МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота бакалавра

ІНФОРМАЦІЙНА СИСТЕМА ДЕКАНАТУ ДЛЯ КЕРУВАННЯ ДОВІДКАМИ ЗА ДОПОМОГОЮ ТЕХНОЛОГІЇ ASP.NET

Здобувачка освіти гр. ІН – 82 Катерина ПОГОРІЛА

Науковий керівник,
кандидат технічних наук,
доцент Сергій ПЕТРОВ

Завідувач кафедри
доктор технічних наук, професор Анатолій ДОВБИШ

СУМИ 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Затверджую_____
Зав. кафедрою Довбиш А.С.
"_____" _____2022 р.

ЗАВДАННЯ

до кваліфікаційної роботи

здобувачки вищої освіти четвертого курсу, групи ІН-82 спеціальності «122 – Комп'ютерні науки» денної форми навчання Погорілої Катерини Віталіївни.

Тема: «ІНФОРМАЦІЙНА СИСТЕМА ДЕКАНАТУ ДЛЯ КЕРУВАННЯ ДОВІДКАМИ ЗА ДОПОМОГОЮ ТЕХНОЛОГІЇ ASP.NET»

Затверджена наказом по СумДУ
№_____ від _____2022 р.

Зміст пояснювальної записки: 1) літературний огляд за обраною тематикою роботи; 2) постановка завдання для розробки; 3) вибір оптимальних інструментів для розробки мобільного додатку; 4) практична реалізація.

Дата видачі завдання «_____»_____ 2022 р.
Керівник роботи _____Сергій ПЕТРОВ
Завдання прийняла до виконання _____ Катерина ПОГОРІЛА

РЕФЕРАТ

Записка: 33 стор., 15 рис., 1 додаток, 10 літературних джерел.

Об’єкт дослідження — Інформаційна система деканату для керування довідками за допомогою технології ASP.NET.

Мета роботи — розробка веб-додатку за допомогою технології ASP.NET для керування довідками.

Результати — проведений аналіз літератури, існуючих систем та технологій, що використовуються для їх реалізації. Після ознайомлення з існуючими рішеннями було розроблено мапу сайту, алгоритм роботи та програмну реалізацію . За допомогою даної розробки можна керувати обігом довідок в деканаті, створювати замовлення на отримання довідки. Додаток реалізовано за допомогою фреймворку ASP.NET.

ІНФОРМАЦІЙНА СИСТЕМА ДЕКАНАТУ ДЛЯ КЕРУВАННЯ ДОВІДКАМИ ЗА ДОПОМОГОЮ ТЕХНОЛОГІЇ ASP.NET, INFORMATION SYSTEM FOR THE DEANERY FOR CERTIFICATES MANAGEMENT WITH THE HELP OF ASP.NET TECHNOLOGY

ЗМІСТ

ВСТУП

Робота з документами це одна з основних «бізнес стратегій», що поєднує деканат та студентів. Але з кожним роком це стає дедалі складніше, через безліч умов, які перешкоджають очним зустрічам в університеті. Дистанційне навчання, віруси, війна, внутрішнє переміщення змушує нас вирішувати багато питань дистанційно, використовуючи telegram, email або дзвінки. Інформація забувається, втрачається серед великої кількості повідомлень або стає неактуальним, у ході тривалого листування поштою.

Через вищезазначене дуже важливо підтримувати стійкі та сталі дистанційні відносини між деканатом та студентами, використовуючи вже розроблені системи (Особистий кабінет СумДУ, Міх), та розробляючи нові.

Мета роботи – розробити таку систему видачі довідок, якою було б зручно користуватися і студентам, і деканату, зберігати важливу інформацію в цифровому, а не паперовому вигляді, що полегшує доступність та цілісність даних, а також забезпечення дистанційної комунікації між деканатом та студентами.

1. ЛІТЕРАТУРНИЙ ОГЛЯД ЗА ОБРАНОЮ ТЕМАТИКОЮ РОБОТИ

1.1 Аналіз існуючих інформаційних систем

Інформаційна система це набір пов’язаних компонентів, що працюють разом для збирання, обробки, зберігання, аналізу та поширення інформації для визначених цілей[1]. Ця інформація підтримує фундаментальні бізнес операції, звітність та візуалізацію даних, аналіз даних, прийняття рішень, комунікації та координацію поміж організацією. Гарно спроектована інформаційна система включає в себе механізм зворотного зв’язку для моніторингу та контролю роботи системи. [2]

Зараз багато хто користується таким додатком, як Дія, кожен підприємець повинен мати справу з «Кабінетом платника податків», а кожен студент СумДУ має свій профіль в особистому кабінеті. Це також є прикладам своєрідних інформаційних систем, що допомагають керуванню стосунків, тільки не з клієнтами, а з громадянами або студентами.

Після перегляду та аналізу систем було виявлено основні недоліки:

1. Надмірність
2. Проблематичність додавання нового функціоналу
3. Безпека
4. Шаблонність
5. Недостатня інформованість користувача
6. Незручний, інтуїтивно незрозумілий інтерфейс

Після визначення можливих недоліків потрібно запобігати їх появі на кожному з етапів розробки.

1.2 Постановка задачі

У рамках виконання дипломної роботи необхідно реалізувати наступні задачі:

1. виконати огляд інструментів для розробки та обрати оптимальні;
2. розробити інтуїтивно зрозумілий інтерфейс інформаційної системи;
3. виконати проектування архітектури веб-додатка;
4. розробити алгоритм роботи;
5. виконати мануальне тестування системи та зробити відповідні висновки по роботі.

У разі успішної реалізації буде можливим слідкувати за існуючими довідками, формувати запити про довідки, надсилати довідки, відповідаючи на запит, переглядати інформацію про запити.

2. МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

2.1 Проектування бази даних

Основною функцією для інформаційної системи є зберігання та управління даними, тому для розробки обов'язково знадобиться база даних. На рисунку 2.1 представлено архітектуру бази даних, що розроблена за допомогою сервісу SQLDBD [3] та пристосована для діалекту PostgreSQL.

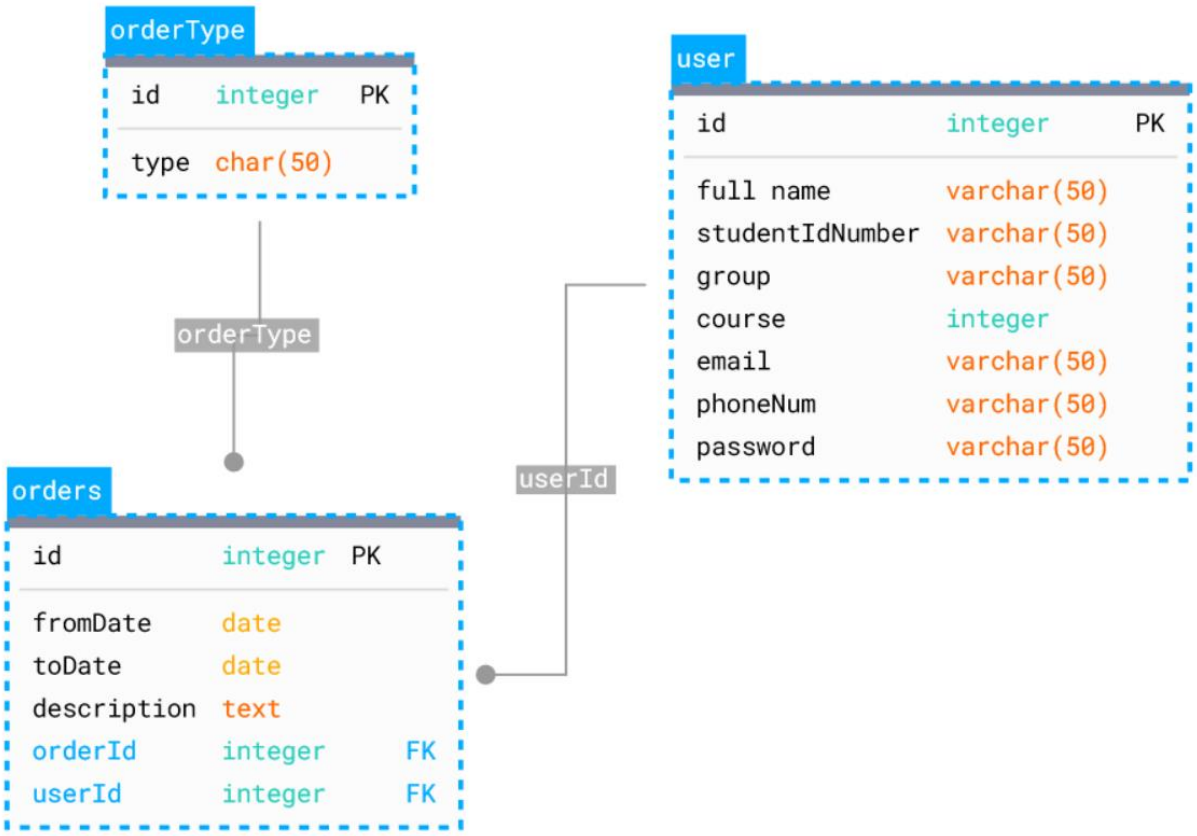

Рисунок 0.1 – Архітектура бази даних

Було створено три сутності: користувач, замовлення і тип замовлення (таблиця, що буде містити у собі тип довідки з унікальним id).

