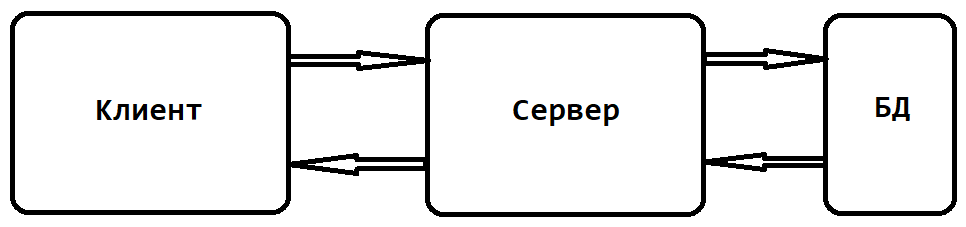
**3 Реализация программного продукта**

Реализация программного продукта – часть процесса, при которой программист непосредственно создает программный продукт.

Реализация данного программного продукта состоит из двух частей:

* реализация модуля сервера,
* реализация модуля клиента.

Архитектура программного продукта представлена на рисунке 3.1.



**Рисунок 3.1 – Архитектура программного продукта**

Взаимодействие серверного и клиентского модулей происходит с помощью программного интерфейса приложения (API). API требуется для того, чтобы серверный модуль мог обработать все запросы клиентского модуля перед непосредственным взаимодействием с базой данных

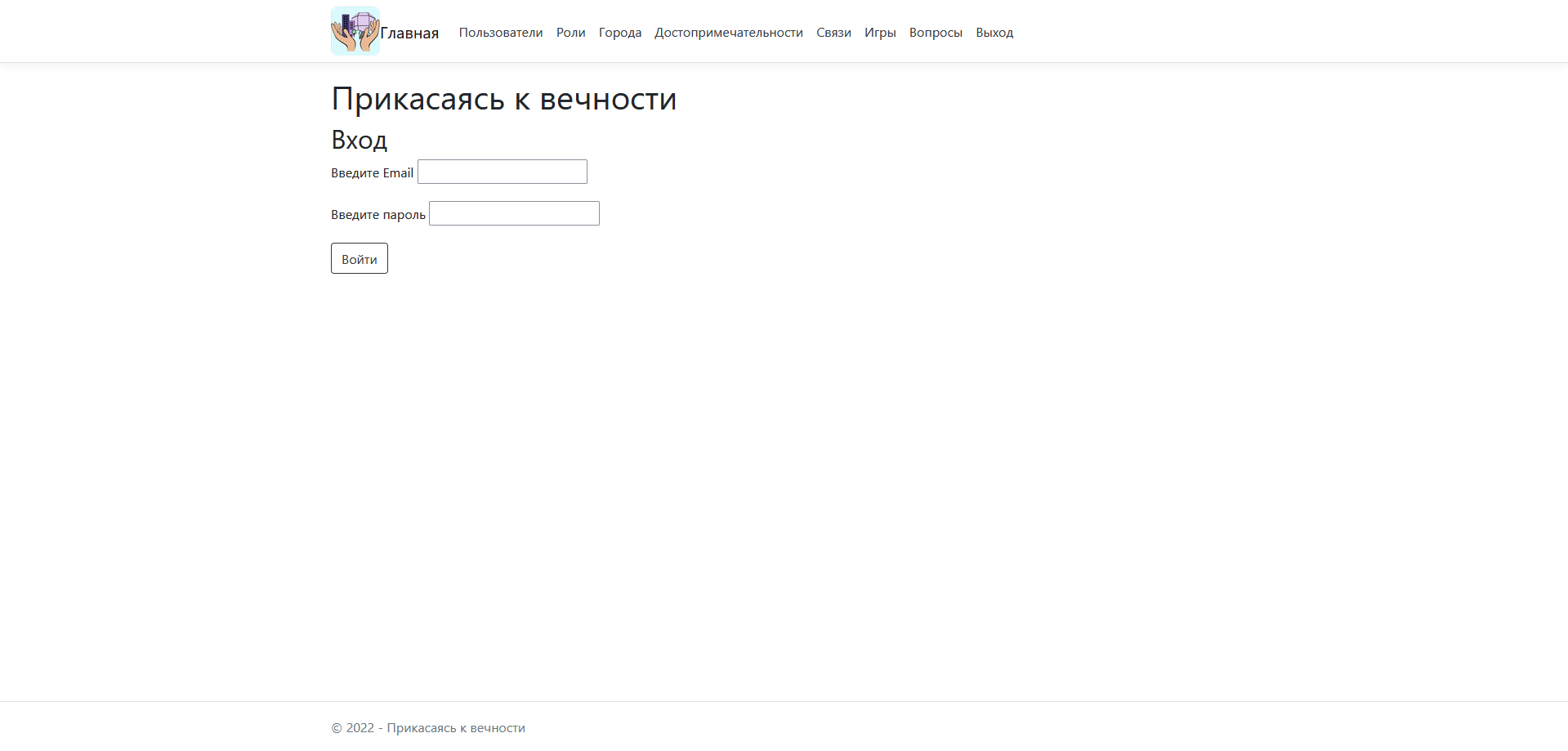
Взаимодействие серверной части с базой данных программного продукта реализована с помощью технологии «ADO.NET Entity Framework Core». Она представляет собой специальную объектно-ориентированную технологию на базе фреймворка «.NET» для работы с данными. Он представляет собой высокий уровень абстракции, который позволяет абстрагироваться от самой базы данных и работать с данными независимо от типа хранилища. На концептуальном уровне происходит работа с объектами.

