РОЗРОБКА ІНТЕРНЕТ ФОРУМУ

1.1. Опис алгоритму розробки.

Model-View-Controller (MVC, Модель-Вигляд-Контролер) – архітектурний патерн, який розділяє додаток на три основні групи компонентів: моделі, вигляди і контролери. Такий підхід дозваляє досягнути розділення проблем. Використовуючи цю модель, запити користувачів направляються до контролера, який відповідає за роботу з Моделлю для виконання дій користувача і / або отримання результатів запитів. Контроллер вибирає вью для відображення користувачеві, і зв’язує з будь-якою моделлю даних, яка вимагається. На рисунку 1.1. показані три основні компоненти, і як вони між собою пов’язані:

Рисунок 1.1. - Схема зв’язків MVC.

Таке розподілення обов’язків набагато спрощує розробку додатку, адже простіше писати код, налагоджувати або тестувати щось, що має одну конкретну роботу. Набагато складніше оновлювати, тестувати і налагоджувати код який виконує одразу кілька функцій. Наприклад, логіка інтерфейсу користувача, як правило, змінюється частіше, ніж бізнес-логіка. Якщо код користувацького інтерфейсу і бізнез-логіка об’єднані в одному об’єкті, то кожного разу, коли ви змінюєте користувацький інтерфейс вам доводиться змінювати об’єкт, який містить бізнес-логіку.

1.2. Розробка бази даних

Бази даних використовуються всюди, включаючи велику частину проектів в світі веб-розробки. Все, починаючи від найпростіших блогів і каталогів, до серйозних соціальних веб-проектів. Незалежно від складності сайту і відповідної бази даних, кожен з них вимагає ретельного проектування, щоб працювати ефективно, а також надійно.

База даних розреблена для студентського форуму містить вісім таблиць. Її діаграма показана на рисунку 1.2.



Рисунок 1.2. – Діаграма бази даних форуму

Далі будуть детальніше розглянуті таблиці бази даних.

* таблиця AspNetUsers містить інформацію про користувачів форуму, має відношення один до багатьох з таблицями: Comment, Post, AspNetLogins, AspNetUserClaims та AspNetUserRoles;
* таблиця Post містить інформацію про пости на сайті(назва, заголовок, автор, дата створення), має відношення один до багатьох з таблицею Comment;
* таблиця Comment відповідає за інформацію про коментарі до постів на форумі(автор, текст, дата створення);
* таблиця AspNetUserLogins в Asp.Net Identity, використовується для інформацію про вхід через інші сервіси, наприклад через facebook, google;
* таблиця AspNetUserClaims використовується для реалізації підходу авторизації IdentiyUserClaim;
* таблиці AspNetUserRoles, AspNetRoles, AspNetRoleClaims використовуються для реалізації IdentityRole підходу авторизації.

Для взаємодії з базою даних було використано Entity Framework підхід code first.

Всі таблиці пов’язанні з інформацією про користувачів створенні фреймворком ASP.NET автоматично. Для створення таблиць для постів та коментарів, використовувався підхід code first. Спочатку було створено моделі цих класів. Модель класу Post:

public class Post

 {

 public int PostID { get; set; }

 public string Title { get; set; }

 public string Text { get; set; }

 public string ApplicationUserId { get; set; }

 public DateTime Created { get; set; }

 public string Author { get; set; }

 public ICollection<Comment> Comments { get; set; }

 }

Поля класу відповідають таблицям в базі даних, а колекція Comments забезпечує відношення один до багатьох з таблицею Comment. Модель класу Comment:

public class Comment

{

 public int CommentID { get; set; }

 public string Text { get; set; }

 public string ApplicationUserId { get; set; }

 public int PostID { get; set; }

 public DateTime Created { get; set; }

 public string ApplicationUserName { get; set; }

}

Для зв’язку з базою даних було створенно клас ApplicationDbContext. Клас має наступний код:

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>

{

 public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

 : base(options)

 {

 }

 public DbSet<Post> Posts { get; set; }

 public DbSet<Comment> Comments { get; set; }

 protected override void OnModelCreating(ModelBuilder builder)

 {

 builder.Entity<Post>().HasKey(x => x.PostID);

 base.OnModelCreating(builder);

 builder.Entity<Post>().ToTable("Post");

 builder.Entity<Comment>().ToTable("Comment");

 }

}

Клас має дві властивості Posts i Comments, в яких зберігаються сутності збережених в базі даних таблицб Post та Comment відповідно. Інформація про користувачів створюється автоматично, адже клас наслідується від IdentityDbContext<ApplicationUser>. Метод OnModelCreating відповідає за зіставлення класів на мові C# з сутностями в базі даних.

