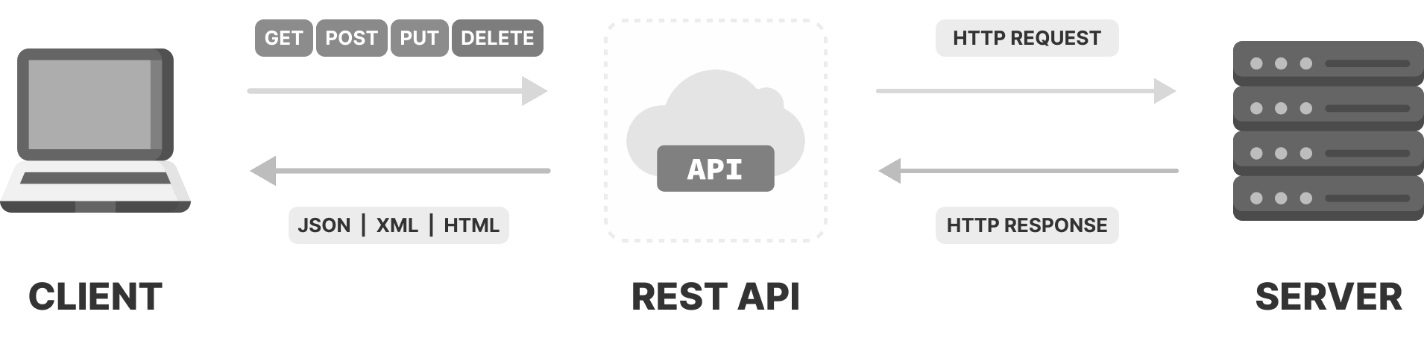
**3 Реализация программного продукта**

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.

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 Структура взаимодействия клиентской и серверной частей представлена на рисунке 3.1.

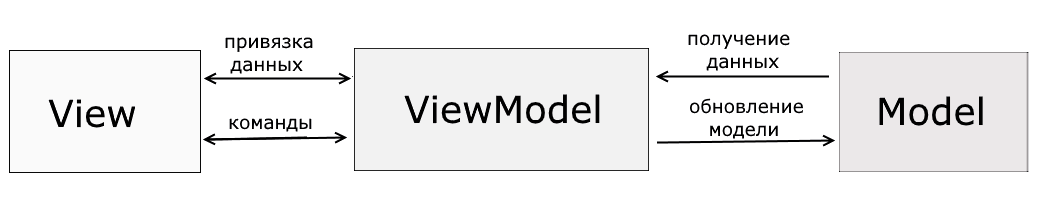


**Рисунок 3.1 – Взаимодействие клиентской и серверной частей**

Взаимодействие серверной части с базой данных программного продукта реализована с помощью технологии «ADO.NET Entity Framework Core». «Entity Framework» представляет 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.

При разработке клиентского модуля используется архитектурный паттерн «MVVM» [13]. Он позволяет отделить логику приложения от визуальной части (представления). Схема взаимодействия компонентов клиентского модуля представлена на рисунке 3.2.