При разработке серверного модуля используется уровень сервиса. Он инкапсулирует бизнес-логику и делает контроллеры довольно тонкими. В основном контроллеры используют сервисный уровень для получения моделей домена, которые затем преобразуются в модели просмотра. Также используется паттерн «Репозиторий», инкапсулирующий в себе всё, что относится к способу хранения данных, отделяющий бизнес-логику от деталей реализации слоя доступа к данным и упрощающий процесс модульного тестирования.

Рассмотрим подробнее реализацию каждого из модулей программного продукта.

**3.1 Серверный модуль**

При запуске административной панели администратор попадает на страницу авторизации. Вид страницы авторизации отражен на рисунке 3.2.



**Рисунок 3.2 – Страница авторизации**

На странице авторизации реализована система разграничения ролей, с помощью которой зайти в административную панель смогут только пользователи с ролью администратора. Код авторизации в панели администратора представлен в листинге 3.1.

**Листинг 3.1 – Авторизация в панели администратора**

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Login(LoginModel model)

{

if (ModelState.IsValid)

{

User user = await \_context.Users.FirstOrDefaultAsync(x => x.Email == model.Email && x.RoleId == 2);

if (user != null && AuthService.VerifyPassword(model.Password, user.PasswordHash, user.PasswordSalt))

{

await Authenticate(model.Email);

return RedirectToAction("Index", "Home");

}

ModelState.AddModelError("", "Некорректные логин и(или) пароль");

}

return View(model);

}

private async Task Authenticate(string email)

{

var claims = new List<Claim>

{

new Claim(ClaimsIdentity.DefaultNameClaimType, email)

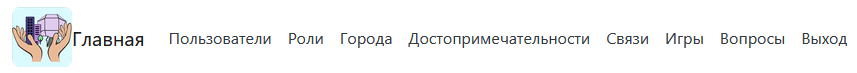
};

var id = new ClaimsIdentity(claims, "ApplicationCookie", ClaimsIdentity.DefaultNameClaimType, ClaimsIdentity.DefaultRoleClaimType);

await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new ClaimsPrincipal(id));

}

После прохождения процедуры авторизации администратору предоставляется возможность переходить на другие страницы, содержащие информацию, используя горизонтальное навигационное меню. Меню представлено на рисунке 3.3.



**Рисунок 3.3 – Навигационное меню**

Код отображения меню представлен в листинге 3.2.