2.2 Мапа сайту

Для кращого розуміння структури майбутньої інформаційної системи було розроблено мапу сайту (рис.2.2) за допомогою сервісу FlowMapp [4].

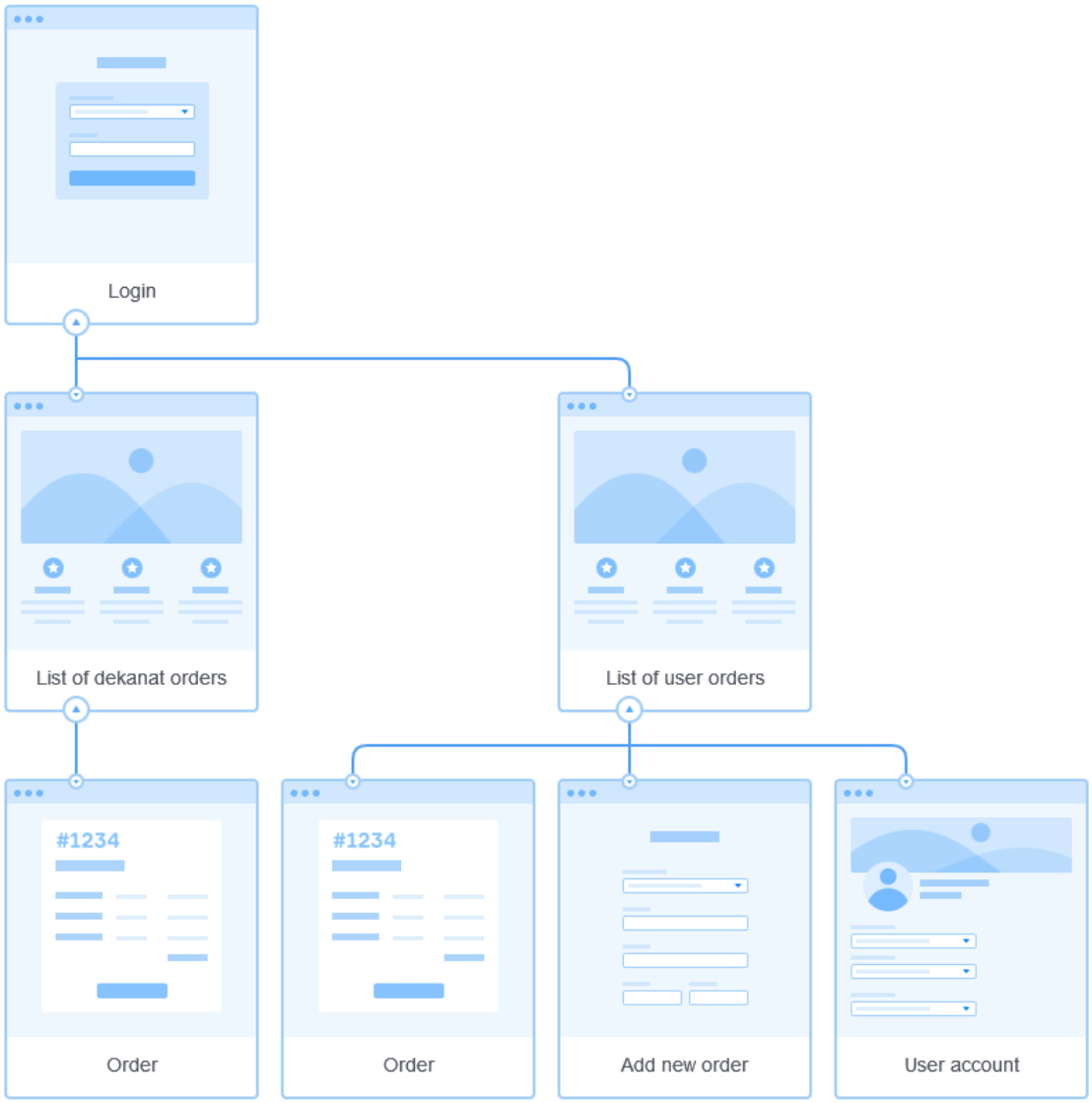

Рисунок 0.2 – Мапа сайту

Після авторизації є два види доступу:

1. Адмін. Для робітників деканату, що зможуть переглядати список заявок на отримання довідки, змінювати статус заявки, керувати ними та надсилати довідки.

2. Користувач. Для студентів, що хочуть надіслати запит на отримання довідки. Є можливість переглянути власний профіль, список запитів для довідок або додати запит.

Також є можливість переглянути деталі про кожен запит окремо. Кожен запит матиме унікальний ідентифікатор, що надає можливість уникнути непорозумінь і знайти запит, якщо з’явиться така потреба.

2.3 Фреймворк ASP.NET

ASP.NET є наступником «класичної» технології Microsoft ASP, провідного у світі інструменту веб-розробки. ASP.NET вирішує багато проблем, пов'язаних з ASP, і забезпечує інтегрований і послідовний підхід до розробки програмного забезпечення, який базується на бібліотеках і мовах платформи .NET.[5]

Зараз існує досить багато можливостей для створення веб-додатків. Можна використовувати PHP, Java, Python. ASP.NET має багато переваг поміж інших можливих програмних рішень.

1. Відкриті вихідні джерела, потужні пакетні менеджери (наприклад, NuGet).
2. Нові версії SDK і рантайму випускаються регулярно і майже не містять несумісних змін, оновлення версії проходить завжди плюс-мінус гладко.
3. Поріг входження для нових розробників не дуже високий, і в 99% випадків можна знайти готову бібліотеку під потрібне завдання. Платформа успішно знаходить золоту середину між безпекою коду та швидкодією.
4. Розробники в більшості випадків можуть не турбуватися про споживання пам'яті або тонкощі взаємодії з ресурсами ОС і очікувати, що платформа виконає поставлене завдання оптимально.

2.4 Архітектура проекту

Для програмної реалізації було вибрано архітектурну модель MVC. Архітектурний шаблон Model-View-Controller (MVC) розділяє програму на три основні групи компонентів: моделі, представлення та контролери. Цей шаблон допомагає досягти розділення проблем. Використовуючи цей шаблон, запити користувачів перенаправляються на контролер, який відповідає за роботу з моделлю для виконання дії користувача та/або отримання результатів запитів. Контролер вибирає представлення для відображення користувачеві та надає його з усіма необхідними даними моделі [6].

Як і кожен підхід, він має свої переваги та недоліки. Переваги MVC шаблону:

- Дозволяє регулювати складність додатку, поділяючи його на три частини.
- Не використовує форми, базовані на сервері, тому підходить для тих розробників, що хочуть повністю еонтролювати поведінку додатку.
- Використовує «front controller» шаблон. Що допомагає обробляти багато вхідних запитів, а також забезпечує централізований контроль.
- Забезпечує підтримку в маршрутизації зв’язків, для створення додатку.

Недоліки MVC шаблону:

- Важко зрозуміти архітектуру.
- Має чіткі правила, щодо методів.
- Немає вбудованого контролю користувача для веб-сторінок.
- Можуть виникнути складнощі з розгортанням на сервері[7].

Не дивлячись на недоліки шаблону, його опанування та доцільне використання зазвичай дає набагато більше переваг, у вигляді повного розуміння архітектури проекту та чітких правил розташування файлів. Це впливає на здатність краще орієнтуватися в проекті та простіше підтримувати його життєдіяльність в довгостроковій перспективі.