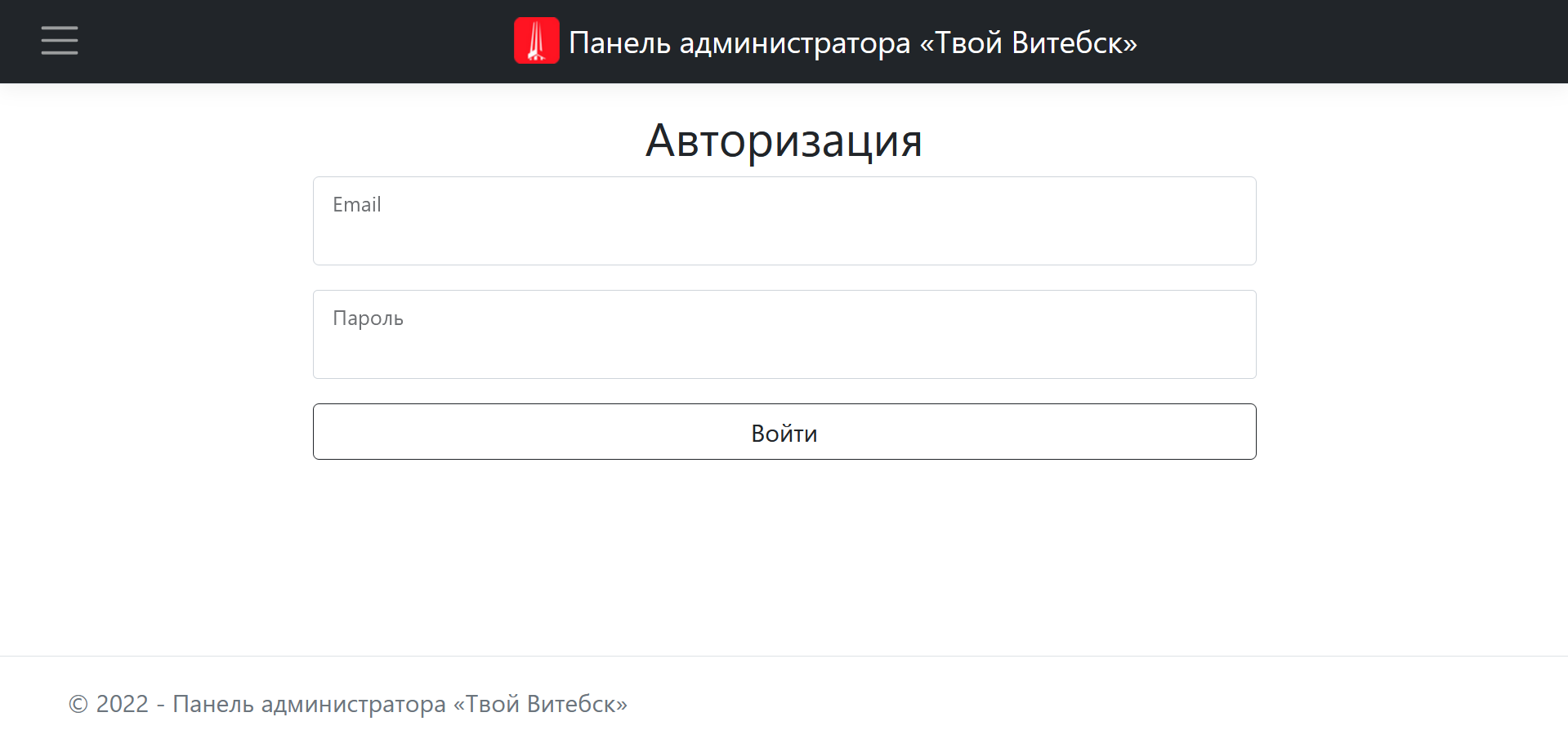


**Рисунок 3.2 – Взаимодействие компонентов клиентского модуля**

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 3.3.



**Рисунок 3.3 – 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);

if (user != null && await \_context.RolePermissionLinks.AnyAsync(x => x.RoleId == user.RoleId) && 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)

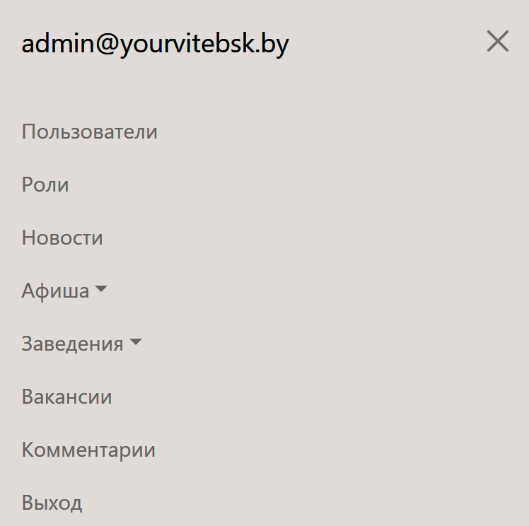
};