**Листинг 3.2 – Отображение меню**

<nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">

<div class="container">

<**img** src="~/logo.svg" width="60" height="60" class="d-inline-block align-top">

<**a** class="navbar-brand" **asp-area**="" **asp-controller**="Home" **asp-action**="Index">Главная</**a**>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">

<ul class="navbar-nav flex-grow-1">

<li class="nav-item"><**a** class="nav-link text-dark" **asp-area**="" **asp-controller**="User" **asp-action**="Index">Пользователи</**a**></li>

<li class="nav-item"><**a** class="nav-link text-dark" **asp-area**="" **asp-controller**="Role" **asp-action**="Index">Роли</**a**></li>

<li class="nav-item"><**a** class="nav-link text-dark" **asp-area**="" **asp-controller**="City" **asp-action**="Index">Города</**a**></li>

<li class="nav-item"><**a** class="nav-link text-dark" **asp-area**="" **asp-controller**="Attraction" **asp-action**="Index">Достопримечательности</**a**></li>

<li class="nav-item"><**a** class="nav-link text-dark" **asp-area**="" **asp-controller**="DataReference" **asp-action**="Index">Связи</**a**></li>

<li class="nav-item"><**a** class="nav-link text-dark" **asp-area**="" **asp-controller**="GameScore" **asp-action**="Index">Игры</**a**></li>

<li class="nav-item"><**a** class="nav-link text-dark" **asp-area**="" **asp-controller**="TestQuestion" **asp-action**="Index">Вопросы</**a**></li>

<li class="nav-item"><**a** class="nav-link text-dark" **asp-area**="" **asp-controller**="Account" **asp-action**="Logout">Выход</**a**></li>

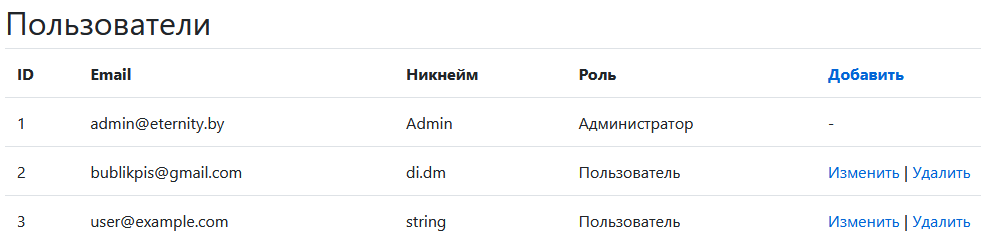
</ul>

</div>

</div>

</nav>

Рассмотрим подробнее страницу «Пользователи». Страница предполагает возможность просмотра, добавления, редактирования и удаления информации Вид страницы «Пользователи» представлен на рисунке 3.4.



**Рисунок 3.4 – Страница «Пользователи»**

Код отображения пользователей на странице отражен в листинге 3.3.

**Листинг 3.3 – Отображение пользователей на странице.**

public IActionResult Index()

{

ViewBag.Roles = \_context.Roles;

return View(\_repository.Get());

}

По нажатию кнопки «Добавить» открывается страница добавления нового пользователя. Вид страницы отображен на рисунке 3.5.



**Рисунок 3.5 – Добавление нового пользователя**

Код добавление нового пользователя представлен в листинге 3.4.

**Листинг 3.4 – Добавление нового пользователя**

[HttpPost]

public IActionResult Create(User newUser, IFormFile uploadedFile)

{

ViewBag.Roles = \_context.Roles;

if (\_context.Users.FirstOrDefault(x => x.Email == newUser.Email) != null) {

ModelState.AddModelError("Email", "Email уже используется");

}

if (\_context.Users.FirstOrDefault(x => x.UserName == newUser.UserName) != null) {

ModelState.AddModelError("Username", "Никнейм уже используется");

}

if (ModelState.IsValid) {

var user = new User {

UserId = null, Email = newUser.Email, UserName = newUser.UserName, Password = newUser.Password, RoleId = newUser.RoleId };