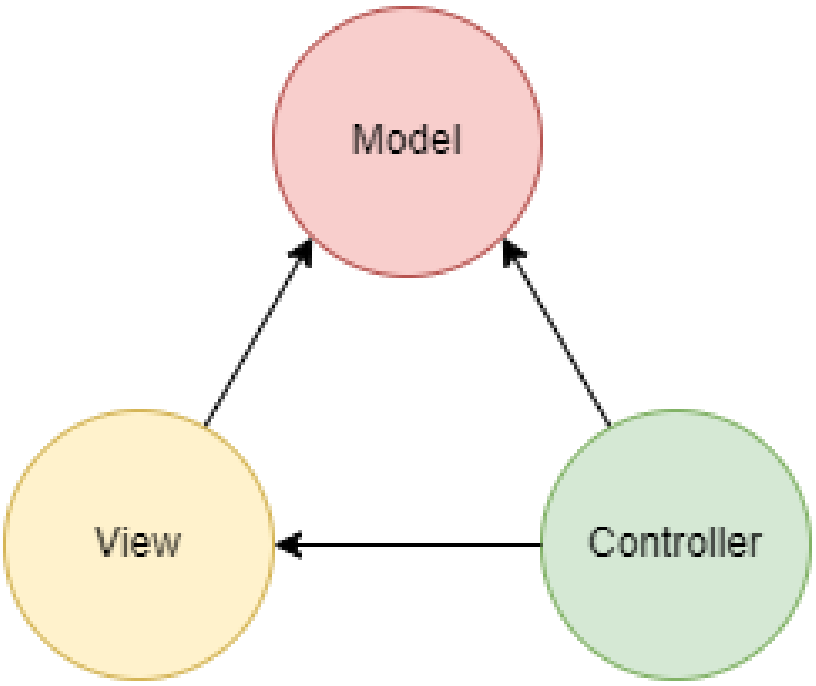

Рисунок 0.3 – Діаграма MVC шаблону проектування

На рисунку 2.3 зображено класичний вигляд діаграми MVC шаблону з усіма можливими існуючим складовими.

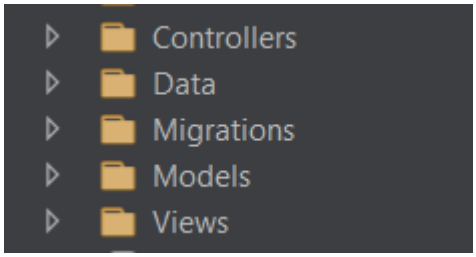

Рисунок 0.4 – Мапа сайту

Згідно даного підходу у проекті було створено основні теки: Models, Views та Controllers (рис 2.4). Крім цього проект містить додаткові теки, такі як Data – для файлів, що керують під'єднанням до бази даних, та Migrations – для зберігання міграцій з БД

2.5 Засоби створення інтерфейсу

Основою кожної веб-сторінки є HTML (мова розмітки) та CSS (мова таблиці стилів). Використання .Net технологій дозволяє при розробці використовувати також технологію Razor pages.

Razor pages це синтаксис розмітки для впровадження коду, написаного за допомогою .Net у веб сторінки. Синтаксис складається з розмітки Razor, C#

та HTML. Файли, що містять Razor, мають розширення файлу .cshtml. Синтаксис Razor аналогічний механізмам таких фреймоврків, як Angular, React та VueJs[8].

Основні принципи Razor синтаксису для C#:

- C# блоки коду вкладені в @{ ... }.
- Вбудовані вирази (змінні або функції) починаються з @
- Операції коду закінчуються крапкою з комою
- Змінні оголошуються за допомогою ключового слова var або типу даних (int, string тощо)
- Код чутливий до регістру
- Файли мають розширення .cshtml. [9]

2.6 Можливість використання OpenID

Кожен студент СумДУ обов'язково має аккаунт в особистому кабінеті, що представляє собою різноманіття персоналізованих інформаційних сервісів. І ця інформаційна система може стати одним з таких. Це можна реалізувати за допомогою технології OpenID.

OpenID Connect (OIDC) — це протокол аутентифікації, заснований на протоколі OAuth2 (який використовується для авторизації). OIDC використовує стандартизовані потоки повідомлень від OAuth2 для надання послуг ідентифікації.

Мета дизайну OIDC – «зробити прості речі простими, а складні – можливими». OIDC дозволяє розробникам автентифікувати своїх користувачів на веб-сайтах і в додатках без необхідності володіти файлами паролів і керувати ними. Це надає розробнику додатків безпечний спосіб перевірити особу людини, яка в даний момент використовує веб-переглядач або рідну програму, підключену до програми [10]. Тобто, студент може мати тільки один аккаунт, використовуючи при цьому декілька функціонально різних сервісів.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Моделі системи

Основною моделлю системи є модель Order. Це клас, що описує об’єкт замовлення довідки. Він містить 6 полів, одне з яких – тип перерахування, або Enumerable. Це потрібно для того, щоб забезпечити замовленню присвоєння тільки певних статусів (Created, In_progress, Rejected, Done). Також там присутні такі поля, як:

- Ідентифікатор замовлення
- Дата початку
- Дата кінця
- Опис
- Ідентифікатор студента
- Статус замовлення.

Всі ці поля потрібні для правильної ідентифкації та керуванням інформацією про довідки. Дана модель описана у коді наступним чином:

```
 namespace AskDekanat.Models
{
   public enum Status
   {
      Created,
      In_progress,
      Rejected,
      Done
   }
   public class Order
   {
      public int OrderId { get; set; }
      public DateTime FromDate { get; set; }
      public DateTime ToDate { get; set; }
      public string Description { get; set; }
      public string StudentId { get; set; }
      public Status OrderStatus { get; set; }
   }
}
```

3.2 Програмна реалізація

Найважливішою функціональністю для вдалої розробки проектів, було створення різних ролей для користувачів. В даному додатку створення ролі адміністратора було реалізовано при ініціалізації контексту бази даних. Алгоритм роботи методу полягає в перевірці БД на існування ролі та додавання її, якщо вона відсутня.

```
namespace AskDekanat.Data
{
   public class DbInitializer
   {
      public static async Task Initialize(IServiceProvider serviceProvider)
      {
         var context = serviceProvider
            .GetRequiredService<ApplicationDbContext>();
         context.Database.EnsureCreated();
         var roleManager = serviceProvider
            .GetRequiredService<RoleManager<IdentityRole>>();
         var roleName = "Administrator";
         IdentityResult result;
         var roleExist = await roleManager.RoleExistsAsync(roleName);
         if (!roleExist)
         {
            result = await roleManager
               .CreateAsync(new IdentityRole(roleName));
            if (result.Succeeded)
            {
               var userManager = serviceProvider
                  .GetRequiredService<UserManager<ApplicationUser>>();
               var config = serviceProvider
                  .GetRequiredService<IConfiguration>();
               var admin = await userManager
                  .FindByEmailAsync(config["AdminCredentials:Email"]);

               if (admin == null)
               {
                  admin = new ApplicationUser()
                  {
                     UserName = config["AdminCredentials:Email"],
                     Email = config["AdminCredentials:Email"],
                     EmailConfirmed = true
                  };
```

```
                result = await userManager
                    .CreateAsync(admin, config["AdminCredentials:Password"]);
                if (result.Succeeded)
                {
                    result = await userManager
                        .AddToRoleAsync(admin, roleName);
                }
            }
        }
    }
}
```

Далі у ході розробки перевірка на наявність прав користувача здійснюється викликом одного методу, наприклад User.IsInRole("Administrator").

3.4 Використання Razor pages для створення інтерфейсу

Razor pages спрощує поєднання моделі та контролеру у представленні таким чином, щоб програмний код, впроваджуваний у розмітку сторінки, не заважав розумінню написаного.

