МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение высшего образования «БЕЛОРУССКИЙ

ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-40 05 01 Информационные системы и технологии

Направление специальности 1-40 05 01-03 Информационные системы и

технологии (издательско-полиграфический комплекс)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОЙ РАБОТЫ**

по дисциплине «Организация веб-порталов и администрирование информационных систем».

Тема: «Администрирование интернет-приложения для знакомств»

Исполнитель

студент 4 курса 1 группы А.А. Иванова

(подпись) (инициалы, фамилия)

Руководитель

ассистент А.Д. Томко

(подпись) (инициалы, фамилия)

Курсовая работа защищена с оценкой

Минск 2022

**Оглавление**

[Введение 3](#_Toc121812416)

[1. Аналитический обзор технологий проекта 4](#_Toc121812417)

[1.1. Этап разработки веб-приложения 5](#_Toc121812418)

[1.2. Администрирование проекта 17](#_Toc121812419)

[1.3. Начало работы 20](#_Toc121812420)

[2. Проверка работоспособности приложения 21](#_Toc121812421)

[2.1. Пользователь 22](#_Toc121812422)

[2.2. Модератор 22](#_Toc121812423)

[2.3. Администратор 23](#_Toc121812424)

[Заключение 24](#_Toc121812425)

[Список литературы 25](#_Toc121812426)

# Введение

В наше время знакомство в интернете перестало быть чем-то необычным или пугающим. Несмотря на то, что индустрия знакомств активно набирает обороты и привлекает все больше людей, период работы из дома внес свои коррективы. Все больше пользователей стало регистрироваться в приложениях для знакомств. Найти себе пару или хотя бы собеседника на вечер не составит труда именно благодаря сервису или приложению, созданного для общения и дружбы. Некоторые приложения даже могут помочь выучить иностранный язык благодаря международному поиску людей и общению (иногда, такие приложения имеют встроенный словарь).

Каждый человек максимально заинтересован в том, чтобы найти интересного, и самое главное безопасного, собеседника. Для этого он максимально ёмко заполняет информацию о себе, выставляет свои фотографии. Необходимо создать максимально безопасное приложение с оперативной техподдержкой клиентов, в котором будет удобно знакомиться и общаться с другими пользователями, также, стоит не забывать об удобстве критерий сортировки и фильтрации пользователей.

Web-приложение — это полноценная программа, доступ к которой пользователь получает через интернет, то есть она не требует установки на устройство, что делает использование программы удобным и простым, так как пользователю нужен лишь браузер. Оно интерактивно и позволяет пользователям взаимодействовать с разными элементами: например, общение с другими пользователями, просмотр фотографий этих пользователей, создание жалобы на страницу любого пользователя.

В качестве языка разработки проекта был выбран C#, платформа ASP.NET Core[1]. Web-приложение написано с применением Angular, также оно является логически завершенным. Развёртывание конечного приложения для последующего использования реализовано на Heroku. В качестве хранилища данных используется PostgreSQL[2].

Тема курсового проекта – администрирование web-приложения для знакомств.

Задачей моего курсового проекта является создание приложения, которое будет позволять людям находить друзей, знакомства, общение по всему миру. Пользователю не нужно будет просматривать каждую анкету в поисках необходимого пользователя по параметрам, система должна позволить клиенту самому настроить необходимые параметры, чтобы выдать правильную подборку людей.

# 1. Аналитический обзор технологий проекта

При разработке проекта, первым делом стоит провести анализ аналогов, чтобы иметь представление о требуемых функциях и разделений по ролям в проекте.

После проведения анализа аналогов, можно сделать вывод, что функционально веб-приложение для знакомств должно:

1. поддерживать роли администратора, модератора и пользователя;
2. позволять регистрироваться и авторизовываться в приложении;
3. позволять администратору, модератору и пользователю:
   1. добавление пользователей в группу избранных;
   2. загружать фотографии на свою страницу;
   3. общение пользователей посредством сообщений в чате;
   4. сортировать и фильтровать пользователей по критериям;
   5. отправлять заявку на блокировку пользователя в приложении;
   6. изменение и просмотр информации о пользователе;
4. позволять администратору и модератору:
   1. подтверждать или удалять загруженные пользователями фотографии;
5. позволять администратору:
   1. изменять роли пользователей;
   2. подтверждать или отклонять заявки на блокировку пользователей;
   3. авторизовываться в приложении.

Что бы корректно произвести обзор веб-приложения для знакомств, как со стороны администрирования, так и разработки, будет реализована следующая структура пояснительной записки:

* этап разработки приложения;
* администрирование проекта;
* установка и начало работы с приложением.

1.1. Этап разработки веб-приложения

Для сохранения данных, с которыми работает веб-приложение, необходимо определиться с хранилищем. Для этого идеально подойдут реляционные СУБД, которое позволяют оперативно сохранять и обрабатывать данные разных типов. Мой взгляд упал на СУБД PostgreSQL. Она является кроссплатформенной и довольно быстрой, а также простой в использовании. Общая структура базы данных представлена на рисунке 1.1 ниже.

