, який забезпечує правильну координацію роботи додатку, розподіляє обов'язки між різними частинами програми та керує їх взаємодією.

3.4.1 Контролер AccountController

Цей контролер використовується для керування системою автентифікації та обліку користувачів у веб-застосунку. Він забезпечує функції реєстрації нових користувачів, входу в систему, виходу з системи та керування сесіями користувачів.

Сам код цього контроллера:

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;
using AdmissionSystem.ViewModels;

namespace AdmissionSystem.Controllers
{
public class AccountController : Controller
{
private readonly UserManager<IdentityUser> _userManager;
private readonly SignInManager<IdentityUser> _signInManager;
```

```
public AccountController(UserManager<IdentityUser> userManager, SignInManager<IdentityUser>
signInManager)
{
_userManager = userManager;
_signInManager = signInManager;
}

[HttpGet]
public IActionResult Register()
{
return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Register(RegisterViewModel model)
{
if (ModelState.IsValid)
{
var user = new IdentityUser { UserName = model.Email, Email = model.Email };
var result = await _userManager.CreateAsync(user, model.Password);
if (result.Succeeded)
{
await _signInManager.SignInAsync(user, isPersistent: false);
return RedirectToAction("Index", "Home");
}
foreach (var error in result.Errors)
{
ModelState.AddModelError(string.Empty, error.Description);
}
}
return View(model);
}

[HttpGet]
public IActionResult Login()
{
return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null)
{
ViewData["ReturnUrl"] = returnUrl;
if (ModelState.IsValid)
{
var result = await _signInManager.PasswordSignInAsync(model.Email, model.Password,
model.RememberMe, lockoutOnFailure: false);
if (result.Succeeded)
{
return RedirectToLocal(returnUrl);
```

```
}
if (result.IsLockedOut)
{
return View("Lockout");
}
else
{
ModelState.AddModelError(string.Empty, "Invalid login attempt.");
return View(model);
}
}
return View(model);
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Logout()
{
await _signInManager.SignOutAsync();
return RedirectToAction("Index", "Home");
}

private IActionResult RedirectToLocal(string returnUrl)
{
if (Url.IsLocalUrl(returnUrl))
{
return Redirect(returnUrl);
}
else
{
return RedirectToAction(nameof(HomeController.Index), "Home");
}
}
}
}
```

Тепер детально про те, що робить цей код:

Контролер отримує екземпляри UserManager і SignInManager через конструктор. Ці об'єкти відповідають за операції з користувачами та їх автентифікацію.

Метод Register() обробляє запит на реєстрацію нового користувача. При GET-запиті він просто повертає представлення з формою реєстрації. При POST-запиті він отримує дані з форми реєстрації (email і пароль) та намагається створити нового користувача в системі. Якщо створення пройшло успішно, користувач автоматично входить в систему і перенаправляється на головну сторінку.

Метод Login() обробляє запит на вхід користувача в систему. При GET-запиті він повертає представлення з формою входу. При POST-запиті він отримує email, пароль та прапорець "Запам'ятати мене" і намагається авторизувати користувача. Якщо авторизація пройшла успішно, користувач перенаправляється на вказану URL-адресу або на головну сторінку.

Метод Logout() обробляє запит на вихід користувача з системи. Він викликає SignOutAsync() для завершення сесії користувача і перенаправляє його на головну сторінку.

Метод RedirectToLocal() використовується для безпечного перенаправлення користувача на вказану URL-адресу після успішного входу в систему. Він перевіряє, чи є ця URL локальною для застосунку, і якщо так, перенаправляє користувача на неї, інакше - на головну сторінку.

3.4.2 Контролер ApplicantsController

Контролер ApplicantsController використовується для керування даними про абітурієнтів (Applicants) у веб-додатку для обліку вступників.

Він забезпечує наступні основні функції:

Отримання списку всіх абітурієнтів (метод Index).

Створення нового абітурієнта (метод Create).

Редагування існуючого абітурієнта (методи Edit).

Видалення абітурієнта (методи Delete та DeleteConfirmed).

Перегляд детальної інформації про абітурієнта (метод Details).

Контролер працює з контекстом бази даних ApplicationDbContext, який надає доступ до сутності Applicant та забезпечує операції зчитування, створення, оновлення та видалення даних абітурієнтів.

Методи контролера відповідають за обробку HTTP-запитів (GET та POST), валідацію вхідних даних, взаємодію з базою даних через контекст ApplicationDbContext, а також перенаправлення та відображення відповідних представлень (View) для абітурієнтів.

Загалом, цей ApplicantsController є центральною точкою для керування життєвим циклом даних абітурієнтів у веб-додатку, забезпечуючи функціональність CRUD (створення, зчитування, оновлення та видалення) для цієї сутності.

3.4.3 Контролер HomeController

Контролер HomeController відповідає за обробку основних маршрутів та сторінок у веб-додатку AdmissionSystem.

Його основні функції наступні:

Метод Index() обробляє запити на головну сторінку веб-додатку. При запиті на корінь веб-сайту або маршрут /Home, він просто повертає представлення Index.cshtml.

Метод Privacy() обробляє маршрут /Home/Privacy і повертає представлення Privacy.cshtml. Зазвичай ця сторінка містить інформацію про конфіденційність та політику захисту даних веб-додатку.

Метод Error() використовується для обробки виключних ситуацій та помилок у додатку. Він створює екземпляр моделі ErrorViewModel, встановлює ідентифікатор поточного запиту або запис про трасування і повертає представлення Error.cshtml з переданою моделлю.