На початку сторінки задаються використовувані змінні, моделі, бібліотеки та впроваджувані залежності. Наприклад:

```
@{ ViewData["Title"] = "Home Page";}
@using Microsoft.AspNetCore.Identity
@inject SignInManager<ApplicationUser> SignInManager
@inject UserManager<ApplicationUser> UserManager
@model IEnumerable<AskDekanat.Models.Order>
```

В наведеній частині коду оголошується заголовок сторінки (“Home Page”), бібліотека Identity та оголошується використовувана модель IEnumerable<AskDekanat.Models.Order> (список замовлень). Далі цю модель буде використано для виведення на сторінку інформації щодо замовлень:

```
        <tbody>
            @foreach (var item in Model)
        {
                <tr style="background-color: @isItem(item.OrderStatus)">
                    <td width="20%"><a
href="/Order/Index/@item.OrderId">@item.OrderId</a></td>
```

```
                <td width="20%">@item.StudentId</td>
                <td width="20%">@item.FromDate</td>
                <td width="20%">@item.ToDate</td>
                <td width="20%">@item.Description</td>
                <td>
                    @Html.ActionLink("Delete", "Delete", new { id =
item.OrderId })
                </td>
            </tr>
        }
    </tbody>
```

Виклик функції Html.ActionLink на даному прикладі забезпечує відправку запиту на сервер, щодо видалення конкретного замовлення за його ідентифікатором.

3.3 Користувацький інтерфейс

На початку роботи було створено мапу сайту, для розуміння того, які сторінки повинен мати веб-додаток, задля повнофункціональної роботи з інформаційною системою. Також потрібно було, щоб вигляд сайту був мінімалістичним та покривав мінімальні потреби користувачів.

З першого погляду на сторінку можна зрозуміти, чи зручно буде користуватись даним додатком. На першій сторінці (рис. 3.1) було розміщено повідомлення про ціль розроблення сайту, а зверху можна побачити меню для реєстрації та логіну.

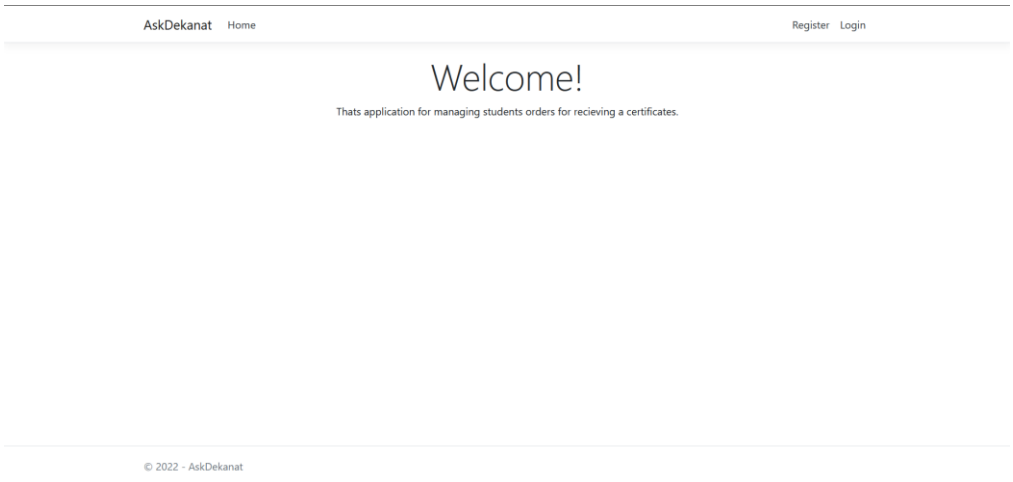

Рисунок 3.1 – Welcome-сторінка для незареєстрованого користувача

Далі користувач може зареєструватися, якщо не має аккаунту, або увійти у свій аккаунт, натиснувши на меню «Login». На рисунку 3.2 можна побачити сторінку реєстрації користувача з полями, що покривають усі необхідні для видачі довідок дані.

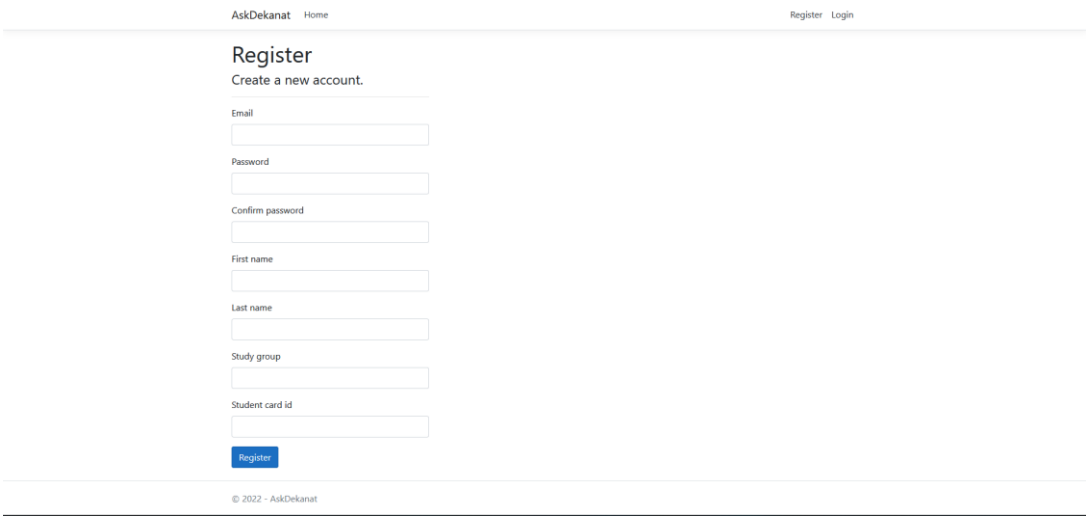

Рисунок 3.2 – Сторінка для реєстрації

Якщо користувач вже зареєстрований, то для авторизації йому потрібно вводити тільки його пошту та пароль, вказаний при реєстрації.

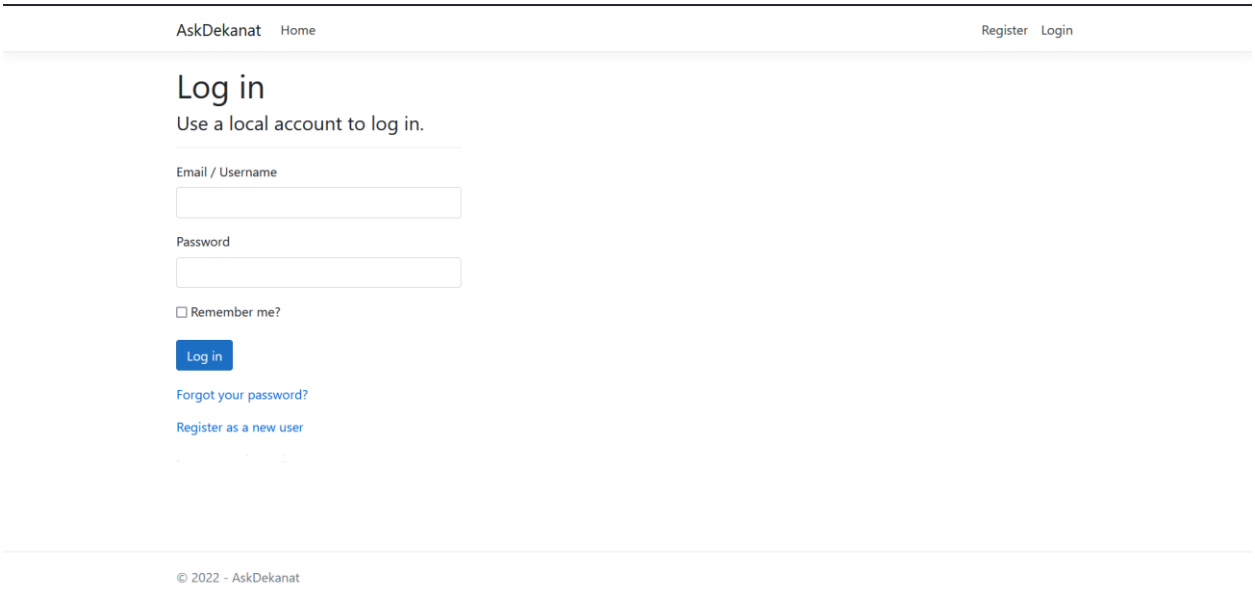

Рисунок 3.3 – Сторінка для логіну

Після реєстрації користувач має змогу переглядати свої заявки на отримання довідки, створювати нові заявки, видаляти старі, а також змінювати дані свого аккаунту. На рисунку 3.4 зображено головну сторінку сайту для

звичайного користувача, кожен колір у таблиці відповідає поточному статусу заявки. Синій – створена, жовтий – в процесі, червоний – відхилена, зелений – виконана.