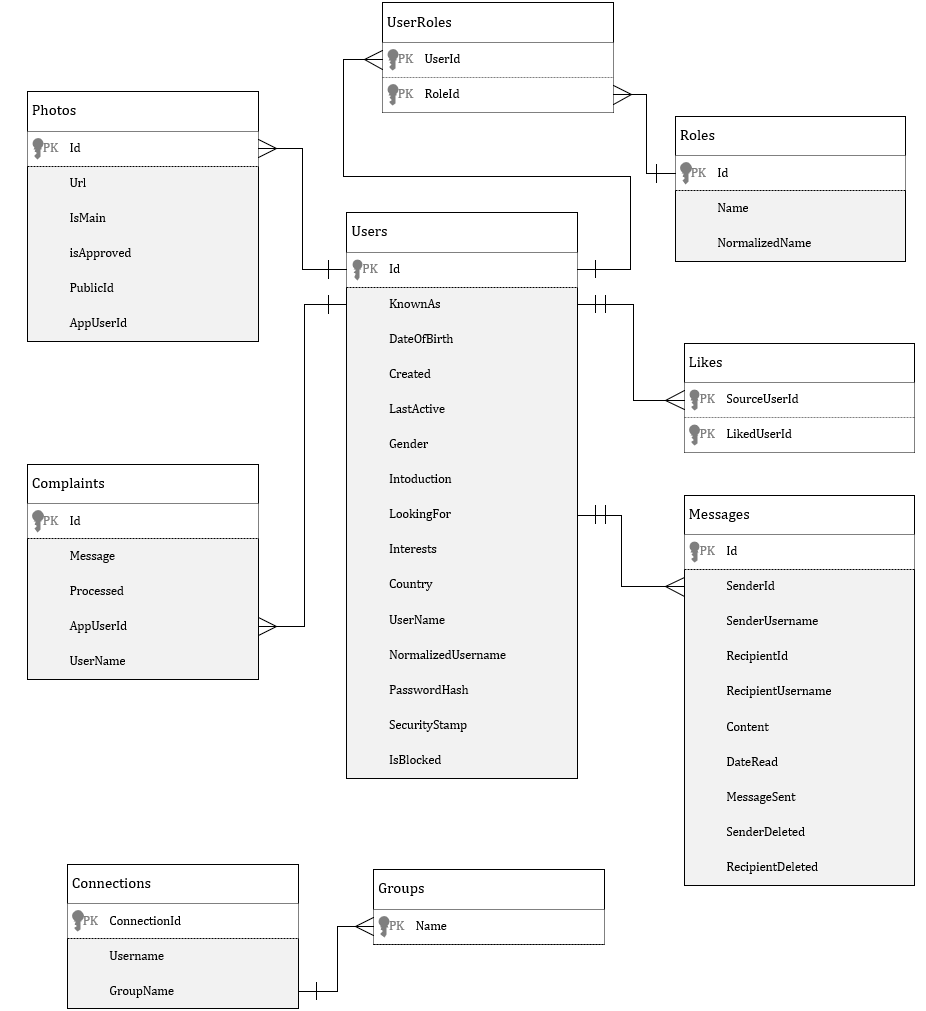


Рисунок 1.1 – Схема связей таблиц базы данных

В базе данных реализованы следующие связи:

* «Users – UserRoles»: один ко многим;
* «Roles – UserRoles»: один ко многим;
* «Users – Likes»: многие ко многим;
* «Users – Messages»: многие ко многим;
* «Users – Photos»: один ко многим;
* «Users – Complaints»: один ко многим;
* «Connections - Groups»: один ко многим.

Следующим шагом является разработка серверной части. Для реализации серверной части использовалась кроссплатформенная технология ASP.NET Core. Язык программирования С#. Первым делом был создан проект типа «Web Api», который предназначен для создания серверной части.

Для реализации серверной части использовалась кроссплатформенная технология ASP.NET Core. Язык программирования C#, в частности использовалась версия языка «.Net 6». Первым делом был создаем проект типа «web-api», который предназначен для создания серверной части. Структура проекта состоит из папок «Controllers», «Data», внутри которой лежит папка «Migrations», «DTOs», «Entities», «Errors», «Extentions», «Helpers», «Interfaces», «Middleeware», «Properties», «Services», «SignalR», «wwwroot» изображенных на рисунке 1.2.

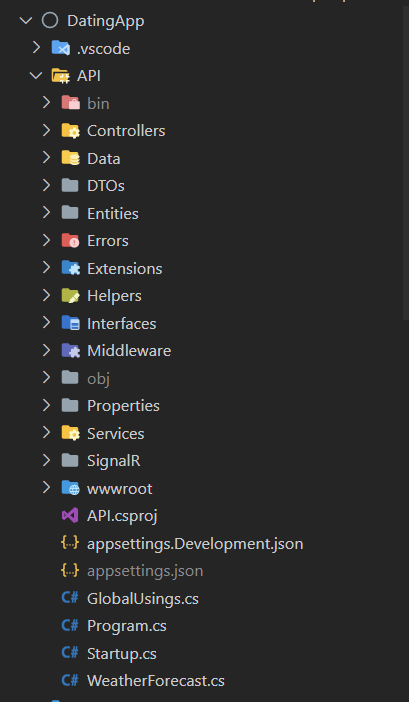


Рисунок 1.2 – Структура сервера web-api

Папка «wwwroot» хранит в себе сборку файлов пользовательского интерфейса. Он является отдельной программой, сборка которой идет в эту папку.

Папка «Controllers» содержит в себе все контроллеры, которые обрабатывают REST запросы из пользовательского интерфейса. На рисунке 1.3 изображены все классы контроллеров, которые используются в приложении.

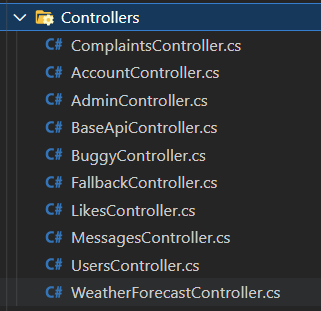


Рисунок 1.3 – Контроллеры, которые используются в проекте

Папка «Controllers» предназначена для обработки запросов от клиента и работы с базой данных. В ней находятся контроллеры для работы с каждой сущностью. Одним из условий проекта было построения приложения, который работает с применением стилевого архитектурного паттерна REST.

REST-архитектура предполагает применение следующих методов или типов запросов HTTP для взаимодействия с сервером:

* get;
* post;
* put;
* delete.

Папка «Data» хранит в себе классы, которые отвечают за связь с базой данных. В данном проекте использует подход «code first» при котором сначала создается код проекта, а после под этот код создается база данных с помощью entity framework [4]. Для того, чтобы реализовать подход «code first» необходимо создавать миграции с помощью консольных команд. В папке «Migrations» находятся все миграционные данные для создания или редактирования базы данных.

Также в папке «Data» находятся класс «Seed.cs» и файл «UserSeedData.json» эти файлы отвечают за вставку первичных данных в базу данных.

Папка «DTOs» хранит в себе все классы, сущности которых используются в пользовательском интерфейсе. Эти классы являются аналогом классов «ViewModel» из архитектурного паттерна «MVVM».

Папка «Entities» содержит все классы сущностей серверного приложения. Тут содержатся классы, по которым создается база данных.