Цей контролер не містить складної бізнес-логіки, а лише виконує роль маршрутизатора для основних сторінок та сторінки обробки помилок.

Контролер отримує екземпляр ILogger<HomeController> через конструктор залежностей, що дозволяє йому реєструвати повідомлення в журналі подій під час виконання.

В цілому, HomeController відповідає за початкову точку входу в додаток, сторінку конфіденційності та обробку помилок, в той час як більш специфічна функціональність реалізована в інших контролерах, як от ApplicantsController.

3.4.4 Контролер SupportController

Цей контролер в ASP.NET Core MVC відповідає за обробку запитів до сторінки підтримки на нашому веб-сервісі обліку абітурієнтів кафедри. Він містить

два методи: Index та SendSupportRequest. Метод Index обробляє GET-запити до сторінки підтримки.

Коли користувач переходить на сторінку підтримки, цей метод повертає відповідне подання (view), яке відображає форму для заповнення даних підтримки. Метод SendSupportRequest обробляє POST-запити, які надходять при відправленні форми підтримки. Цей метод приймає три параметри: name, email, message, які користувач вводить у форму.

4. УДОСКОНАЛЕННЯ ТА МОЖЛИВОСТІ РОЗВИТКУ ВЕБ-СЕРВІСУ

Удосконалення та постійний розвиток веб-сервісів є критично важливим для забезпечення їх актуальності, ефективності та конкурентоспроможності. Ось чому вдосконалення необхідне і які переваги та недоліки воно може мати:

Чому треба удосконалювати веб-сервіси:

1. Відповідність сучасним трендам і технологіям. Технології швидко розвиваються, з'являються нові фреймворки, мови програмування, інструменти та підходи. Регулярне вдосконалення дозволяє веб-сервісам залишатися сучасними і використовувати найновіші досягнення.

2. Покращення продуктивності та масштабованості. Оновлення архітектури, оптимізація коду, впровадження кешування та інших технік можуть значно підвищити швидкодію та здатність сервісу обробляти більше навантаження.

3. Розширення функціоналу. Додавання нових можливостей, інтеграція з іншими системами, підтримка нових пристроїв - все це потребує вдосконалення веб-сервісу.

4. Підвищення безпеки. Регулярні оновлення забезпечують виправлення виявлених вразливостей, посилення захисту від атак і дотримання найновіших стандартів безпеки.

5. Покращення користувацького досвіду. Модернізація інтерфейсу, поліпшення навігації, адаптивний дизайн - все це робить веб-сервіс більш зручним і привабливим для користувачів. Звичайно, розглянемо більш детально переваги та недоліки вдосконалення веб-сервісів.

Переваги:

1. Покращення продуктивності та швидкодії. Оновлення технологій, оптимізація коду, застосування кешування, використання більш ефективних алгоритмів і структур даних може значно підвищити швидкість обробки запитів і відгуку веб-сервісу. Це позитивно впливає на задоволеність користувачів.

2. Розширення функціональних можливостей. Додавання нових функцій, інтеграція з іншими системами та сервісами розширює спектр послуг, що надаються веб-сервісом. Це робить його більш привабливим для клієнтів та підвищує конкурентоспроможність.

3. Покращення користувацького досвіду. Модернізація інтерфейсу, спрощення навігації, адаптивний дизайн під різні пристрої та роздільні здатності підвищують зручність використання веб-сервісу та задоволеність користувачів.

4. Підвищення безпеки. Регулярні оновлення забезпечують виправлення критичних вразливостей, посилення захисту від атак і дотримання найновіших стандартів безпеки. Це захищає дані користувачів і репутацію веб-сервісу.

5. Відповідність сучасним трендам і технологіям. Використання нових технологій, фреймворків та мов програмування дозволяє веб-сервісу залишатися актуальним, легше інтегруватися з іншими системами та залучати кваліфікованих розробників.

6. Можливість масштабування. Вдосконалення архітектури, переходу на більш масштабовані технології та розподілені системи надає змогу веб-сервісу обробляти більші обсяги трафіку та даних у міру зростання бізнесу.

Недоліки:

1. Витрати ресурсів. Вдосконалення веб-сервісу вимагає значних зусиль і витрат на розробку, тестування, розгортання та навчання персоналу. Це може бути дорого для бізнесу.

2. Ризики виникнення помилок та несумісностей. При внесенні змін до існуючої кодової бази завжди є ризик появи нових помилок або порушень сумісності з існуючими системами чи додатками.

3. Потреба в ретельному тестуванні. Будь-які оновлення повинні ретельно тестуватися на різних рівнях, щоб гарантувати коректну роботу веб-сервісу та уникнути збоїв чи втрати даних.

4. Перехідний період для користувачів. Значні зміни в інтерфейсі чи логіці можуть вимагати від користувачів певного періоду адаптації та навчання.

5. Необхідність технічної підтримки старих версій. Під час поступового переходу на нову версію веб-сервісу може знадобитися підтримка і обслуговування старих версій, що збільшує навантаження на розробників.

6. Потенційна несумісність з існуючими системами. У разі радикальних змін в архітектурі або використовуваних технологіях, може виникнути проблема інтеграції з іншими системами, що вже використовуються в бізнесі.

Незважаючи на деякі ризики та недоліки, регулярне вдосконалення веб-сервісів є необхідним для забезпечення їх актуальності, безпеки та конкурентоспроможності на ринку. Ретельне планування, керування ризиками, поступове впровадження змін та належне тестування допоможуть максимізувати переваги та мінімізувати проблеми під час удосконалення веб-сервісів.