Для ініціалізації та оновлення бази данних використовується такий механізм, як міграція бази даних. Міграції задаються у Package Manager Console. Щоб використувувати міграції, необхідно спочатку ввести команду enable-migrations. Для того щоб додати міграцію, необхідно написати add-migration та назву міграції, а після цього update-database. Після цього Visual Studio автоматично генерує клас міграції, який містить два методи Up та Down. У методі Up за допомогою виклику методу CreateTable створюються таблиці і проводяться їх налаштування: створення стовпців, установка ключів. І також за необхіднісю додабться нові стовпці в уже наявні таблиці. Метод Down видаляє стовпці і таблиці, якщо вони вже існують. Фактично ці методи рівнозначні висловом ALTER в мові SQL, яке змінює структуру бази даних і її таблиць.

1.3. Розробка Контролерів

В процесі розробки форуму було створено три контролери. Кожен з них відповідає за свою сутність та керування ними. Далі йде опис цих контролерів.

AccountController відповідає за авторизацію та реєстрацію користувачів. Для початку реєстрації або входу користувач повинен натиснути на кнопку Register або Log in. При реєстрації користувач спочатку вводить свої дані і при натисканні кнопку реєстрації, програма переходить до методу Register цього контролеру:

public async Task<IActionResult> Register(RegisterViewModel model, string returnUrl = null)

 {

 ViewData["ReturnUrl"] = returnUrl;

 if (ModelState.IsValid)

 {

 var user = new ApplicationUser { UserName = model.UserName, Email = model.Email };

 var result = await \_userManager.CreateAsync(user, model.Password);

 if (result.Succeeded)

 {

 await \_signInManager.SignInAsync(user, isPersistent: false);

 \_logger.LogInformation(3, "User created a new account with password.");

 return RedirectToLocal(returnUrl);

 }

 AddErrors(result);

 }

 return View(model);

}

Який спочатку перевіряє на валідність введені дані, а потім створює користувача, за допомогою методу CreateAsync об’єкту userManager, який відноситься до класу UserManager. Після цього за допомогою об’єкту signInManager класу SignInManager зареєстрований користувач набуває статусу увійшовшого, і відбувається перехід на головну сторінку файлу. При авторизації користувача, після вводу даних, програма переходить до методу Login, який і реалізує авторизацію коритстувача.

ManageController відповідає за керування обліковими записами. Тобто він відповідає за зміну інформації введеної при реєстрації(електронної пошти, ім’я користувача, паролю), додавання нової інформації(номеру мобільного телефону, додаткової електронної пошти), а також за підтвердження тієїж інформації(електронної пошти та номеру мобільного).

PostsController відповідає за ствроння, зміну та видалення постів та коментарів на форумі, а також пошук по темам постів. При створенні нового посту програма переходить до методу Create:

public async Task<IActionResult> Create(Post post)

 {

 var user = await \_userManager.GetUserAsync(HttpContext.User);

 post.ApplicationUserId = user.Id;

 if (ModelState.IsValid)

 {

 post.Author = user.UserName;

 post.Created = DateTime.Now;

 \_context.Add(post);

 await \_context.SaveChangesAsync();

 return RedirectToAction("Index");

 }

 return View(post);

}

Спочатку за допомогою userManager ми отримуємо поточного активного користувача для того, щоб задати автора посту. Потім перевіряємона валідність введені користувачем дані, якщо введено не коректні дані, завантажується сторінка створення знову, для повторного введення даних. Після цього додаємо створенний пост до об’єкту \_context, зберігаємо зміни та повертаємось на головну сторінку.

Метод Post відповідає за відображення певного посту, коментарів до цього посту та додавання нових коментарів до цього посту. Додавання нового коментарю до посту відбувається за допомогою наступного методу:

public async Task<IActionResult> Post(int? id, Comment comment)

 {

 var post = await \_context.Posts

 .SingleOrDefaultAsync(m => m.PostID == id);

 post.Comments = await \_context.Comments.ToListAsync();

 post.Comments = post.Comments.Where(c => c.PostID == post.PostID).ToList();

 if (!String.IsNullOrEmpty(comment.Text))

 {

 var user = await \_userManager.GetUserAsync(HttpContext.User);

 comment.ApplicationUserId = user.Id;

 comment.ApplicationUserName = user.UserName;

 comment.Created = DateTime.Now;

 comment.PostID = post.PostID;

 post.Comments.Add(comment);

 \_context.Add(comment);

 await \_context.SaveChangesAsync();

 comment.Text = "";

 }

 return RedirectToAction("Post");

}

Спочатку за допомогою об’єкту \_context ми дістаємо з списку постів поточний пост, за його id номером. Потім з того ж об’єкту \_context ми отримуємо список коментарів до поточного посту. Далі перевіряємо чи коректно введений текст коментарю. Якщо текст введено коректно, то ми додаємо коментар.