Папка «Middleware» содержит в себе класс, основная задача которого обрабатывать все ошибки приложения, которые хранятся в формате JSON. Эти ошибки, необходимо форматировать для того, чтобы вернуть пользователю в понятном ему виде. Для этого в папке «Errors» находится класс, который содержит в себе 3 поля: статус ошибки, код ошибки и детали ошибки. Объект именно этого класса возвращают все методы папки «Middleware».

Папка «Extensions» хранит в себе все классы расширения, например: класс для внедрения зависимостей, регистрацию сервиса «Cloudinary», настройки использования определенной базы данных в зависимости от режима проекта: «Development» или «Realese» (листинг переключения контекста баз данных находится в приложении В) , настройки заголовков для ответов пользовательскому интерфейсу и тому подобное. На рисунке 1.4 представлен состав папки «Extrnsions».

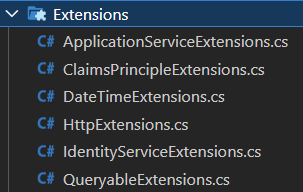


Рисунок 1.4 – Состав папки «Extensions»

Папка «Helpers» хранит в себе все классы, сущности которых являются помощниками, то есть они не используются в ответах или редактировании базы данных, однако используются при обработке запросов и ответов или в поддержании сервера в рабочем состоянии.

Папка «Interfaces» содержит все интерфейсы проекта, в том числе и те, которые поддерживают внедрение зависимостей и паттерн «Unit of work».

Папка «Services» содержит в себе классы «PhotoService» и «TokenService». Методы первого класса работают с сервисом «Cloudinary», который хранит в облачном хранилище все фотографии приложения. Второй класс содержит один метод, который создает токен каждому пользователю. Который является валидным в течение 7 дней. Токен необходим для взаимодействия пользовательского интерфейса и сервера. Он содержит в себе всю, необходимою информацию о пользователе.

При разработке приложения перед мной стала задача, суть которой заключается в автообновлении данных, которую можно перефразировать следующим образом: показывать ли актуальные значения после обновления страницы или же можно реализовать автообновление всех этих данных? В данном случае потребуется такое решение, которое позволит динамически менять элементы без лишней перезагрузки страницы и с минимальными ресурсозатратами.

Папка «SignalR» содержит в себе классы, которые реализуют библиотеку «Microsoft.AspNetCore.SignalR»[3]. Это библиотека с открытым кодом, которая упрощает добавление веб-функций в режиме реального времени в приложения. Веб-функции в режиме реального времени позволяют серверным кодом мгновенно отправлять содержимое клиентам. Важно понимать, что SignalR создает свой отдельный канал подключения «hub connection» по которому он общается с клиентом при помощи отправки сообщений в канале связи. Этот канал связи создает пользовательский интерфейс. Содержание папки «SignalR» показано на рисунке 1.5. В проекте эта библиотека реализована для следующих функций:

* информирование пришедших сообщений пользователю;
* информирование пользователя о прочитанности сообщения;
* показ статус пользователя (online/offline).

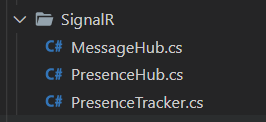


Рисунок 1.5 – Состав папки «SignalR»

В листинге 1.1 показаны метод из класса «MessageHub». создания сообщения и отправка сообщения получателю о том, что ему пришло новое письмо.

|  |
| --- |
| public async Task SendMessage(CreateMessageDto createMessageDto)  { var username = Context.User.GetUsername();      if (username == createMessageDto.RecipientUsername.ToLower())     throw new HubException("You cannot send messages to yourself");  var sender = await \_unitOfWork.UserRepository.GetUserByUsernameAsync(username);  var recipient = await \_unitOfWork.UserRepository.GetUserByUsernameAsync(createMessageDto.RecipientUsername);      if (recipient == null) throw new HubException("Not found user");      var message = new Message{         Sender = sender, Recipient = recipient,         SenderUsername = sender.UserName,         RecipientUsername = recipient.UserName,         Content = createMessageDto.Content};  var groupName = GetGroupName(sender.UserName,  recipient.UserName); var group = await \_unitOfWork.MessageRepository.GetMessageGroup(groupName);              if (group.Connections.Any(x => x.Username == recipient.UserName)) {message.DateRead = DateTime.UtcNow;}              else{var connections = await \_tracker.GetConnectionsForUser(recipient.UserName);  if (connections != null) {  await \_presenceHub.Clients.Clients(connections).  SendAsync("NewMessageReceived",  new { username = sender.UserName, knownAs = sender.KnownAs });}}  \_unitOfWork.MessageRepository.AddMessage(message);  if (await \_unitOfWork.Complete())  {await Clients.Group(groupName).SendAsync("NewMessage", \_mapper.Map<MessageDto>(message));}} |

Листинг 1.1 – Метод, реализующий создание нового сообщения и отправку уведомления получателю о его поступлении

Все фотографии пользователей хранятся на внешнем сервере от сервиса «Cloudinry», в папке «Services» находится класс который напрямую работает с этим сервисом. Для работы с сервисом необходимо допольнительно создать глобальные переменные «ApiKey» и «ApiSecret».

Для того, чтобы пользователям не надо было каждый раз при посещении приложения авторизовываться в сети, была создана релизация JWT токенов, которые хранят в себе всю необходимую информацию о пользователе (id, имя, роль). В листинге 1.2 находится метод «CreateToken» из класса «TokenService».

|  |
| --- |
| public async Task<string> CreateToken(AppUser user)  {   var claims = new List<Claim>  {         new Claim(JwtRegisteredClaimNames.NameId, user.Id.ToString()),         new Claim(JwtRegisteredClaimNames.UniqueName, user.UserName)  };   var roles = await \_userManager.GetRolesAsync(user);  claims.AddRange(roles.Select(role => new Claim(ClaimTypes.Role, role)));  var creds = new SigningCredentials(\_key,  SecurityAlgorithms.HmacSha512Signature);  var tokenDescriptor = new SecurityTokenDescriptor{      Subject = new ClaimsIdentity(claims),       Expires = DateTime.Now.AddDays(7),       SigningCredentials = creds};   var tokenHandler = new JwtSecurityTokenHandler();  var token = tokenHandler.CreateToken(tokenDescriptor);              return tokenHandler.WriteToken(token);  } |

Листинг 1.2 – Метод, создающий токены для пользователей

Токены создаются со сроком действия в семь дней, после чего они являются не валидными. Они хранятся в cookies в браузере, и каждый раз вставляются в заголовки запросов, которые идут от пользовательского интерфейса на сервер.