\_repository.Create(user, uploadedFile);

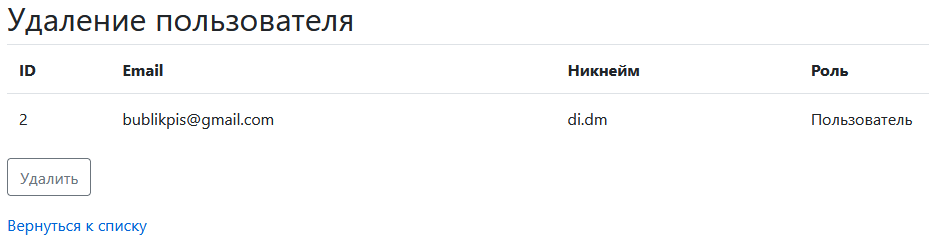
return RedirectToAction("Index");

}

return View(newUser);

}

После нажатия на кнопку «Изменить» открывается страница изменения существующего пользователя, реализованная схожим со страницей добавления пользователя способом. По нажатию на кнопку «Удалить» открывается страница подтверждения удаления пользователя, изображенная на рисунке 3.6.



**Рисунок 3.6 – Подтверждение удаления пользователя**

Код подтверждения удаления и удаления пользователя представлен в листинге 3.5.

**Листинг 3.5 – Подтверждение удаления и удаление пользователя**

[HttpGet]

[ActionName("Delete")]

public ActionResult ConfirmDelete(int id)

{

if (id == 1) { return RedirectToAction("Index"); }

User user = \_repository.Get(id);

if (user != null) {

ViewData["RoleName"] = \_context.Roles.First(x => x.RoleId == user.RoleId).Name;

return View(user);

}

return NotFound();

}

[HttpPost]

public ActionResult Delete(int id)

{

\_repository.Delete(id);

return RedirectToAction("Index");

}

Другие информационные страницы реализованы схожим образом. Таким образом, была проведена реализация административной панели.

Для получения данных из клиентской части используется API. Код метода API, отправляющего список городов представлен в листинге 3.6.

**Листинг 3.6 – Получение списка городов**

// Получает список городов

[HttpGet]

public async Task<IEnumerable<CityViewModel>> Get()

{

IEnumerable<CityViewModel> result = new List<CityViewModel>();

IEnumerable<City> cities = await \_context.Cities.ToListAsync();

foreach (var item in cities)

{

result = result.Append(new CityViewModel

{

CityId = item.CityId,

Title = item.Title,

Description = item.Description,

References = await \_context.DataReferences.Where(x => x.CityId == item.CityId).Select(x => x.AttractionId).ToListAsync()

});

}

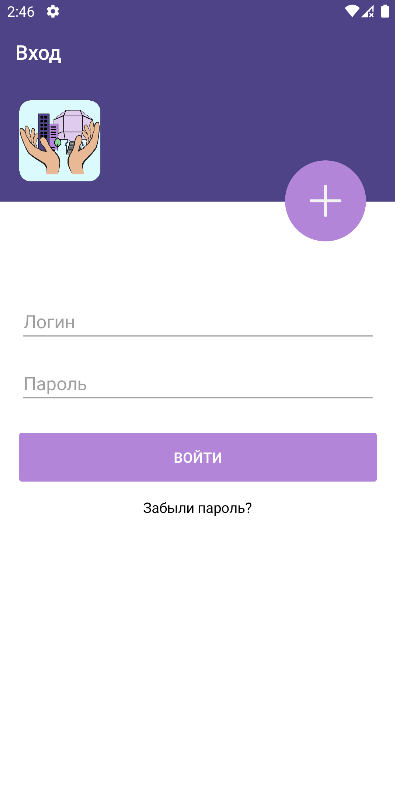
return result;

}

Другие методы получения информации реализованы схожим образом. Таким образом, были реализованы методы API для получения информации. Реализация серверного модуля прошла успешно.

**3.2 Клиентский модуль**

При запуске клиентского мобильного приложения, пользователь попадает в окно авторизации. Вид окна авторизации отображен на рисунке 3.7.