4.1 Пропозиції щодо покращення функціональності веб-сервісу

Розглянемо детально потенційні можливості для покращення функціональності та ефективності вашого веб-сервісу з обліку абітурієнтів кафедри:

1. Інтеграція з системами зовнішнього незалежного оцінювання (ЗНО):

- Автоматичний імпорт результатів ЗНО дозволить заощадити час співробітників на ручному введенні балів кожного абітурієнта.

- Мінімізує ризики людських помилок при перенесенні даних.- Актуальні та точні дані про результати ЗНО важливі для коректного розрахунку конкурсних балів абітурієнтів.

- Підвищить загальну ефективність та швидкість обробки заяв під час вступної кампанії.

2. Розширені аналітичні можливості та генератор звітів:

- Аналітика допоможе керівництву кафедри відстежувати ключові показники вступної кампанії в режимі реального часу.

- Візуалізація даних у вигляді графіків, діаграм полегшить сприйняття інформації та виявлення трендів.

- Гнучкий генератор звітів дозволить формувати звіти за потрібними параметрами для різних цілей (наради, планування, акредитація тощо).

- Забезпечить можливість приймати більш обґрунтовані рішення на основі аналізу даних.

3. Система сповіщень та комунікації з абітурієнтами - це комплексне рішення, яке дозволяє веб-сервісу обліку абітурієнтів кафедри ефективно інформувати вступників та взаємодіяти з ними.

Ця система може включати кілька взаємопов'язаних компонентів:

Модуль сповіщень

Це засіб для надсилання важливих повідомлень абітурієнтам різними каналами: електронна пошта, SMS, push-сповіщення в мобільному додатку тощо. Сповіщення можуть інформувати про ключові дати подачі документів, графік іспитів, терміни зарахування тощо. Також можна налаштувати автоматичне розсилання персоналізованих сповіщень у певні періоди часу.

Інтеграція з месенджерами/чат-ботами

Підключення веб-сервісу до популярних месенджерів (Telegram, WhatsApp, Facebook Messenger та ін.) Надання можливості абітурієнтам задавати питання, отримувати відповіді в зручному форматі діалогу. Використання чат-ботів для автоматизації обробки типових запитів і розвантаження адміністративного персоналу.

Панель адміністрування системи комунікації

Інструмент для керування розсилками сповіщень, редагування їх контенту та налаштування цільових груп і графіків розсилки. Моніторинг ефективності сповіщень та відстеження зворотного зв'язку від абітурієнтів. Налаштування бізнес-правил для автоматизації розсилки сповіщень у певних ситуаціях.

Переваги такої системи:

Своєчасне інформування абітурієнтів про важливі події, уникнення випадків пропущених термінів/подій. Можливість персоналізованої комунікації та надання релевантних підказок кожному абітурієнту. Зручний канал для оперативного вирішення питань та запитів. Автоматизація рутинних процесів інформування та

підтримки абітурієнтів. Підвищення залученості абітурієнтів та загальної користувацької задоволеності.

Впровадження ефективної системи сповіщень та комунікацій допоможе нашому веб-сервісу.

Особистий кабінет абітурієнта:

- Надасть абітурієнтам зручний доступ до своїх персональних даних та статусу заяв у режимі реального часу.

- Дозволить вносити зміни до персональної інформації чи документів без звернення до адміністрації.

- Підвищить залученість абітурієнтів та прозорість процесу вступу.

- Розвантажить адміністративний персонал від необхідності надавати таку інформацію вручну.

5. Мобільний додаток:

- Забезпечить абітурієнтам мобільний доступ до сервісу з будь-якого місця і пристрою.

- Push-сповіщення гарантуватимуть своєчасне інформування.

- Покращить загальний користувацький досвід та зручність взаємодії з сервісом.

- Дозволить бути на зв'язку з абітурієнтами в режимі 24/7. 6. Покращення безпеки та захисту даних:

- Двофакторна автентифікація посилить захист облікових записів від несанкціонованого доступу.

- Шифрування конфіденційних даних захистить їх від витоку чи компрометації.

- Своєчасні оновлення усунуть відомі вразливості та загрози безпеці.

- Підвищить довіру користувачів та репутацію сервісу.

7. Оптимізація продуктивності та масштабованості:

- Ефективне кешування зменшить навантаження на базу даних та прискорить обробку запитів.

- Переведення на масштабовану архітектуру дозволить без проблем справлятися з високими навантаженнями під час пікових періодів.

- Забезпечить плавну роботу сервісу навіть зі зростанням кількості користувачів та заяв.

8. Інтеграція з навчальними системами університету:

- Автоматичне перенесення даних зарахованих абітурієнтів спростить їх реєстрацію як студентів.

- Уникнення дублювання даних та необхідності ручного введення.

- Прискорить і автоматизує обробку вхідного потоку студентів.

9. Система керування навчальним контентом:

- Інтеграція з системами дистанційного навчання надасть абітурієнтам доступ до підготовчих матеріалів.

- Вони зможуть самостійно готуватися в зручному форматі відео, текстів, тестів тощо.

- Допоможе залучити більше потенційних кандидатів на навчання шляхом надання якісних матеріалів.

Ці вдосконалення комплексно покращать функціональність, ефективність, безпеку, аналітику та загальний користувацький досвід вашого веб-сервісу. Їх впровадження зробить його більш сучасним, конкурентоспроможним та цінним як для абітурієнтів, так і для адміністрації кафедри.