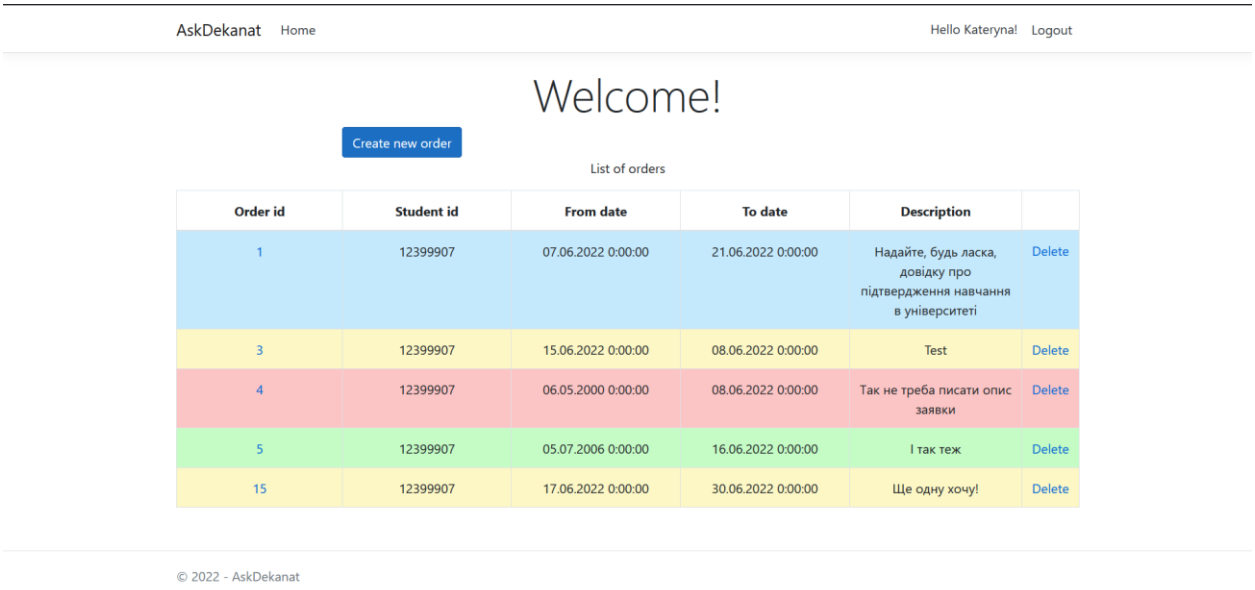

Рисунок 3.4 – Вигляд головної сторінки користувача

При натисканні кнопки “Create new order” користувач переадресовується на сторінку з формою (рис 3.5), що дозволяє вказати дату актуальності даних для довідки та написати додаткову інформацію у полі з назвою «description»

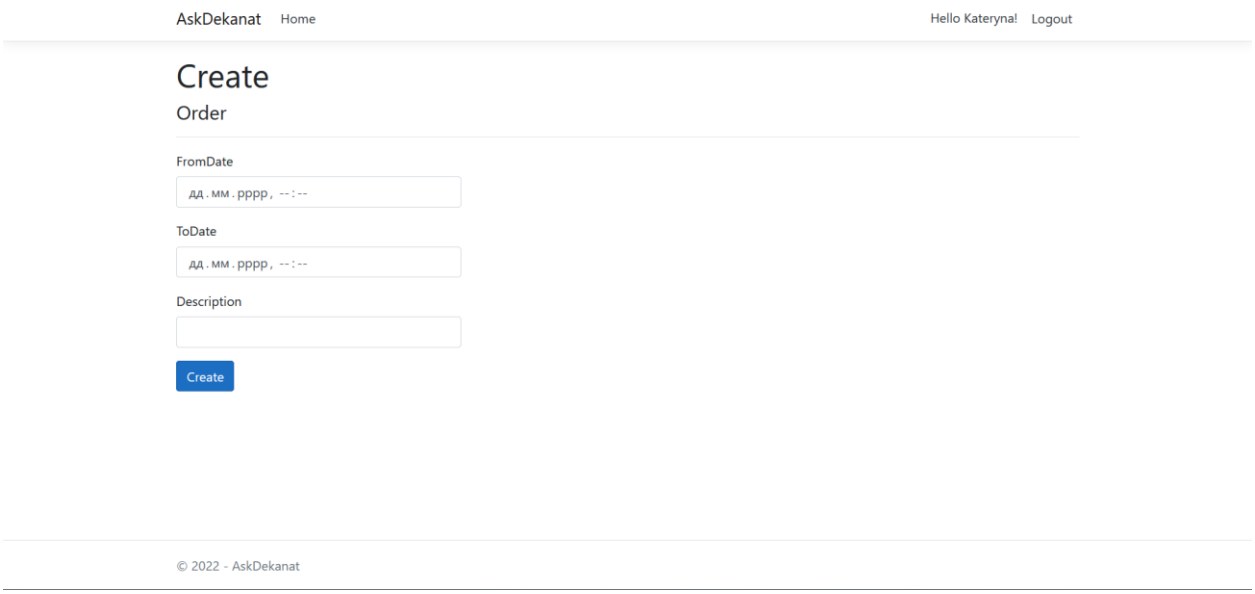

Рисунок 3.5 – Форма створення запиту на отримання довідки

При натисканні на id замовлення на головній сторінці, користувач має змогу перейти на сторінку з повною інформацією про довідку, що було замовлено. Сторінка зображена на рисунку 3.6.

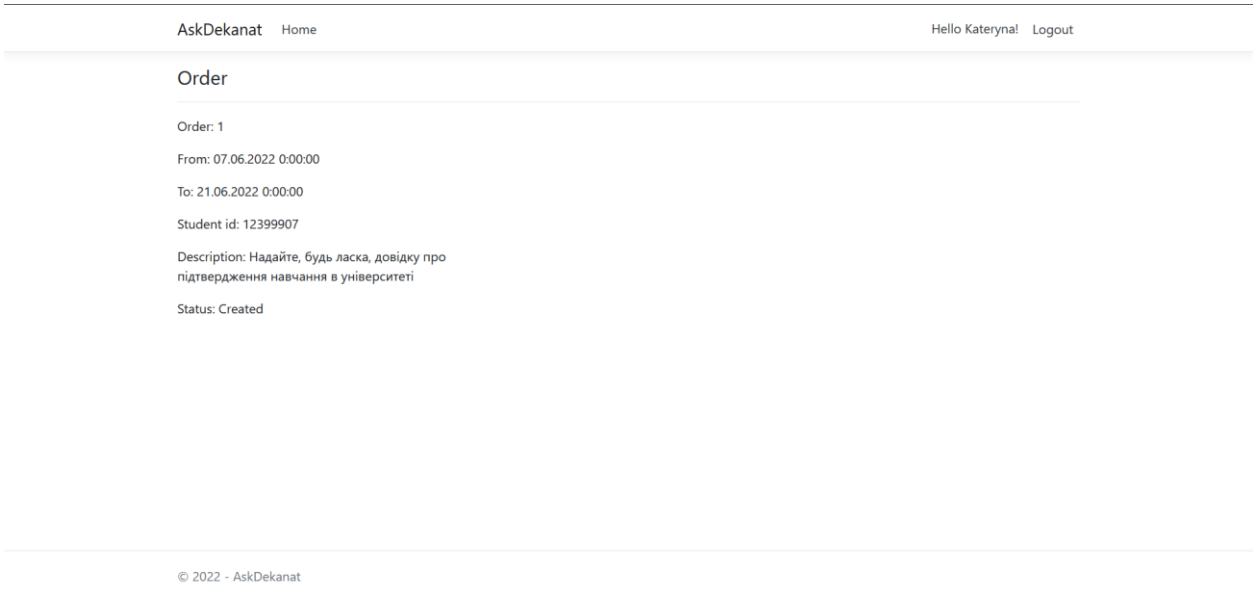

Рисунок 3.6 – Вигляд сторінки замовлення довідки для користувача

Також користувач має змогу змінити свої персональні дані, за допомогою зміни даних у полях форми. Перейти на цю сторінку можна натиснувши на меню «Hello, {ім’я користувача}!».