1.4. Розробка клієнтської частини форуму

Головна сторінка форуму показана на рисунку 1.3. На головній показаний список всіх тем форуму. Також є пошук по темам постів і кнопка для створення нового посту.



Рисунок 1.3. - Головна сторінка форуму.

Верхня панель сайту описана у файлі Layout.cshtml, в папці Views\Shared, і відображається на кожній сторінці сайту. Кнопки Student forum FAM, About, Contact додані за допомогою тегу <ul>, який в html відповідає за списки, для задання стилю цих кнопок використовувся bootstrap, а саме клас «collapse».

У правій частині верхньої панелі знаходяться кнопки Register та Log in. Вони описані в окрумому файлі, LoginPartial.cshtml, який також знаходиться у папці Views\Shared. Це зроблено для того, щоб можна було перевіряти чи є авторизований користувач, адже якщо є, то кнопку Register заміняє відання користувача, а замість кнопки Log in з’являється кнопка Log out, це можна побачити на рисунку 3.6. в правому верхньому куті.

Також у папці Shared знаходиться файл Error.cshtml, який відображається в разі виникнення помилок.

За опис головної сторінки відповідає файл Index.cshtml, який знаходиться в папці Views\Posts. Для пошуку потрібної теми використовується html тег <TextBox> та кнопка, кнопці задано bootstrap класи btn та btn-default. Кнопці Create New задано bootstrap класи btn btn-success. Для відображення всіх постів на форумі використовується таблиця. Таблиці задано bootstrap класи table table-striped. Таблиця складається з трьох стовбців:

* Title – заголовок посту;
* Author – автор посту;
* Created – дата створення.

Сторінка реєстрації зображена на рисунку 1.4.


Рисунок 1.4. – Сторінка реєстрації.

Файл Register.cshtml відповідає за сторінку реєстрації, знаходиться в папці Views\Account. Для вводу даних використовувався тег <input> та bootstrap клас form-control. Для кнопки Register тег <button> та bootstrap класи btn та btn-default. Для реєстрації користувач повинен ввести свою електронну пошту, ім’я користувача, пароль та підтвердження паролю. Всі поля проходять перевірку на валідність. Пошта повинна містити символ @. Пароль повинен бути не коротшим шести символів, складатись з латинських літер та містити хоча б одну цифру.

Сторінка авторизації зображена на рисунку 1.5.


Рисунок 1.5. – Сторінка авторизації.

За сторінку авторизації відповідає файл Login.cshtml, який теж знаходиться в папці Views\Account. Для введення даних і використовуються тіж теги та класи, що й для реєстрації. Для авторизації користувач повинен ввести свою електронну пошту та пароль.

Після того, як користувач зареєструвався або авторизувався він може створювати нові пости на форумі та коментувати вже створенні іншими користувачами.

Для створення нового посту необхідно натиснути кнопку Create New на головній сторінці(на рисунку 1.3. у правому верхньому куті). Сторінка створення нового посту показана на рисунку 1.6.


Рисунок 1.6. – Сторінка створення нового посту.

Файл Create.cshtml, який знаходиться в папці Views\Posts відповідає за сторінку створення нового посту. Для вводення заголовку та тексту посту використовуються html теги <TextBox> та <TextArea> відповідно.

Сторінка самого посту показана на рисунку 1.7. За сторінку посту відповідає файл Post.cshtml, який теж знаходиться в папці Views\Posts. Сторінка скаладається з заголовку посту, тексту та коментраів. Заголовок розташований зверху і записаний за допомогою html тегу <h1>. Далі йде текст посту, який відображається за допомогою тегу <h2>. Після цього вказана дату створення та автора за допомогою тегу <p>. Далі йде таблиця коментарів, яка створена за допомогою тегу <table> з bootstrap класом table-striped. І в кінці сторінки розміщений блок додавання нового коментарю, який складається з <TextArea> та кнопки, яка використовує bootstrap клас btn-primary.


Рисунок 1.7. – Сторінка посту.