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

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

}

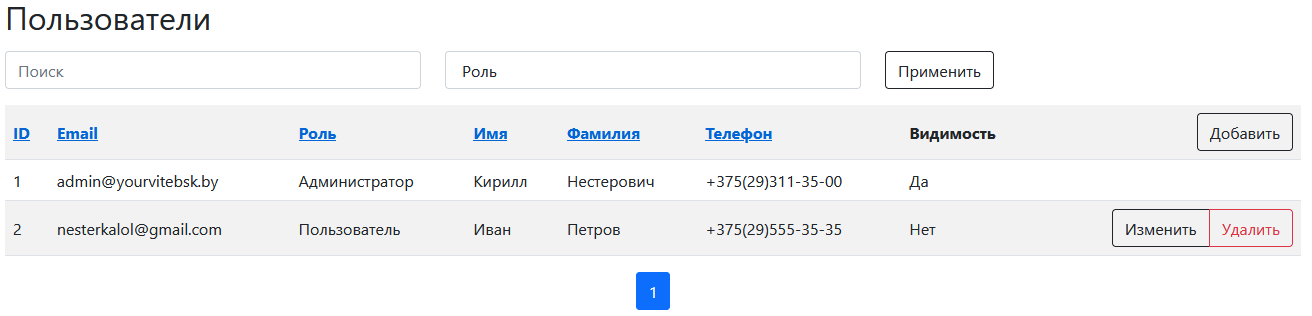
После прохождения процедуры авторизации администратору предоставляется возможность переходить на другие страницы, содержащие информацию, используя вертикальное выпадающее навигационное меню. Меню представлено на рисунке 3.4.



**Рисунок 3.4 – Навигационное меню**

Код отображения меню представлен в листинге В.1 приложения В.

Рассмотрим подробнее страницу «Пользователи». Страница предполагает возможность просмотра, добавления, редактирования, удаления информации, а также сортировки, поиска данных и фильтрации по ролям пользователей. Вид страницы «Пользователи» представлен на рисунке 3.5.



**Рисунок 3.5 – Страница «Пользователи»**

Код отображения пользователей на странице отражен в листинге 3.2.

**Листинг 3.2 – Отображение пользователей на странице.**

public IEnumerable<IViewModel> Get()

{

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

IEnumerable<User> users = \_context.Users.ToList().OrderBy(x => x.UserId);

foreach (User user in users)

{

result = result.Append(new UserViewModel()

{

UserId = (int)user.UserId, Email = user.Email, RoleId = user.RoleId, Role = \_context.Roles.First(x => x.RoleId == user.RoleId).Name, IsVisible = user.IsVisible, FirstName = user.FirstName, LastName = user.LastName, PhoneNumber = user.PhoneNumber

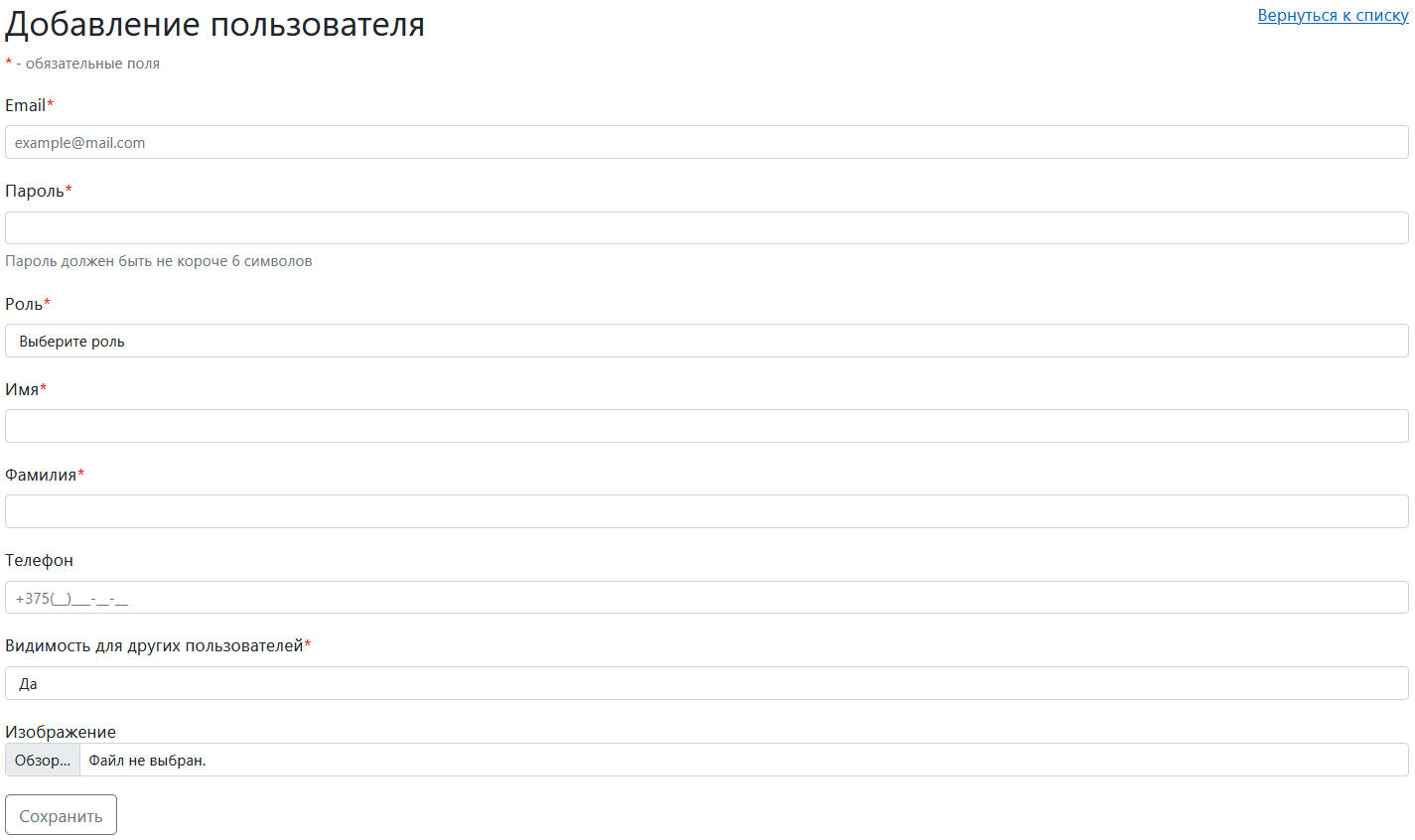
});