4.2 Аналіз питань безпеки даних

Безпека даних є критично важливим аспектом для веб-сервісу обліку абітурієнтів кафедри, оскільки він оперує конфіденційною особистою інформацією абітурієнтів.

Розглянемо детально ключові питання безпеки даних та можливі рішення для їх вирішення:

1. **Захист персональних даних абітурієнтів** - Впровадження шифрування даних, що містять персональну інформацію (імена, контактні дані, результати

іспитів тощо) як під час передачі, так і в стані спокою в базі даних. - Використання алгоритмів захищеного хешування та солі для захисту паролів користувачів. - Обмеження доступу до персональних даних лише для авторизованого персоналу з відповідними правами.

2. **Безпека облікових записів** - Двофакторна автентифікація для підвищення захисту облікових записів абітурієнтів і адміністраторів. - Перевірка міцності паролів та обов'язкова зміна пароля після першого входу в систему. - Аудит та моніторинг спроб входу в систему для виявлення потенційних атак. - Обмеження кількості невдалих спроб входу та блокування облікових записів після перевищення ліміту.

3. **Захист від вебвразливостей** - Впровадження заходів захисту від поширених вебвразливостей, таких як впровадження коду, міжсайтовий скриптинг (XSS), підробка міжсайтових запитів (CSRF). - Належна валідація, очищення та контекстне кодування всіх вхідних даних. - Використання механізмів захисту від витоку інформації, таких як коректні заголовки безпеки (HttpOnly, X-XSS-Protection). - Регулярне оновлення всіх бібліотек та фреймворків для усунення відомих вразливостей.

4. **Безпека передачі даних** - Використання протоколу HTTPS для захищеного з'єднання та передачі даних між клієнтом та сервером. - Налаштування та примусове використання захищених налаштувань протоколу TLS з відключенням старих та вразливих версій. - Впровадження захисту від атак типу "людина посередині" (man-in-the-middle).

5. **Контроль доступу та розмежування обов'язків** - Реалізація ролевої моделі доступу з ретельно продуманими правами для різних категорій користувачів (абітурієнти, адміністратори, персонал). - Розмежування критичних обов'язків між адміністраторами для запобігання зловживань. - Реалізація принципу мінімальних привілеїв з використанням "найменшого спільного привілею".

6. **Аудит та моніторинг** - Логування дій користувачів, спроб входу, змін даних для відслідковування та розслідування інцидентів. - Впровадження систем

виявлення вторгнень (IDS/IPS) та механізмів моніторингу безпеки. - Періодичне проведення аудиту безпеки та тестування на проникнення незалежними експертами.

7. **Планування реагування на інциденти** - Розробка плану реагування на інциденти інформаційної безпеки з чіткими інструкціями та протоколами дій. - Визначення відповідальних осіб та формування команди реагування на інциденти. - Регулярне проведення навчань та практичних тренувань.

8. **Безперервне вдосконалення безпеки** - Відстеження новин про вразливості та загрози безпеки, пов'язані з використовуваними технологіями. - Регулярний перегляд та оновлення політик безпеки відповідно до найкращих практик галузі. - Залучення експертів з інформаційної безпеки для консультацій та оцінки ризиків.

Впровадження цих заходів допоможе створити комплексну систему безпеки даних для вашого веб-сервісу обліку абітурієнтів, захистити конфіденційну інформацію, підвищити довіру користувачів та відповідати вимогам законодавства щодо захисту персональних даних.

4.3 Розширення галузей використання

Розширення галузей використання веб-сервісу для обліку абітурієнтів кафедри може відкрити нові можливості та перспективи для збільшення його функціональності та охоплення користувачів. Розглянемо потенційні напрямки розширення його використання:

1. Розширення на рівень факультету/університету:

- Адаптація веб-сервісу для обліку абітурієнтів не лише на рівні кафедри, а й на рівні факультету чи всього університету.

- Це дозволить централізовано керувати всім процесом прийому абітурієнтів в межах навчального закладу.

2. Інтеграція з системою електронного документообігу:

- Підключення веб-сервісу до системи електронного документообігу університету.

- Дозволить автоматизувати обмін даними та документами між відділом кадрів, приймальною комісією та структурними підрозділами.

3. Підтримка різних форм навчання:

- Розширення функціональності для обліку абітурієнтів на різні форми навчання: денна, заочна, дистанційна, друга освіта тощо.

- Уніфікований підхід до обробки заяв від різних категорій вступників.

4. Інтеграція з системами електронної пошти та повідомлень:

- Підключення сервісу розсилки повідомлень для інформування абітурієнтів через електронну пошту, SMS, месенджери тощо.

- Своєчасне інформування про важливі деталі, події, оповіщення про пройдені етапи тощо.

5. Впровадження мобільного додатку:

- Розробка мобільного додатку для абітурієнтів, який би інтегрувався з веб-сервісом та надавав потрібну інформацію в зручному форматі.

- Можливість відстежувати статус своєї заяви, отримувати сповіщення тощо прямо зі смартфону.

6. Розширення функціональності для міжнародних студентів:

- Додавання можливостей для обробки заяв від іноземних студентів, з урахуванням специфічних вимог та особливостей.

- Підтримка різних мов інтерфейсу, можливість завантаження відповідних документів тощо.

7. Інтеграція з системами керування навчальними процесами:

- Інтеграція з системами, такими як електронні журнали, портали для студентів тощо.

- Дозволить забезпечити безперервність і зручність переходу від процесу вступу до навчання.