Папка «SignalR» кроме класса, который отвечает за сообщения, еще есть классы, отвечающие за статус пользователя в сети. Проект был разработан с помощью среды разработки Microsoft Visual Studio 2019.

Для разработки клиентской части веб-приложения был выбрал прогрессивный фреймворк Angular, который написан на языке TypeScript. Он один из самых масштабных из существующих веб-фреймворков. Он включает в себя множество встроенных возможностей. Проектирование с помощью Angular подразумевает создание компонентов, которые взаимодействуют друг с другом. Старт приложения начинается со страницы index.html, которая уже в свою очередь внутри содержит тег «app-root», который и определяет весь интерфейс приложения. Структуру проекта можно увидеть на рисунке 1.6. Каждый angular проект содержит в себе файл «angular.json». Он является главным конфигурационным файлом так называемого Angular Workspace (директория my-app), сгенерированного с использованием @angular/cli и объединяющего в себе множество проектов (само приложение и созданные библиотеки для него). Также в этом файле идет попределения ssl сертификатов и ключей для https подключения.

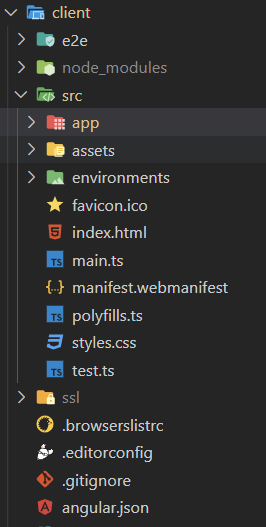


Рисунок 1.6 – Структура проекта пользовательского интерфейса

Папка «ssl» хранит в себе ключ и сертификат ssl необходимые для работы с «back-end» который принимает только https запросы. Такой подход гарантирует защищенность канала запросов. Папка «app» хранит в все компоненты, интерфейсы и вспомогательные классы проекта, перехватчики, которые перехватывают и обрабатывают все запросы и ответы на сервер и от сервера. На рисунке 1.7 представлено содержание папки.

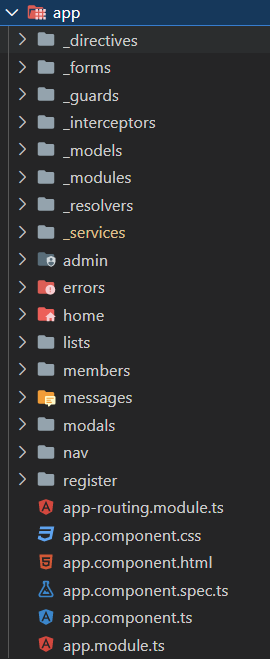


Рисунок 1.7 – сосав папки «app»

В этой папке содержатся основные файлы: «app.module.ts» который содержит в себе все включения всех фреймворков, которые были использованы в пользовательском приложении, компоненты которые были созданы в приложении и объявление подключения перехватчиков ошибок (Error), любой загрузки любых компонентов (Loading) и расшифровщик токенов пользователей (JWT), которые перехватывают контекст данных и обрабатывает его если он подходи под условия (все эти классы реализованы в папке \_interceptors); «app-routing.ts» который является маршрутизатором приложения, так как один из самых важных принципов angular – это создание одностраничных приложений с множеством компонентов; «app.component.html/.ts/.css» является дочерним корневым компонентом с тегом «app-root», который вызывает файл «index.html», внутри содержится ссылка на меню приложения и основной контекст, которым управляет файл «app-routing.ts».

Папка «\_derectives» содержит файл, который проверяет наличие ролей у пользователя. Папка «\_forms» содержит в себе все модальные окна, которые всплывают у пользователей в приложении. Папка «\_models» хранит все модели данных пользовательского интерфейса которые возвращаются ответами от сервера.

Папка «\_guards» содержит в себе файлы, которые препятствуют различным действиям, она содержит следующие файлы: «admin.guard.ts» содержит класс, который проверяет наличие ролей администратора или модератора и если ни одной из ролей у пользователя нет, запрещает вход на администраторские компоненты, данный класс подключается к «admin-panel.component» который находится в папке «admin»; «auth.guard.ts» проверяет вошел ли пользователь в приложение и если нет, то выводит соответствующее сообщение, это необходимо на тот случай, если пользователь поделился внутренней ссылкой и чтобы не авторизованный пользователь не смог по ней пройти, необходимо это проверять; «prevent-unsaved-changes.guard.ts» содержит класс, который препятствует закрытию страницы без сохранения изменений и выводит сообщение о том, что пользователь забыл сохранить изменения, этот класс необходим на странице, где пользователь изменяет данные.

Папка «\_services» хранит в себе классы, которые отправляют запросы на сервер. Также в папке хранятся сервисы, которые поддерживают hub-соединение (MessageService, PresenceService) с помощью библиотек «HubConnection, HubConnectionBuilder» и класс «PaginationHelper» который берет из заголовка ответа от сервера информацию для того, чтобы отображать пользователей постранично на домашней странице веб-приложения. Остальные классы с помощью библиотеки «HttpClient» общаются с сервером с помощью http запросов. Листинг класса MessageService.ts находится в листинге 1.3.