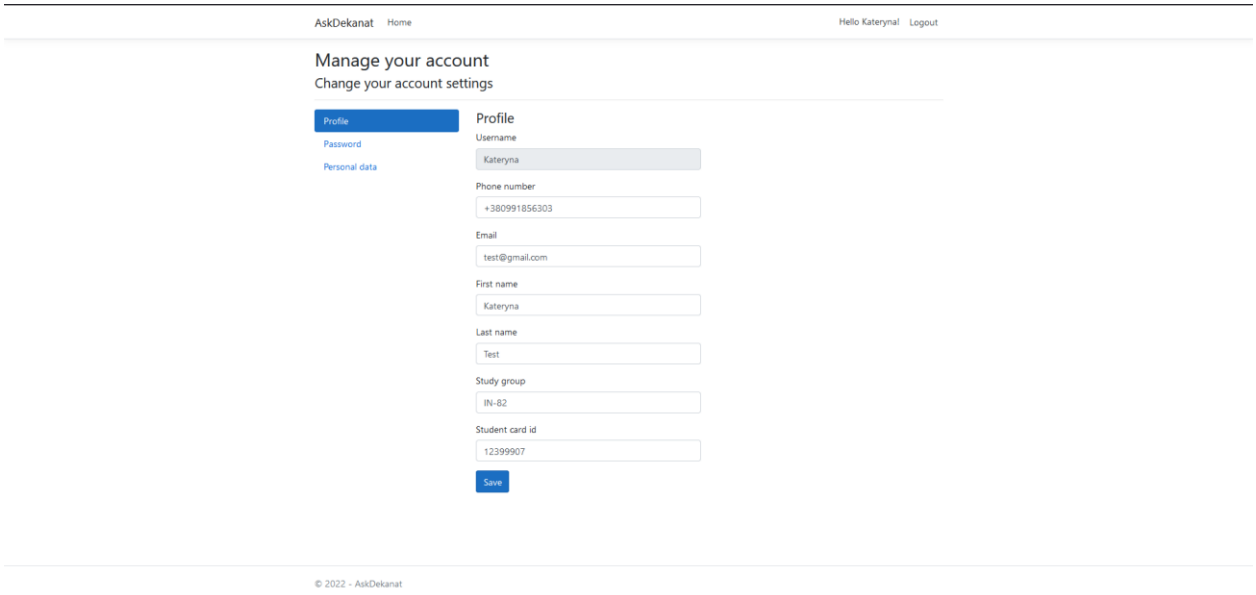

Рисунок 3.7 – Сторінка профіля користувача

На цій сторінці також можна побачити меню зліва, що дає доступ також до зміни пароля та видалення аккаунту користувача. Для того, щоб змінити пароль (рис 3.8), потрібно вказати поточний та новий паролі. Останній – два рази, для підтвердження.

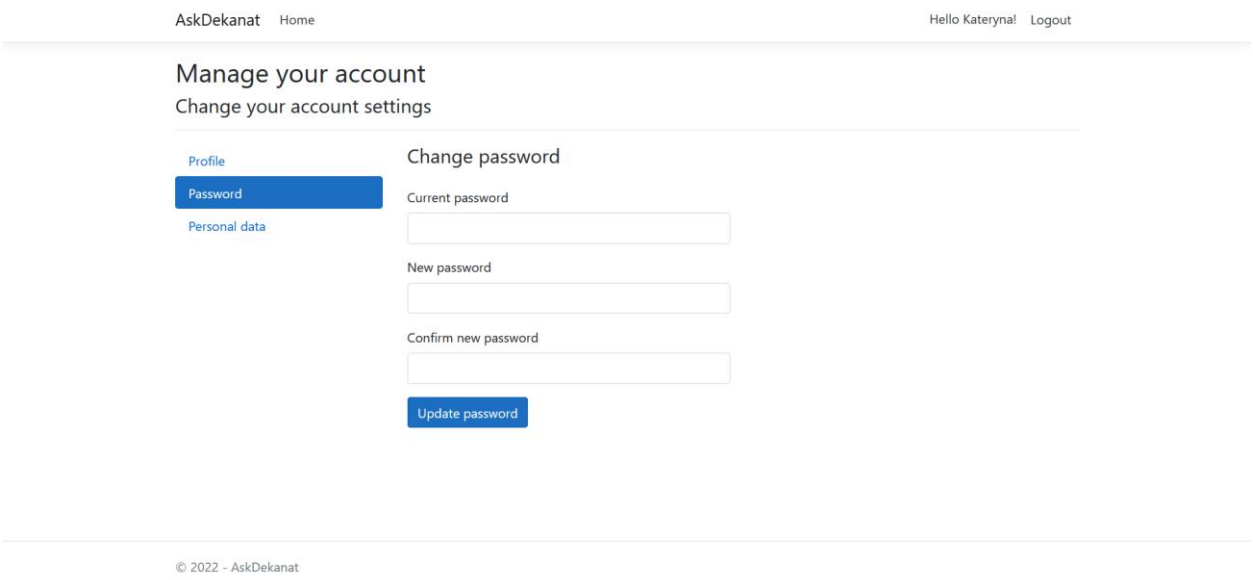

Рисунок 3.8 – Сторінка зміни паролю користувача

Якщо користувач не хоче мати аккаунт, або хоче завантажити свої персональні дані зі сторінки, також є така можливість (рис 3.9). У меню потрібно обрати відповідну дію та натиснути на кнопку.

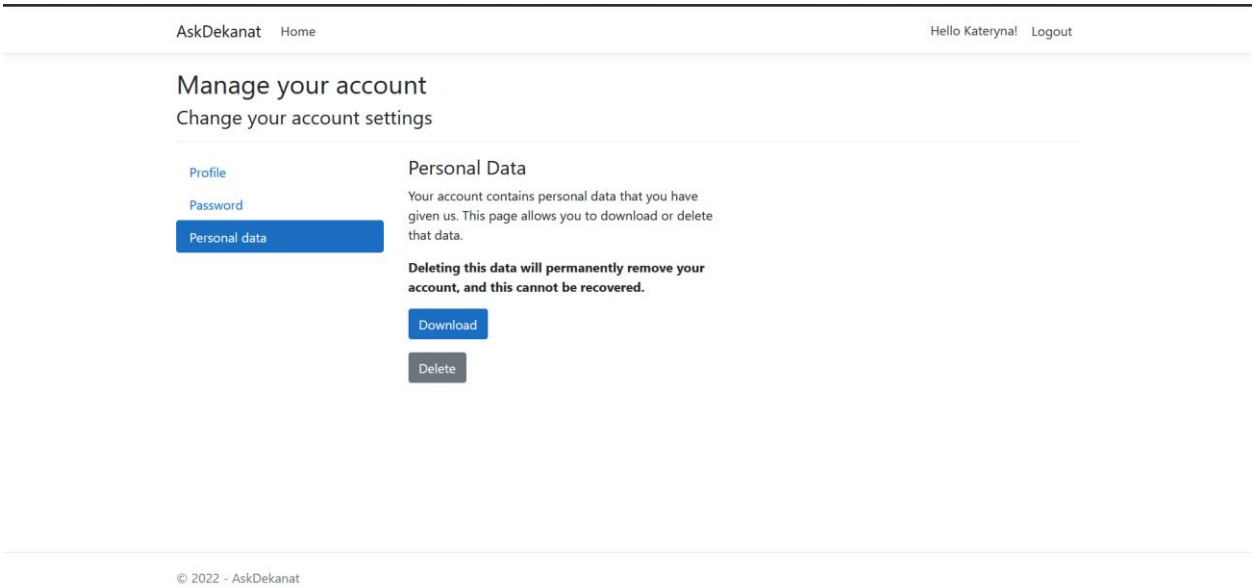

Рисунок 3.9 – Сторінка видалення аккаунту користувача

Зі сторони працівника деканату (адміністратора) сторінка виглядала б схоже на ту, що й у користувача, без прав адміністрування. Але є певні відмінності. Наприклад, адміністратор має змогу переглядати замовлення від всіх користувачів (рис 3.10).