8. Додавання функцій для супроводу іноземних студентів:

- Інформування щодо візових питань, перекладачів, гуртожитків тощо.

- Корисно для залучення більшої кількості іноземних студентів.

Таким чином, розширюючи галузі використання веб-сервісу та адаптуючи його до різних потреб, можна значно підвищити його цінність, функціональність та привабливість як для абітурієнтів, так і для навчального закладу в цілому.

ВИСНОВОК

1. Проведено всебічний аналіз галузі управління абітурієнтами, що дозволило ідентифікувати основні потреби та вимоги сучасних систем обліку.

В ході аналізу виявлені ключові проблеми, з якими стикаються навчальні заклади під час прийому нових студентів.

Основними проблемами є:

Необхідність ефективного управління великими обсягами інформації про абітурієнтів.

Забезпечення зручності та простоти для користувачів при подачі заявок та відстеженні статусу їх розгляду.

Надання можливості адміністрування та обробки даних у режимі реального часу.

Гарантія безпеки та конфіденційності даних абітурієнтів.

Аналіз показав, що для успішного управління абітурієнтами необхідно створити систему, яка буде відповідати цим потребам та вирішувати виявлені проблеми.

2. Досліджено існуючі рішення для управління абітурієнтами, що дозволило визначити їх переваги та недоліки. Це дало змогу виділити найкращі практики та уникнути недоліків у власному проекті.

3. На основі аналізу галузі та готових рішень сформовано вимоги до програмного забезпечення для системи обліку абітурієнтів. Вимоги включають функціональні та нефункціональні аспекти, що забезпечують ефективність, надійність та зручність використання системи.

4. Проведено детальний аналіз різних підходів до розробки веб-сайтів з використанням ASP.NET та C#.

Було розглянуто різні бібліотеки, бази даних та інші компоненти, що забезпечують оптимальну продуктивність та масштабованість системи.

Обрано наступні інструменти:

ASP.NET: як основна платформа для розробки веб-додатків, завдяки її високій продуктивності, надійності та підтримці різних функціональних можливостей.

C#: як основна мова програмування, що забезпечує високу продуктивність та можливість створення надійного коду.

SQL Server: як система управління базами даних, що забезпечує ефективне зберігання та обробку великої кількості даних.

JavaScript: для реалізації інтерактивних елементів на веб-сторінках.

Bootstrap: для створення адаптивного та сучасного дизайну інтерфейсу користувача.

5.Робота пройшла апробацію: Романський А.В., Гребенюк В.В. Визначення засобів реалізації Web-платформи для обліку абітурієнтів кафедри. Всеукраїнсько науково-технічна конференція “Сучасні інтелектуальні інформаційні технології в науці та освіті”, 15 травня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій Збірник тез. К.: ДУКІТ С. 194 – 198.

ПЕРЕЛІК ПОСИЛАНЬ

1. ASP.NET Core Documentation [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core.

2. Entity Framework Core Documentation [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/ef/core/.

3. Blazor: Interactive web UI with .NET [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-7.0.

4. Razor Pages in ASP.NET Core [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-7.0&tabs=visual-studio.

5. Identity on ASP.NET Core [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-7.0&tabs=visual-studio.

6. Introducing SQL Server 2022 [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://www.microsoft.com/en-us/sql-server.

7. C# Documentation [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/csharp/.

8. Mastering Entity Framework Core: Learn to leverage the full potential of the Entity Framework Core ORM [Електронний ресурс] / 2021. – Режим доступу до ресурсу: https://www.oreilly.com/library/view/mastering-entity-framework/9781788294137/.

9. Pro ASP.NET Core MVC 2: Develop Cloud-Ready Web Applications Using MVC, C#, and EF Core / Adam Freeman. – Apress, 2017. – 1015 с.

10. Building Web Applications with Visual Studio 2022: Build modern and scalable full stack applications using the power of .NET 6, Blazor, and Visual Studio 2022 / Daniel Baharestani. – Packt Publishing, 2022. – 652 с.

11. The Complete ASP.NET Core 3 API Tutorial: Practical ASP.NET Core REST API Design / Les Jackson. – Leanpub, 2021. – 442 с.

12. Clean Architecture: A Craftsman's Guide to Software Structure and Design / Robert C. Martin. – Prentice Hall, 2017. – 432 с.

13. SQL Performance Explained: Learn to Develop Fast and Maintainable Database Applications / Markus Winand. – Winand Publishing, 2012. – 204 с.

14. Hands-On RESTful Web Services with ASP.NET Core 3: Develop, deploy, and debug robust and secure web APIs with C# 8 and .NET Core 3.0 / Samuele Resca. – Packt Publishing, 2019. – 592 с.

15 ."Professional ASP.NET MVC 5" / Jon Galloway, Brad Wilson, K. Scott Allen, David Matson. – Wrox, 2014. – 624 с.

16. "Programming Microsoft ASP.NET MVC" / Dino Esposito. – Microsoft Press, 2014. – 782 с.

17. "Modern Web Development with ASP.NET Core 3: An end to end guide covering the latest features of Visual Studio 2019, Blazor and Entity Framework, 2nd Edition" / Ricardo Peres. – Packt Publishing, 2019. – 618 с.

18."ASP.NET Core in Action" / Andrew Lock. – Manning Publications, 2018. – 775 с.

19. "Learning SQL: Master SQL Fundamentals" / Alan Beaulieu. – O'Reilly Media, 2020. – 384 с.