|  |
| --- |
| export class MessageService  {    baseUrl = environment.apiUrl;    hubUrl = environment.hubUrl;    private hubConnection: HubConnection;    private messageThreadSource = new  BehaviorSubject<Message[]>([]);    messageThread$ = this.messageThreadSource.asObservable();    constructor(private http: HttpClient, private busyService: BusyService) { }    createHubConnection(user: User, otherUsername: string) {      this.busyService.busy();     this.hubConnection = new HubConnectionBuilder()        .withUrl(this.hubUrl + 'message?user=' + otherUsername, {          accessTokenFactory: () => user.token})        .withAutomaticReconnect()        .build()      this.hubConnection.start()        .catch(error => console.log(error))        .finally(() => this.busyService.idle());      this.hubConnection.on('ReceiveMessageThread', messages =>  {        this.messageThreadSource.next(messages);      })      this.hubConnection.on('NewMessage', message =>  {        this.messageThread$.pipe(take(1)).subscribe(messages =>  {          this.messageThreadSource.next([...messages, message]) |

|  |
| --- |
| })  })      this.hubConnection.on('UpdatedGroup', (group: Group) => {        if (group.connections.some(x => x.username === otherUsername)) {          this.messageThread$.pipe(take(1)).subscribe(messages => {            messages.forEach(message => {              if (!message.dateRead) {                message.dateRead = new Date(Date.now())              }  })            this.messageThreadSource.next([...messages]);          })  }})}    stopHubConnection() {      if (this.hubConnection) {        this.messageThreadSource.next([]);        this.hubConnection.stop();      }}    getMessages(pageNumber, pageSize, container) {      let params = getPaginationHeaders(pageNumber, pageSize);      params = params.append('Container', container);      return getPaginatedResult<Message[]>(this.baseUrl + 'messages', params, this.http);  }    getMessageThread(username: string) {      return this.http.get<Message[]>(this.baseUrl + 'messages/thread/' + username); }    async sendMessage(username: string, content: string)  {      return this.hubConnection.invoke('SendMessage', {  recipientUsername: username, content})        .catch(error => console.log(error));  }    deleteMessage(id: number)  {      return this.http.delete(this.baseUrl + 'messages/' + id); }} |

Листинг 1.3 – Скрипт класса MessageService

Все остальные папки содержат в себе компоненты приложения с одноименными названиями. На примере компонента «member-card», который находится в папке «members» я объясню основной принцип работы angular компонентов. На рисунке 1.8 находится скриншот папки компонента «member-card». Все angular компоненты состоят из: файла языка разметки, файла стилей и файла класса компонента. Файл «member-card.component.html» содержит в себе html разметку компонента, в данном случает карточку пользователя с именем, возрастом и фотографией.

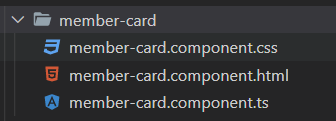


Рисунок 1.8 – Cостав компонента members

Также в данной карточке есть всплывающие кнопки с иконками, которые помогают перейти в чат, на личную страницу пользователя или лайкнуть пользователя. Листинг файла языка разметки представлен в листинге 1.4.

|  |
| --- |
| <div class="card mb-4">  <div class="card-img-wrapper">   <img src="{{member.photoUrl || './assets/user.png'}}" alt="{{member.knownAs}}" class="card-img-top">   <ul class="list-inline member-icons animate text-center">   <li class="list-inline-item"><button routerLink='/members/{{member.username}}' class="btn btn-primary">  <i class="fa fa-user"></i></button></li>  <li class="list-inline-item"><button (click)="addLike(member)" class="btn btn-primary">  <i class="fa fa-heart"></i></button></li>  <li class="list-inline-item">  <button         routerLink='/members/{{member.username}}'         [queryParams]='{tab: 3}'         class="btn btn-primary">         <i class="fa fa-envelope"></i>  </button></li>  </ul>  </div>  <div class="card-body p-1">  <h6 class="card-title text-center mb-1">   <span [class.is-online]="(presence.onlineUsers$ | async).  includes(member.username)">  <i class="fa fa-user mr-2"></i>   </span>  {{member.knownAs}}, {{member.age}}  </h6>  <p class="card-text text-muted text-center">{{member.city}}</p>  </div> </div> |

Листинг 1.4 – Файл языка разметки компонента «member-card»

Класс компонента содержит в себе опеределения «Component»: название тега, по которому можно вставить компонент в другой компонент через html разметку под полем «selector», ссылку на html страницу, к которой привязан компонент (поле «templateUrl») и ссылки (их может быть много) на файл стилей (поле «styleUrls»). Также внутри класса компонента есть конструктор, в котором можно определить объекты других классов. В классе компонента включаются методы, которые можно вызвать из разметки. Листинг файла класса компонента находится в листинге 1.5.

|  |
| --- |
| @Component({    selector: 'app-member-card',    templateUrl: './member-card.component.html',    styleUrls: ['./member-card.component.css'],  })  export class MemberCardComponent implements OnInit {    @Input() member: Member;  constructor(private memberService: MembersService, private toastr: ToastrService,   public presence: PresenceService) { }  ngOnInit(): void { }    addLike(member: Member) {      this.memberService.addLike(member.username).subscribe(() => {        this.toastr.success('You have liked ' + member.knownAs);      })    }} |

Листинг 1.5 – Скрипт класса компонента «member-card»

Файл «member-card.component.css» содержит в себе все стили, применяемые к компоненту. Они написаны на языке css и описывают блоки кода конкретного компонента. В проекте был использован подход к построению «flex box», что дает возможность располагать блоки в строку и накладывать их друг на друга. Несмотря на то, что весь пользовательский интерфейс использует библиотеку «bootstrap». «Bootstrap» — это открытый и бесплатный HTML, CSS и JS фреймворк, который используется веб-разработчиками для быстрой вёрстки адаптивных дизайнов сайтов и веб-приложений.  Он позволяет верстать сайты в несколько раз быстрее, чем на «чистом» CSS и JavaScript. Однако необходимость в доработке дизайна остается, именно поэтому в некоторых компонентах в файле стилей есть код, а в некоторых нет. Ещё один аспект библиотеки – доступность. Она сводится к тому, что предоставляет возможность даже начинающему веб-разработчику (без глубоких знаний и достаточной практики) создавать достаточно качественные макеты.

1.2. Администрирование проекта

В современном мире существует большое количество сред и оперативных систем, которые имею свою инфраструктуру. Приложения, созданные на одной ОС, вряд ли заработают на другой ОС. Так же, чтобы запустить приложение на той же ОС, на какой создавалось приложение, требуется установить вспомогательные среды разработки, базы данных. На мой взгляд, наиболее быстрый и практичный способ является использование облачных сервисов, например Heroku.