}

return result;

}

По нажатию кнопки «Добавить» открывается страница добавления нового пользователя. Вид страницы отображен на рисунке 3.6.



**Рисунок 3.6 – Добавление нового пользователя**

Код добавление нового пользователя представлен в листинге 3.3.

**Листинг 3.3 – Добавление нового пользователя**

[HttpPost]

public ActionResult Create(UserViewModel newUser, IFormFileCollection uploadedFiles) {

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

{

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

}

if (!string.IsNullOrEmpty(newUser.PhoneNumber))

{

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

{

ModelState.AddModelError("PhoneNumber", "Такой номер телефона уже используется");

}

}

if (ModelState.IsValid)

{

\_repository.Create(newUser, uploadedFiles);

return RedirectToAction("Index");

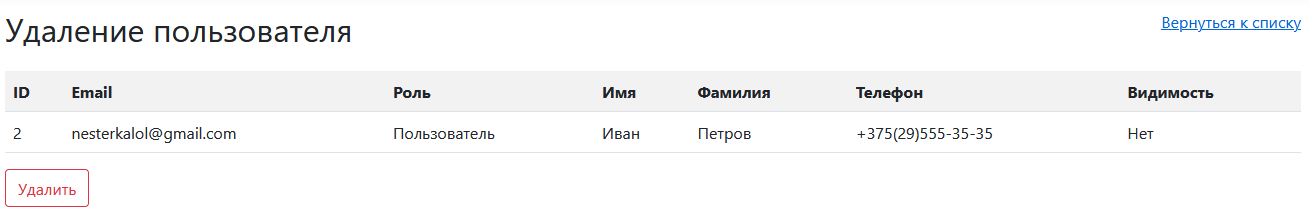
}

ViewBag.Roles = \_context.Roles;

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 3.7.



**Рисунок 3.7 – Подтверждение удаления пользователя**

Код подтверждения удаления и удаления пользователя представлен в листинге 3.4.

**Листинг 3.4 – Подтверждение удаления и удаление пользователя**

[HttpGet] // Подтверждение удаления пользователя

[ActionName("Delete")]

public ActionResult ConfirmDelete(int id)

{

if (id == 1) {

return RedirectToAction("Index");

}

UserViewModel user = (UserViewModel)\_repository.Get(id);

if (user != null) { return View(user); }

return NotFound();

}

[HttpPost] // Удаление пользователя

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 использования API пользователю системы необходимо пройти процедуру авторизации или регистрации, чтобы получить токен доступа [14]. Токен позволяет пользователям взаимодействовать с методами API и получать или изменять информацию. Код класса, отвечающего за выдачу токена представлен в листинге В.2 приложения В.