ДОДАТОК А. ДЕМОСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка Web-сервісу для обліку абітурієнтів кафедри мовою C#

Виконав студент 4 курсу

групи ПД-41

Романський Андрій Васильович

Керівник роботи

Доктор філософії, доцент кафедри ІПЗ Гребенюк Віктор Вікторович

Київ – 2024

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести аналіз галузі управління абітурієнтами.
2. Провести аналіз готових рішень для галузі управління абітурієнтами.
3. Розробка та формування вимог до програмного забезпечення для системи обліку абітурієнтів кафедри.
4. Провести аналіз підходів до розробки веб-сайтів з використанням ASP.NET, C#, обрати бібліотеки, базу даних та інші компоненти для реалізації веб-додатку.
5. Спроектувати веб-сервіс для системи обліку абітурієнтів кафедри.
6. Розробити веб-сервіс для системи обліку абітурієнтів.
7. Провести тестування програмного забезпечення.

АНАЛІЗ АНАЛОГІВ

	ApplyBroad	StudyLink	EducationUSA	Облік Абітурієнтів
Платформа	Web, Mobile	Web	Web, Mobile	Web
CRUD-операції з інформацією про абітурієнтів	-	+	+	+
Пошук по даним абітурієнтів	+	+	-	+
Форма зворотнього зв'язку	+	+	+	+
Реєстрація користувача	+	-	+	+

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги користувача:

1. Реєстрація користувача у системі для перегляду свої інформації.
2. Авторизація користувача у системі.
3. Можливість додавання нових абітурієнтів.
4. Редагування інформації про абітурієнтів.
5. Подання заявок на спеціальності.

Нефункціональні вимоги користувача:

1. Безпека даних наданих користувачем.
2. Обмеження відповідно до ролі користувача.
3. Продуктивність.

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги адміністратора:

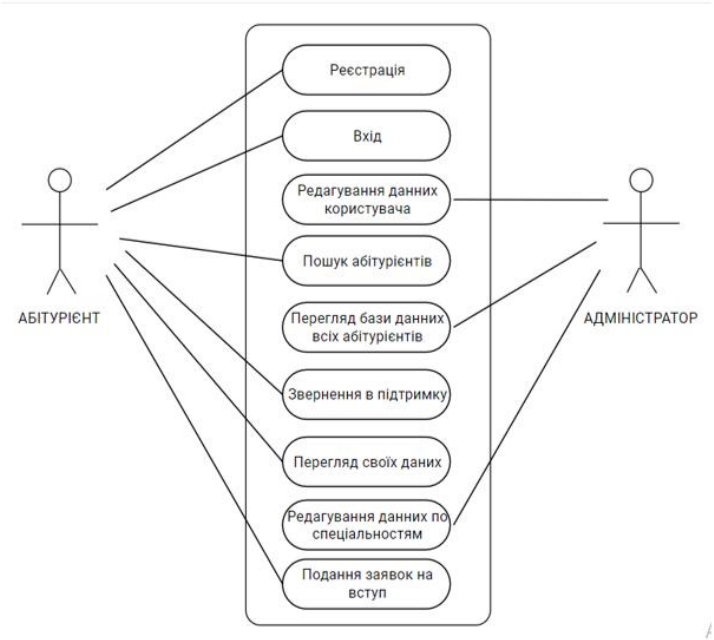
1. Авторизація користувача у системі.
2. Можливість додавання нових абітурієнтів.
3. Редагування інформації про абітурієнтів.
4. Редагування спеціальностей.

Нефункціональні вимоги адміністратора:

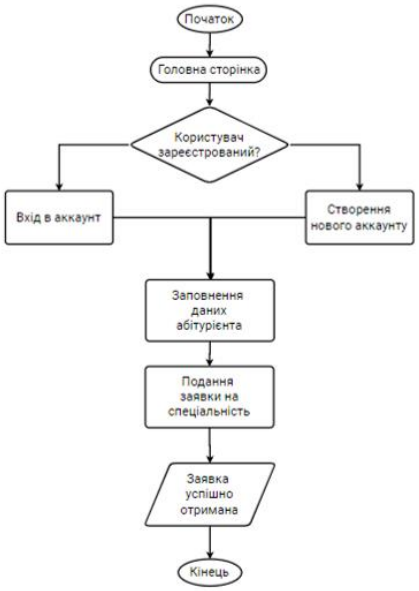
1. Безпека даних наданих користувачем.
2. Обмеження відповідно до ролі користувача.
3. Продуктивність.