Heroku — облачная мультиязычная платформа как услуга (PaaS), основанная на управляемой [контейнерной системе](https://blog.skillfactory.ru/glossary/kontejnerizacziya/), с интегрированными службами передачи данных и развитой экосистемой для развертывания и запуска приложений. Обычно приложения работают на выделенном сервере, а для сайтов используют хостинги. Но возможности хостингов ограничены. А выделенные серверы, такие как VPS, нужно настраивать: самостоятельно определять архитектуру, собирать приложение, заботиться о безопасности. Тратить на это ресурсы не всегда возможно. В таких случаях используется Heroku. Платформа позволяет загружать любое приложение и не заниматься настройкой серверной части. Heroku — Platform as a Service. Это означает, что платформа работает как сервис: предоставляет пользователю определенные функции и возможности, доступ к системам и ПО. При этом ее инфраструктура полностью скрыта.

За пользователя все делают сотрудники сервиса — эта работа остается «под капотом», а многие процессы автоматизированы. За безопасность, архитектуру и настройку сервера отвечают специалисты платформы.

Поэтому Heroku нужна:

* для размещения приложений и веб-сервисов;
* упрощения и ускорения цикла разработки;
* снижения потребности в сложной работе с сервером;
* работы с нагруженными приложениями;
* быстрого масштабирования проектов.

Для того, чтобы выгрузить приложение на Heroku необходимо зарегистрироваться на сервисе и создать там приложение. На рисунке 1.9 показан список приложений после создания. Во время создания приложения, сервис предложит выбрать местонахождение сервера, на который будет выгружаться приложение. По умолчанию это Unated States.

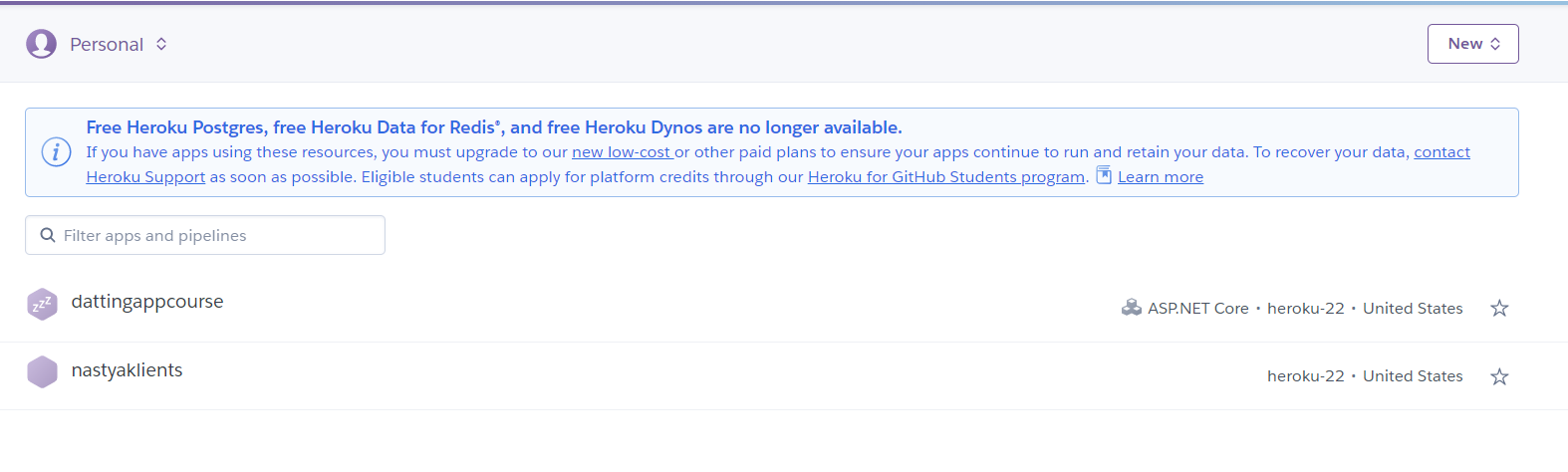


Рисунок 1.9 – Список приложений пользователя

Далее необходимо перейти в сервис Heroku в настройки приложения. В вкладке Recourses необходимо добавить «Heroku postgress», это база данных на основе PostgreSql от сервиса. На рисунке 1.10 представлен интерфейс вкладки «Resources» с уже добавленной базой данных.