Код метода API, отправляющего новостные статьи представлен в листинге 3.5.

**Листинг 3.5 – Получение новостных статей**

// Gets all news

[HttpGet("all")]

public async Task<IEnumerable<News>> GetAll(int offset, int count)

{

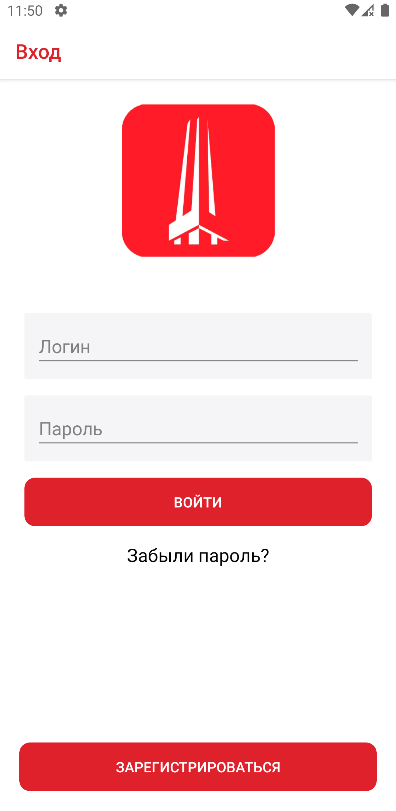
return (await \_newsService.GetAll()).Skip(offset).Take(count);

}

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 3.8.



**Рисунок 3.8 – 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 3.6.

**Листинг 3.6 – Процедура авторизации**

private async Task LogIn()

{

if (Connectivity.NetworkAccess == NetworkAccess.Internet)

{

IsBusy = true;

try

{

string token = await \_authService.Login(new UserLoginDTO

{

Email = Email, Password = Password,

});

\_authService.SaveUserCreds(token);

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

IsBusy = false;

}

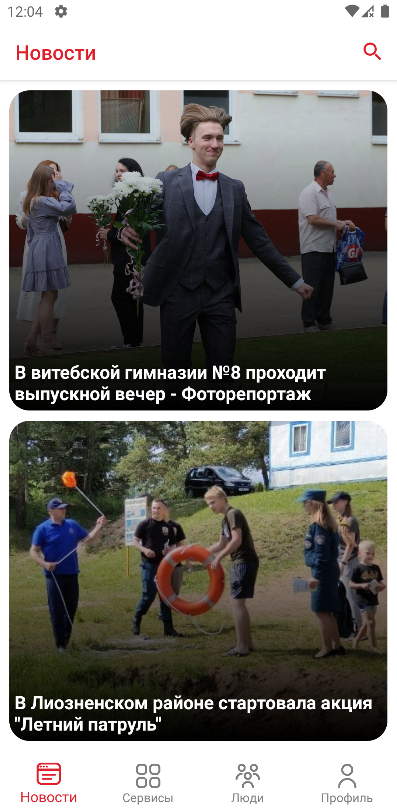
catch (ArgumentException e) { Error = e.Message; }

IsBusy = false;

}

}

После прохождения процедур авторизации или регистрации открывается доступ к основной части приложения, содержащей навигационное меню, позволяющее переходить к другим информационным окнам приложения. Первым открывается окно «Новости», представленное на рисунке 3.9.



**Рисунок 3.9 – Окно «Новости»**

Код отображения новостей представлен в листинге 3.7.