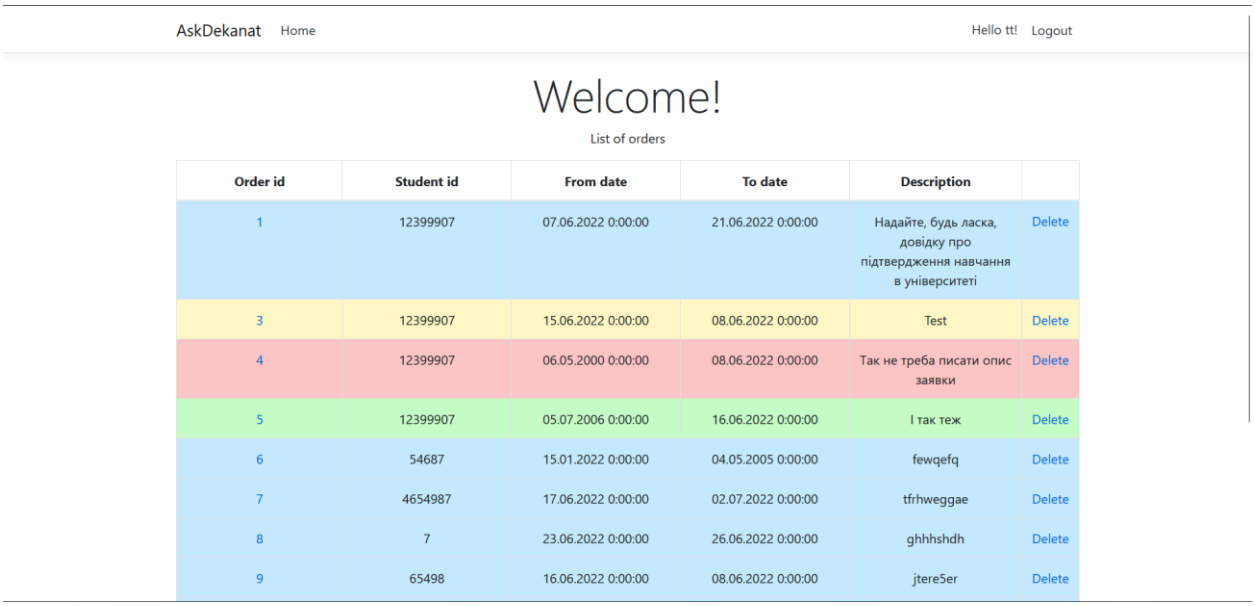

Рисунок 3.10 – Вигляд головної сторінки адміністратора

Також є різниця на сторінці перегляду інформації про окреме замовлення – користувач з роллю адміністратора має змогу змінювати статус замовлення (рис 3.11).

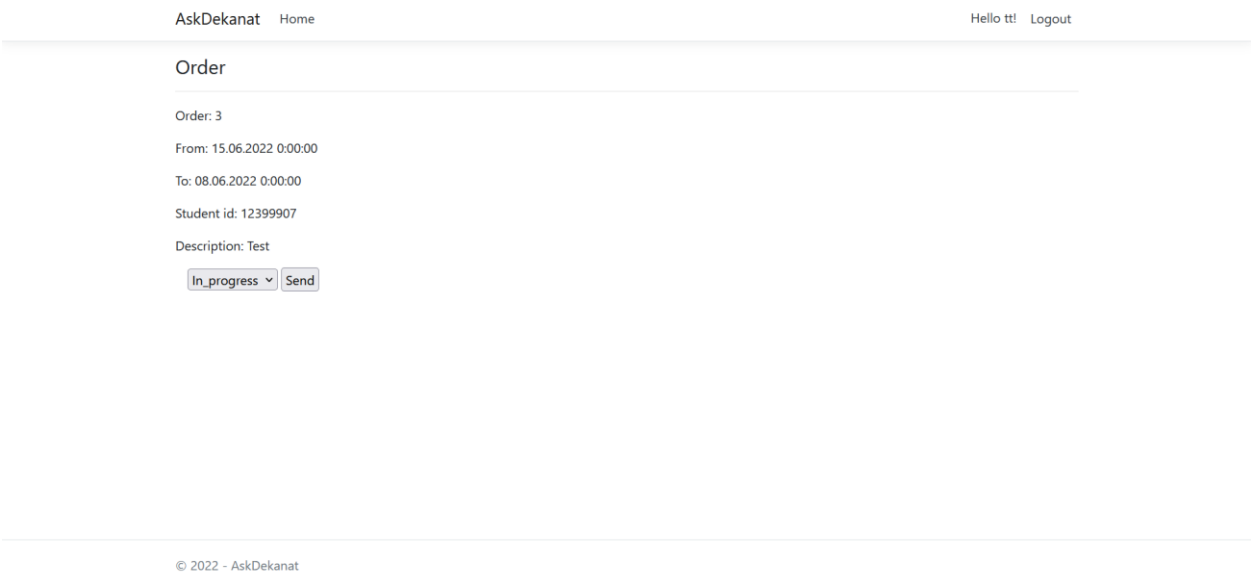

Рисунок 3.11 – Вигляд сторінки замовлення довідки для адміністратора

Тож, користувацький інтерфейс інформаційної системи деканату має достатньо функціоналу та зручності для обліку обігу отриманих і замовлених довідок.

ВИСНОВКИ

У ході виконання дипломної роботи було зроблено літературний огляд сучасних джерел за тематикою інформаційних систем та розробки за допомогою технології ASP.NET, розглянуто доступні і обрано оптимальні інструменти для розробки веб-додатків, вирішено проблему архітектури веб-додатків та обрано оптимальний (MVC шаблон).

Також було створено програмну реалізацію інформаційної системи для керування довідками, виконано розподіл функціональності згідно ролям користувача, реалізовано можливість, створення та видалення запитів, адміністрування їх статусів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Prince B., Rainer R. K. Introduction to information systems supporting and transforming business. 9-те вид. Hoboken, NJ John Wiley & Sons, Inc., 2022.

2. Stair R., Reynolds G. Principles of information systems. 14-те вид. Mason, OH : Cengage, 2020.

3. SqlDBD [Електронний ресурс] // SqlDBD.com. – Режим доступу: https://sqldbd.com/app (дата звернення: 25.05.2022).

4. FlowMapp [Електронний ресурс] // FlowMapp. – Режим доступу: https://app.flowmapp.com/share/fdb51efa35e980ee9e1da5447f7289b5/ (дата звернення: 25.05.2022).

5. Liberty J. Programming ASP. NET / Jesse Liberty. – 2-ге вид. – Sebastopol, CA : O'Reilly, 2003. – 988 с.

6. Smith S. Overview of ASP.NET core MVC. 2017. URL: https://studreadywork.ru/wp-content/uploads/2021/09/Overview-of-ASP.NET-Core-MVC.pdf (дата звернення: 13.06.2022).

7. Singh S. MVC framework: a modern web application development approach and working. Irjet. 2020.

8. Anderson R., Mullen T., Vicarel D. Razor syntax reference for ASP.NET core. Microsoft Docs. URL: https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-6.0 (дата звернення: 14.06.2022).

9. ASP.NET Web Pages Razor. W3Schools. URL: https://www.w3schools.com/asp/webpages_razor.asp (дата звернення: 14.06.2022).

10. OpenID Connect authentication with Azure Active Directory - Microsoft Entra. Microsoft Docs. URL: https://docs.microsoft.com/en-us/azure/active-directory/fundamentals/auth-oidc#implement-oidc-with-azure-ad (дата звернення: 14.06.2022).

ДОДАТКИ

Додаток А. Лістинг програмного коду

HomeController.cs

```
using AskDekanat.Data;
using AskDekanat.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading.Tasks;

namespace AskDekanat.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;
        private readonly ApplicationDbContext _db;
        private readonly UserManager<ApplicationUser> _userManager;

        public HomeController(ILogger<HomeController> logger, ApplicationDbContext db,
UserManager<ApplicationUser> userManager)
        {
            _logger = logger;
            _db = db;
            _userManager = userManager;
        }

        public async Task<IActionResult> IndexAsync()
        {
            var model = _db.Orders.AsQueryable();
            if (!User.IsInRole("Administrator")){
                var user = await _userManager.GetUserAsync(HttpContext.User);
                model = model.Where<Order>(x => x.StudentId == user.StudentId);
            }
            return View(model);
        }

        public IActionResult Privacy()
        {
            return View();
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore =
true)]
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ??
HttpContext.TraceIdentifier });
        }

        public IActionResult Delete(int id)
        {
            _db.Orders.Remove(_db.Orders.FirstOrDefault(x => x.OrderId == id));
            _db.SaveChanges();
            return RedirectToAction("Index");
        }
```

```
    }
}
```