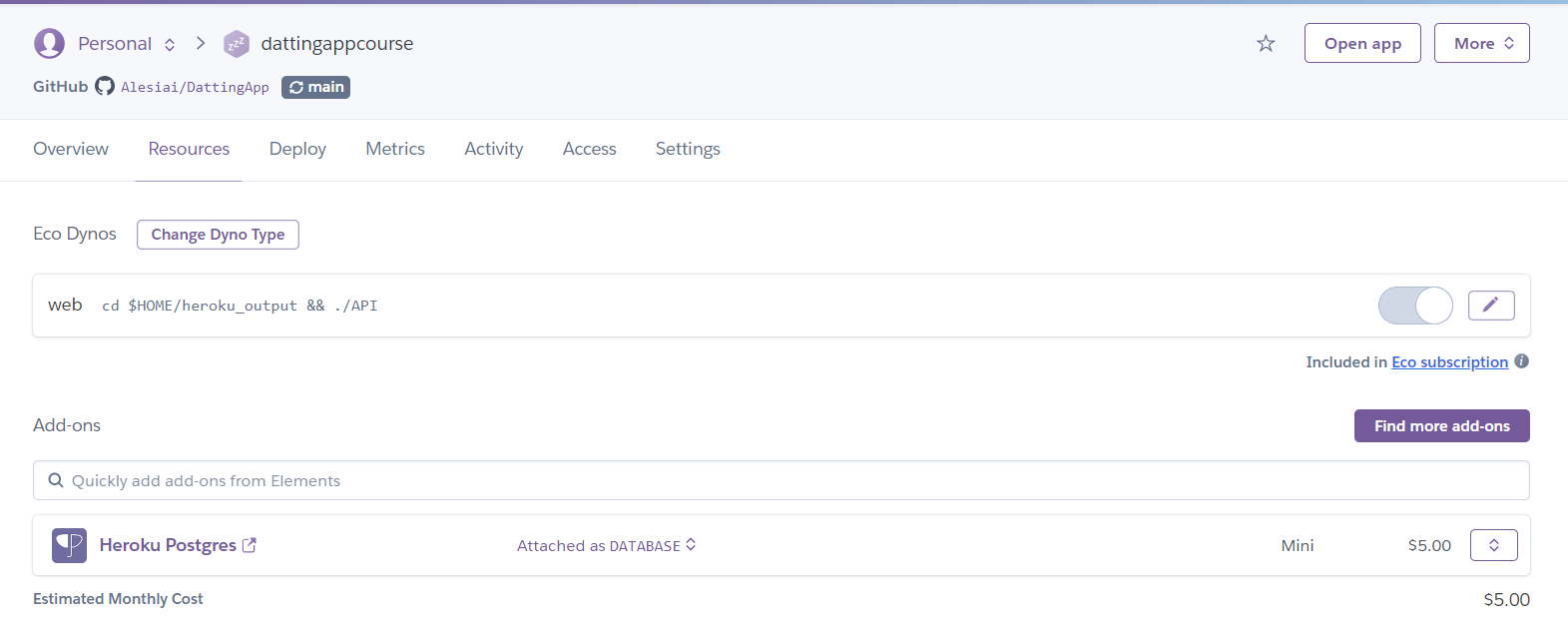


Рисунок 1.10 – Вкладка «Resources»

Далее во вкладке «Settings» необходимо ввести глобальные переменные приложения (те, что находятся в файле «appsettings.json»). На рисунке 1.11 представлен интерфейс вкладки и то, что необходимо ввести в переменные.

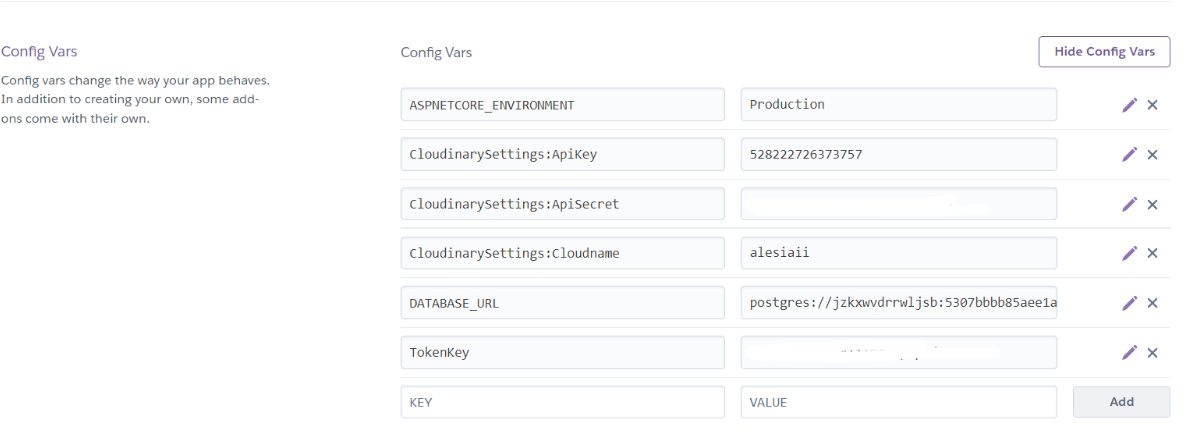


Рисунок 1.11 – Глобальные переменные проекта

После создания глобальных переменных проекта внутри сервиса, стоит создать код, который будет менять контекст подключения к базе данных в зависимости от режима проекта (Development / Production). В листинге 1.6 представлен этот код. Режим проекта Development подразумевает режим отладки проекта на локальном сервере. В этом режиме проект работает с базой данных «PostgreSQL», которая работает на docker в контейнере dev. Для отладки базы данных использовалась «pgAdmin». «[PgAdmin](https://www.pgadmin.org/)» — это платформа с открытым исходным кодом для администрирования и разработки для базы данных PostgreSQL и связанных с ней систем управления базами данных. Платформа написана на Python и jQuery и поддерживает все функции PostgreSQL. Эту программу можно использовать для любых операций, начиная с записи базовых SQL-запросов и заканчивая осуществлением мониторинга ваших баз данных и настройки продвинутых архитектур баз данных.

|  |
| --- |
| public static class ApplicationServiceExtensions      {          public static IServiceCollection AddApplicationServices(this IServiceCollection services, IConfiguration config)          {services.AddSingleton<PresenceTracker>();       services.Configure<CloudinarySettings>(config.GetSection("CloudinarySettings"));              services.AddScoped<ITokenService, TokenService>();              services.AddScoped<IPhotoService, PhotoService>();              services.AddScoped<IUnitOfWork, UnitOfWork>();              services.AddScoped<LogUserActivity>();              services.AddAutoMapper(typeof(AutoMapperProfiles).Assembly);              services.AddDbContext<DataContext>(options =>              {var env = Environment.GetEnvironmentVariable("ASPNETCORE\_ENVIRONMENT");  string connStr;                  if (env == "Development")                  {connStr = config.GetConnectionString("DefaultConnection");}                  else                  {// Use connection string provided at runtime by Heroku.               var connUrl = Environment.GetEnvironmentVariable("DATABASE\_URL");  // Parse connection URL to connection string for Npgsql                      connUrl = connUrl.Replace("postgres://", string.Empty);                      var pgUserPass = connUrl.Split("@")[0];                      var pgHostPortDb = connUrl.Split("@")[1];                      var pgHostPort = pgHostPortDb.Split("/")[0];                      var pgDb = pgHostPortDb.Split("/")[1];                      var pgUser = pgUserPass.Split(":")[0];                      var pgPass = pgUserPass.Split(":")[1];                      var pgHost = pgHostPort.Split(":")[0];                      var pgPort = pgHostPort.Split(":")[1];  connStr = $"Server={pgHost};Port={pgPort};User Id={pgUser};Password={pgPass};Database={pgDb}; SSL Mode=Require; Trust Server Certificate=true";                  } options.UseNpgsql(connStr);              }); return services; }} |

Листинг 1.6 – Код для динамической смены базы данных

В документации Heroku есть информация о именовании серверов базы данных, пользователей, паролей и других необходимых данных. На основании документации «Heroku postgress» был разработан этот код.