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

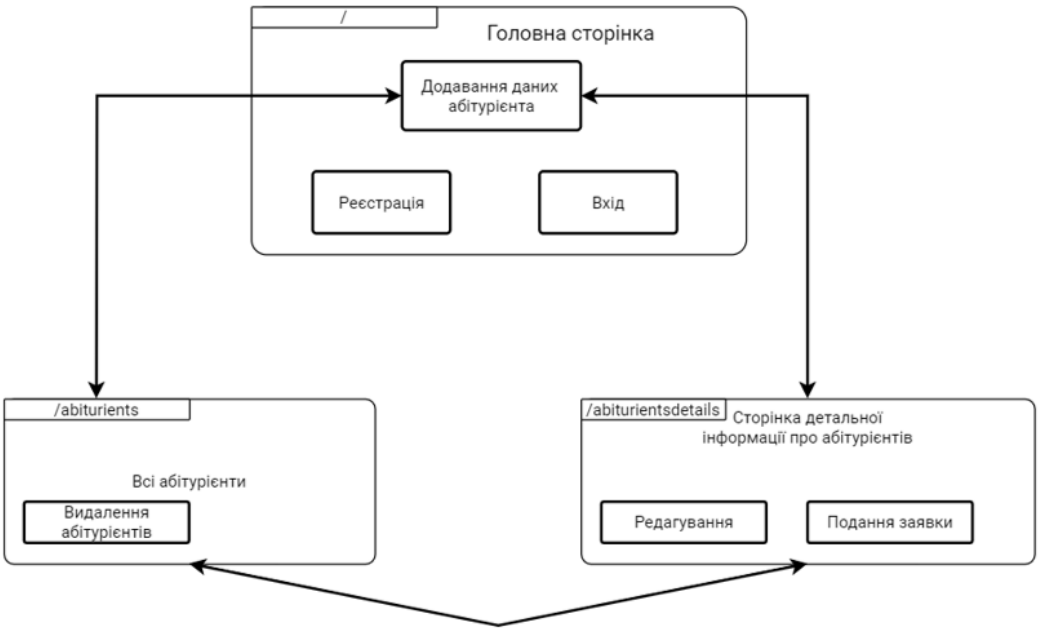
ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



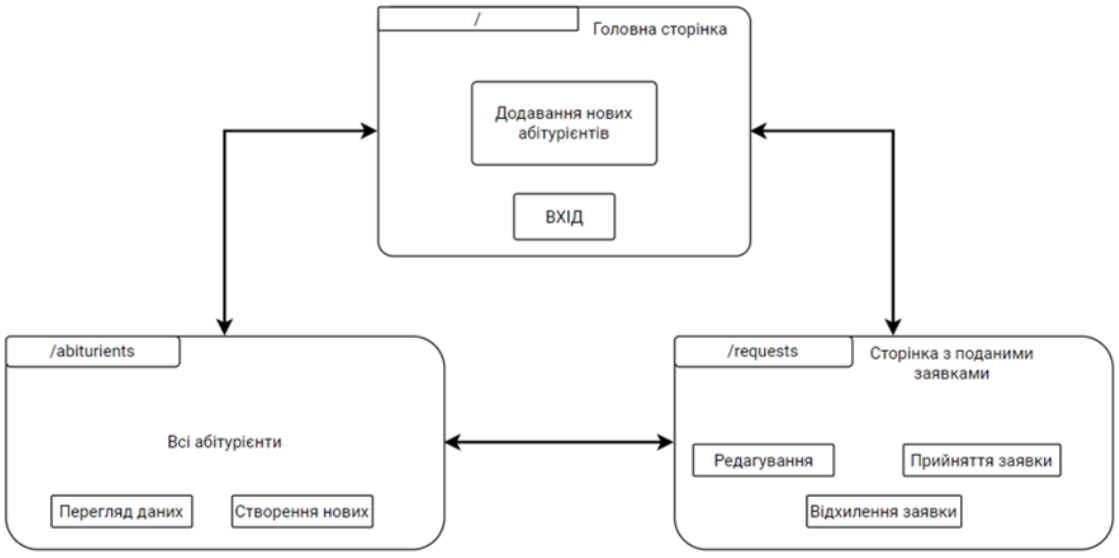
БЛОК-СХЕМА ПРОЦЕСУ ПОДАННЯ ЗА’ЯВКИ



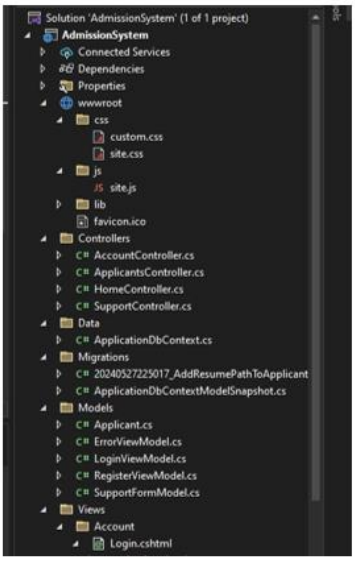
КАРТА ВЕБ-ЗАСТОСУНКУ КОРИСТУВАЧА



КАРТА ВЕБ-ЗАСТОСУНКУ АДМІНІСТРАТОРА



ФАЙЛОВА СТРУКТУРА ПРОЕКТУ



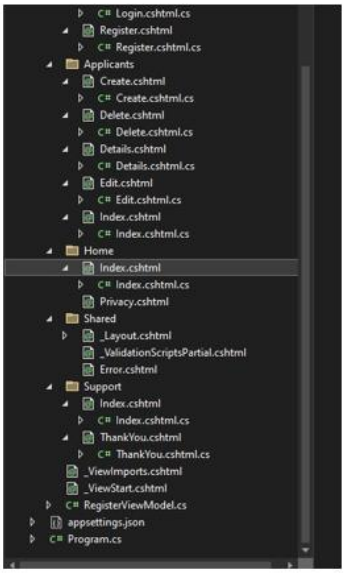
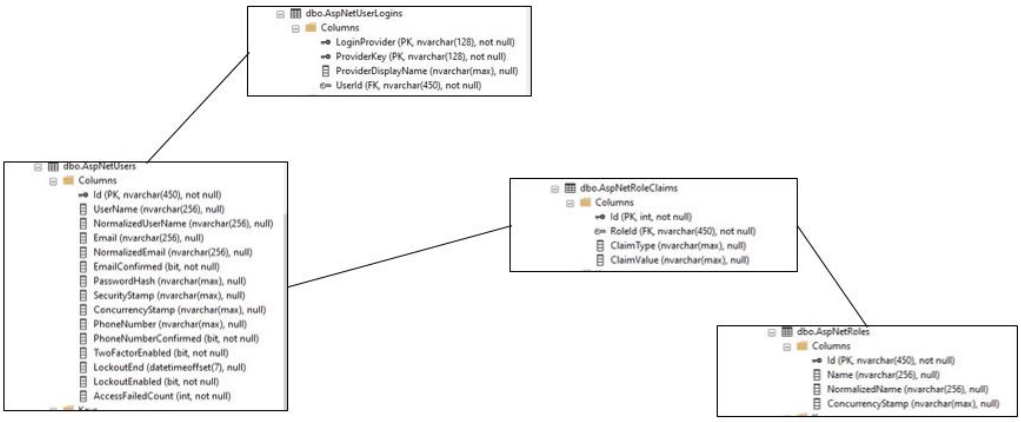
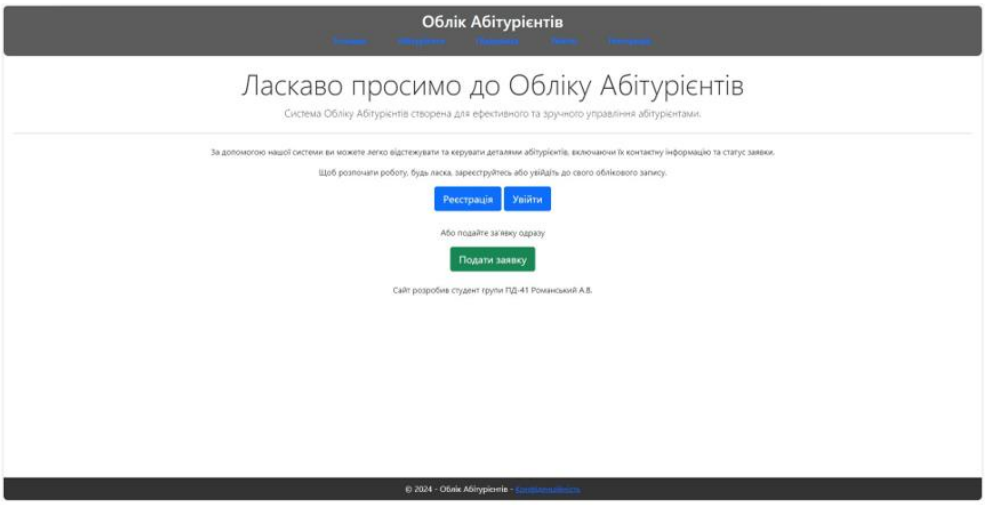


СХЕМА БАЗИ ДАНИХ

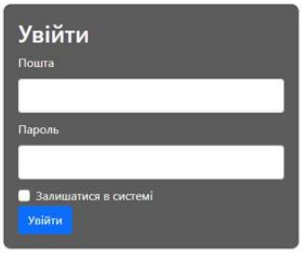


ЕКРАННІ ФОРМИ

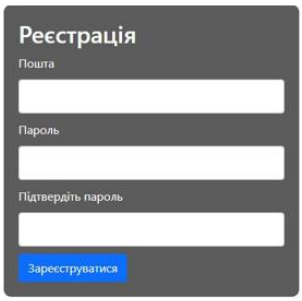


ГОЛОВНА СТОРІНКА

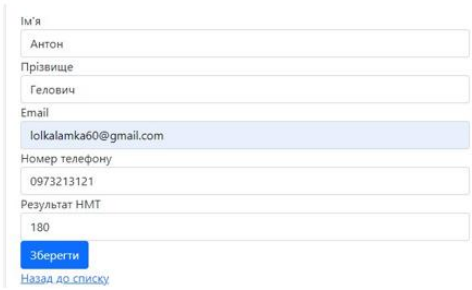
ЕКРАННІ ФОРМИ



ФОРМА ДЛЯ ВХОДУ



ФОРМА ДЛЯ РЕЄСТРАЦІЇ



ФОРМА ДЛЯ ВНЕСЕННЯ ДАННИХ АБІТУРІЄНТА

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1.Романський А.В., Гребенюк В.В. Визначення засобів реалізації Web-платформи для обліку абітурієнтів кафедри: Матеріали Всеукраїнської науково-практичної конференції “Сучасні інтелектуальні інформаційні технології в науці та освіті”. Збірник тез. 15.05.2024, ДУКІТ, м.Київ К.: ДУКІТ- Подано до друку

ВИСНОВКИ

1. Проведено всебічний аналіз галузі управління абітурієнтами, що дозволило ідентифікувати основні потреби та вимоги сучасних систем обліку. Виявлені основні проблеми та можливості для вдосконалення процесу обліку абітурієнтів.

2. Досліджено існуючі рішення для управління абітурієнтами, що дозволило визначити їх переваги та недоліки. Це дало змогу виділити найкращі практики та уникнути недоліків у власному проекті.

3. На основі аналізу галузі та готових рішень сформовано вимоги до програмного забезпечення для системи обліку абітурієнтів. Вимоги включають функціональні та нефункціональні аспекти, що забезпечують ефективність, надійність та зручність використання системи.

4. Проведено детальний аналіз різних підходів до розробки веб-сайтів з використанням ASP.NET та C#. Обрано найбільш підходящі бібліотеки, базу даних та інші компоненти для реалізації проекту, що забезпечують оптимальну продуктивність та масштабованість системи.

5. Проведено тестування реалізованого веб-додатку, що включає функціональне тестування, тестування на безпеку та продуктивність. Виявлені та усунуті недоліки, що забезпечує високу якість та надійність системи в експлуатації.