OrderController.cs

```
using AskDekanat.Data;
using AskDekanat.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using System.Web;

namespace AskDekanat.Controllers
{
    public class OrderController : Controller
    {
        private readonly ILogger<OrderController> _logger;
        private readonly ApplicationDbContext _db;
        private readonly UserManager<ApplicationUser> _userManager;

        public OrderController(ILogger<OrderController> logger, ApplicationDbContext db,
UserManager<ApplicationUser> userManager)
    {
            _logger = logger;
            _db = db;
            _userManager = userManager;
        }

        public IActionResult Index(int id)
        {
            var model = _db.Orders.SingleOrDefault( x => x.OrderId == id);
            return View(model);
        }

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

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> CreateAsync(Order obj)
        {
            var user = await _userManager.GetUserAsync(HttpContext.User);
            obj.StudentId = user.StudentId;
            _db.Orders.Add(obj);
            _db.SaveChanges();
            return Redirect("../");
        }

        public IActionResult Post(int id, string status)
        {
            var order = _db.Orders.SingleOrDefault(x => x.OrderId == id);
            switch (status) {
                case "Done":
```

```
                    {
                        order.OrderStatus = Status.Done;
                        break;
                    }
                case "Created":
                    {
                        order.OrderStatus = Status.Created;
                        break;
                    }
                case "In_progress":
                    {
                        order.OrderStatus = Status.In_progress;
                        break;
                    }
                case "Rejected":
                    {
                        order.OrderStatus = Status.Rejected;
                        break;
                    }
            }

            _db.Orders.Update(order);
            _db.SaveChanges();
            return Redirect("Home");
        }
    }
}
```

ApplicationDbContext.cs

```
using AskDekanat.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Text;

namespace AskDekanat.Data
{
    public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
        }

        public DbSet<Order> Orders { get; set; }

        protected override void OnModelCreating(ModelBuilder builder)
        {
            base.OnModelCreating(builder);
            builder.HasDefaultSchema("Identity");
            builder.Entity<ApplicationUser>(entity =>
            {
                entity.ToTable(name: "User");
            });
            builder.Entity<Order>(entity =>
            {
                entity.ToTable("Orders");
            });
            builder.Entity<IdentityRole>(entity =>
```

```
            {
                entity.ToTable(name: "Role");
            });
            builder.Entity<IdentityUserRole<string>>(entity =>
            {
                entity.ToTable("UserRoles");
            });
            builder.Entity<IdentityUserClaim<string>>(entity =>
            {
                entity.ToTable("UserClaims");
            });
            builder.Entity<IdentityUserLogin<string>>(entity =>
            {
                entity.ToTable("UserLogins");
            });
            builder.Entity<IdentityRoleClaim<string>>(entity =>
            {
                entity.ToTable("RoleClaims");
            });
            builder.Entity<IdentityUserToken<string>>(entity =>
            {
                entity.ToTable("UserTokens");
            });
        }
    }
}
```

DbInitializer.cs

```
using AskDekanat.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using System;
using System.Threading.Tasks;


namespace AskDekanat.Data
{
    public class DbInitializer
    {
        public static async Task Initialize(IServiceProvider serviceProvider)
        {
            var context = serviceProvider
                .GetRequiredService<ApplicationDbContext>();
            context.Database.EnsureCreated();
            var roleManager = serviceProvider
                .GetRequiredService<RoleManager<IdentityRole>>();
            var roleName = "Administrator";
            IdentityResult result;
            var roleExist = await roleManager.RoleExistsAsync(roleName);
            if (!roleExist)
            {
                result = await roleManager
                    .CreateAsync(new IdentityRole(roleName));
                if (result.Succeeded)
                {
                    var userManager = serviceProvider
                        .GetRequiredService<UserManager<ApplicationUser>>();
                    var config = serviceProvider
                        .GetRequiredService<IConfiguration>();
                    var admin = await userManager
```

```
                        .FindByEmailAsync(config["AdminCredentials:Email"]);

                    if (admin == null)
                    {
                        admin = new ApplicationUser()
                        {
                            UserName = config["AdminCredentials:Email"],
                            Email = config["AdminCredentials:Email"],
                            EmailConfirmed = true
                        };
                        result = await userManager
                            .CreateAsync(admin, config["AdminCredentials:Password"]);
                        if (result.Succeeded)
                        {
                            result = await userManager
                                .AddToRoleAsync(admin, roleName);
                        }
                    }
                }
            }
        }
    }
}
```

ApplicationUser.cs

```
using Microsoft.AspNetCore.Identity;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace AskDekanat.Models
{
    public class ApplicationUser : IdentityUser
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Group { get; set; }
        public string StudentId { get; set; }
    }
}
```

Order.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;

namespace AskDekanat.Models
{
    public enum Status
    {
        Created,
        In_progress,
        Rejected,
        Done
```

```
    }
    public class Order
    {
        public int OrderId { get; set; }
        public DateTime FromDate { get; set; }
        public DateTime ToDate { get; set; }
        public string Description { get; set; }
        public string StudentId { get; set; }
        public Status OrderStatus { get; set; }
    }
}
```

Home/Index.cshtml

```
@{
    ViewData["Title"] = "Home Page";
}
@using Microsoft.AspNetCore.Identity
@inject SignInManager<ApplicationUser> SignInManager
@inject UserManager<ApplicationUser> UserManager
@model IEnumerable<AskDekanat.Models.Order>


@{
    string isItem(Status status) {
        if (status.Equals(Status.Created))
            return "#c5e9fc";
        else if (status.Equals(Status.Done))
            return "#c5fcc5";
        else if (status.Equals(Status.In_progress))
            return "#fcf7c5";
        else if (status.Equals(Status.Rejected))
            return "#fcc5c5";
        else
            return "#c5e9fc";
    }
}

<div class="text-center">
    <h1 class="display-4">Welcome!</h1>
    @if (SignInManager.IsSignedIn(User))
            {
        @if (!User.IsInRole("Administrator"))
        {
        <div class="col-6">
            <a asp-controller="Order" asp-action="Create" class="btn btn-primary">Create
new order</a>
        </div>
        }
        @if (Model?.Count() > 0)
            {
            <p>List of orders</p>
            <table class="table table-bordered table-striped" style="width: 100%">
                <thead>
                    <tr>
                        <th>
                            Order id
                        </th>
                        <th>
                            Student id
                        </th>
                        <th>
```

```
                            From date
                        </th>
                        <th>
                            To date
                        </th>
                        <th>
                            Description
                        </th>
                    </tr>
                </thead>
                <tbody>
                    @foreach (var item in Model)
            {
                        <tr style="background-color: @isItem(item.OrderStatus)">
                            <td width="20%"><a
href="/Order/Index/@item.OrderId">@item.OrderId</a></td>
                            <td width="20%">@item.StudentId</td>
                            <td width="20%">@item.FromDate</td>
                            <td width="20%">@item.ToDate</td>
                            <td width="20%">@item.Description</td>
                            <td>
                                @Html.ActionLink("Delete", "Delete", new { id =
item.OrderId })
                            </td>
                        </tr>
                    }
                </tbody>
            </table>
        }
        else
        {
            <p>No orders created yet</p>
        }
    }
    else
    {
        <p>Thats application for managing students orders for recieving a
certificates.</p>
    }
</div>
```

Order/Create.cshtml

```
@model AskDekanat.Models.Order

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

<h1>Create</h1>

<h4>Order</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Create">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="FromDate" class="control-label"></label>
                <input asp-for="FromDate" class="form-control" />
                <span asp-validation-for="FromDate" class="text-danger"></span>
            </div>
```

```
            <div class="form-group">
                <label asp-for="ToDate" class="control-label"></label>
                <input asp-for="ToDate" class="form-control" />
                <span asp-validation-for="ToDate" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Description" class="control-label"></label>
                <input asp-for="Description" class="form-control" />
                <span asp-validation-for="Description" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>
@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

Order/Index.cshtml

```
@model AskDekanat.Models.Order

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

<h4>Order</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <p>Order: @Model.OrderId</p>
        <p>From: @Model.FromDate</p>
        <p>To: @Model.ToDate</p>
        <p>Student id: @Model.StudentId</p>
        <p>Description: @Model.Description</p>

        @if (User.IsInRole("Administrator"))
        {

    <div class="col-md-10">
        @using (Html.BeginForm("Post", "Order"))
        {
        @Html.DropDownListFor(x => x.OrderStatus, Html.GetEnumSelectList<Status>())
        @Html.ValidationMessageFor(model => model.OrderStatus)
        <input type="submit" value="Send" />
        }
    </div>
        }
        else
        {
            <p>Status: @Model.OrderStatus</p>
        }

    </div>
</div>
@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```