**Рисунок 3.7 – Окно авторизации**

После ввода данных в поля «Логин» и «Пароль» и нажатия кнопки «Войти» происходит передача введенных данных на сервер посредством API. На сервере данные проходят проверку, после чего клиентское приложение получает ответ. Код процедуры авторизации представлен в листинге 3.7.

**Листинг 3.7 – Процедура авторизации**

private async void LoginButton\_Clicked(object sender, EventArgs e) {

var userService = new UserService();

var imageService = new ImageService();

if (Username.Text != null && Password.Text != null) {

User currentUser = await userService.Get(Username.Text, Password.Text);

await SecureStorage.SetAsync("ID", currentUser.UserId.ToString());

await SecureStorage.SetAsync("Username", currentUser.UserName);

(Application.Current.MainPage as AppShell).ViewModel.Username = currentUser.UserName;

try {

await SecureStorage.SetAsync("ImageUri", $"{AppSettings.Url}images/users/{(int)currentUser.UserId}/{await imageService.GetTitleImage("users", (int)currentUser.UserId)}");

(Application.Current.MainPage as AppShell).ViewModel.ImageSource = await SecureStorage.GetAsync("ImageUri"); }

catch {

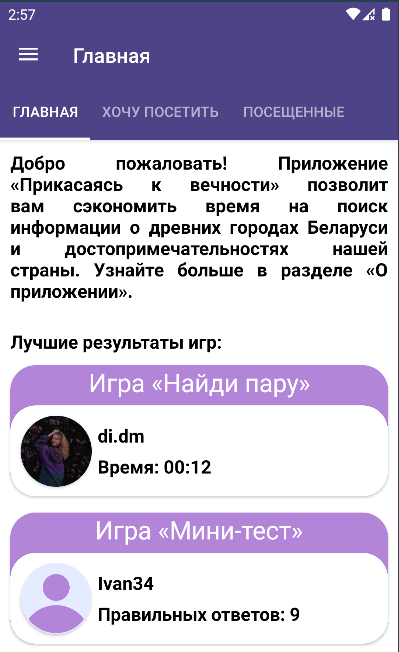
(Application.Current.MainPage as AppShell).ViewModel.ImageSource = "icon\_no\_avatar.png"; }

await Shell.Current.GoToAsync("//MainPage");

}

}

После прохождения процедур авторизации или регистрации открывается окно «Главная», представленное на рисунке 3.9.



**Рисунок 3.9 – Окно «Новости»**

Код отображения лучших результатов игр представлен в листинге 3.8.

**Листинг 3.8 – Отображение лучших результатов игр**

private async Task LoadData() {

try {

Game1Result.IsVisible = true;

Game1NoResult.IsVisible = false;

game1Score = await \_gameScoreService.Get(1);

Game1UserName.Text = game1Score.UserName;

Game1UserScore.Text = $"Время: {game1Score.Score}";

if (game1Score.Image == null) {

Game1UserImage.Source = "icon\_no\_avatar.jpg";

}

else {

Game1UserImage.Source = $"{AppSettings.Url}images/Users/{game1Score.UserId}/{game1Score.Image}"; }

}

catch {

Game1Result.IsVisible = false;

Game1NoResult.IsVisible = true;

}

try {

Game2Result.IsVisible = true;

Game2NoResult.IsVisible = false;

game2Score = await \_gameScoreService.Get(2);

Game2UserName.Text = game2Score.UserName;

Game2UserScore.Text = $"Правильных ответов: {game2Score.Score}";

if (game2Score.Image == null) {

Game2UserImage.Source = "icon\_no\_avatar.jpg";

}

else {

Game2UserImage.Source = $"{AppSettings.Url}images/Users/{game2Score.UserId}/{game2Score.Image}";

}

}

catch {

Game2Result.IsVisible = false;

Game2NoResult.IsVisible = true;

}

}

Другие окна мобильного приложения. Реализация клиентского модуля прошла успешно.