**Листинг 3.7 – Отображение новостей**

// Получаем список новостей

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

{

string response = await \_client.GetStringAsync(\_url + "all");

var result = JsonSerializer.Deserialize<IEnumerable<News>>(response, \_options);

foreach (var item in result)

{

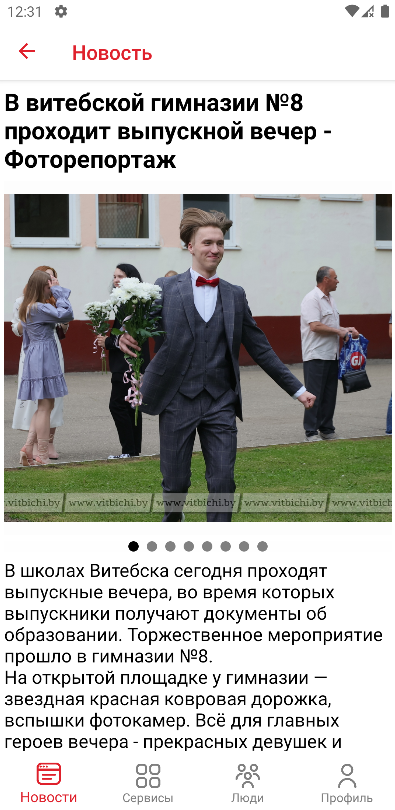
item.TitleImage = $"{AppSettings.BaseApiUrl}/images/News/{item.NewsId}/{item.TitleImage}";

}

return result;

}

При нажатии на изображение новости, происходит переход к окну, подробно отображающему эту новость. Вид окна конкретной новости представлен на рисунке 3.10.



**Рисунок 3.10 – Окно конкретной новости**

Код отображения окна представлен в листинге 3.8.

**Листинг 3.8 – Отображение окна конкретной новости**

<StackLayout IsVisible="{Binding IsMainLayoutVisible}" AbsoluteLayout.LayoutBounds="0,0,1,1" AbsoluteLayout.LayoutFlags="All" VerticalOptions="Fill" HorizontalOptions="Fill">

<ScrollView>

<StackLayout Padding="0">

<Label Text="{Binding News.Title}" FontSize="Title" FontAttributes="Bold" />

<CarouselView ItemsSource="{Binding News.Images}" HeightRequest="350" IndicatorView="indicatorview" >

<CarouselView.ItemTemplate>

<DataTemplate x:DataType="x:String">

<ffimageloading:CachedImage x:Name="Image" Source="{Binding .}" HorizontalOptions="Center" VerticalOptions="Center" />

</DataTemplate>

</CarouselView.ItemTemplate>

</CarouselView>

<IndicatorView x:Name="indicatorview" IndicatorColor="Gray" IndicatorSize="10" SelectedIndicatorColor="Black" />

<Label Text="{Binding News.Description}" />

<BoxView Color="{AppThemeBinding Dark={StaticResource TextColorDark}, Light={StaticResource TextColorLight}}" HeightRequest="1" HorizontalOptions="Fill" Margin="20" />

<Label Text="Источник" TextColor="{AppThemeBinding Dark={StaticResource AccentColorDark}, Light={StaticResource AccentColorLight}}" TextDecorations="Underline">

<Label.GestureRecognizers>

<TapGestureRecognizer Command="{Binding TapCommand}" CommandParameter="{Binding News.ExternalLink}" />

</Label.GestureRecognizers> </Label>

<BoxView Margin="0,0,0,50" />

</StackLayout>

</ScrollView>

</StackLayout>

Также на странице «Новости» реализован поиск, с помощью которого можно перейти к окну конкретной новости. Код класса, отвечающего за поиск по новостям представлен в листинге 3.9.

**Листинг 3.9 – Поиск по новостям**

using System.Collections.Generic;

using System.Linq;

using Xamarin.Forms;

using YourVitebskApp.Models;

using YourVitebskApp.Views;

namespace YourVitebskApp.Controls

{

public class NewsSearchHandler : SearchHandler

{

protected override void OnQueryChanged(string oldValue, string newValue)

{

base.OnQueryChanged(oldValue, newValue);

if (string.IsNullOrWhiteSpace(newValue)) { ItemsSource = null; }

else

{

ItemsSource = (BindingContext as IEnumerable<News>).Where(x => x.Title.ToLower().Contains(newValue.ToLower())).ToList();

}

}

protected override async void OnItemSelected(object item)

{

base.OnItemSelected(item);

await Shell.Current.GoToAsync($"{nameof(SpecificNewsPage)}?NewsId={((News)item).NewsId}");

}

protected override void OnUnfocus() { base.OnUnfocus(); }

}

}

Другие окна мобильного приложения и системы поиска реализованы схожим образом. Реализация клиентского модуля прошла успешно.