1.3. Начало работы

Для того, чтобы настроить приложение, необходимо установить приложение Heroku.cli. Далее необходимо открыть консоль, зайти в папку с проектом и выполнить команды, представленные в листинге 1.7.

|  |
| --- |
| heroku login  heroku git:remote -a dattingappcourse  heroku buildpacks:set https://github.com/jincod/dotnetcore-buildpack |

Листинг 1.7 – Первичная настройка сервиса Heroku

Так как Heroku — это облачная платформа как услуга (PaaS), которая поддерживает множество языков программирования (и этим она очень хвастается и выделяется). История Heroku началась в 2007, и тогда первым языком программирования был Ruby. Теперь она поддерживает .Net, Java, Node.js, Scala, Clojure, Python, PHP и Go. Heroku держит все проекты в docker контейнерах, которые работают также как и локальные контейнеры, поэтому перед выгрузкой приложения на сервер, необходимо устанавливать языковые пакеты .net. В установке node.js нет необходимости, так как во всех контейнеры изначально пакеты node.js установлены. Angular пакеты также нет необходимости настраивать, так как в сборке front-end части уже есть все необходимые механизмы для их автоматической установки на сервере.

Для того, чтобы приложение работало в режиме «Production» необходимо прописать глобальную переменную, это можно сделать из консоли. Также необходимо создать ключ токена, для корректной совместной работы компонентов приложения. Для этого необходимо вернуться в консоль и ввести команды из листинга 1.8. После настройки всех глобальных переменных остается выгрузить проект.

|  |
| --- |
| heroku config:set ASPNETCORE\_ENVIRONMENT = Production  heroku config:set TokenKey = D1k6RLXZyJPtx11Ibg2Oprp3dO8m77zA  git push heroku main |

Листинг 1.8 – Команды для установки конфигурационных данных и выгрузки проекта на сервер Heroku

Сервис Heroku удобен тем, что при выгрузке проекта на сервер, сервис сам выполняет компиляцию и выгрузку файлов. Соответственно, после деплоя всего проекта, по ссылке <https://dattingappcourse.herokuapp.com/> можно будет найти необходимый проект.

Далее для удобства, можно подключить github репозиторий чтобы выгрузка на хостинг производилась автоматически при каждом commit, сделать это можно на вкладке «Deploy». Также в этой вкладке можно настроить из какой ветки будет идти выгрузка при каждом сохранении проекта, что является очень важным и нужным моментом. Такой подход позволяет создавать параллельную отладку проекта или внедрение новых функций или особенностей проекта без выгрузки на хостинг, и только при слиянии веток эта выгрузка будет происходить на сервера Heroku. Выгрузка происходит сразу после commit нужной (указанной) ветки.

# 2. Проверка работоспособности приложения

Первым делом необходимо перейти на <https://dattingappcourse.herokuapp.com/> . Видим поля для авторизации. При успешной авторизации и отображении списка пользователей можно сказать, что приложение работает корректно. После запуска серверной части, пользователь попадает на приветственную страницу приложения, где у него есть возможность или зарегистрироваться, нажав на кнопку «Regester» или авторизоваться, введя данные авторизации в поля «Username» и «Password», после нажав на кнопку «Login» (Рисунок 2.1).

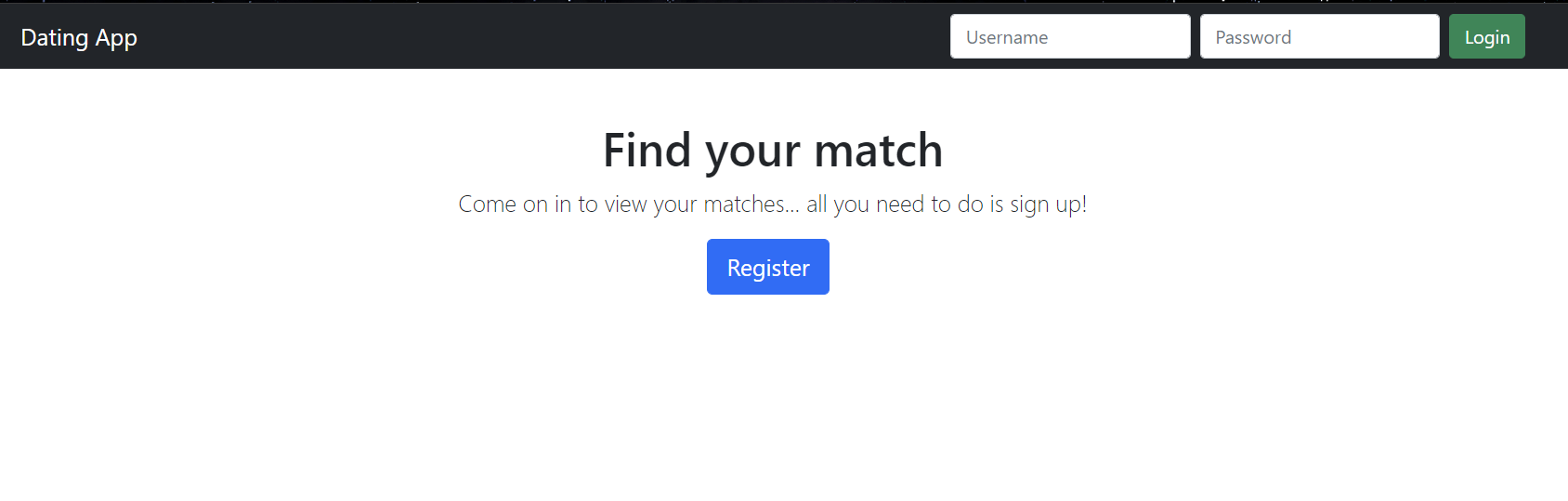


Рисунок 2.1 – Приветственная страница приложения

Так как пользователь не зарегистрирован, он должен нажать на кнопку «Register». Результат нажатия на кнопку показан на рисунке 2.2. Перед пользователем появляется форма регистрации.

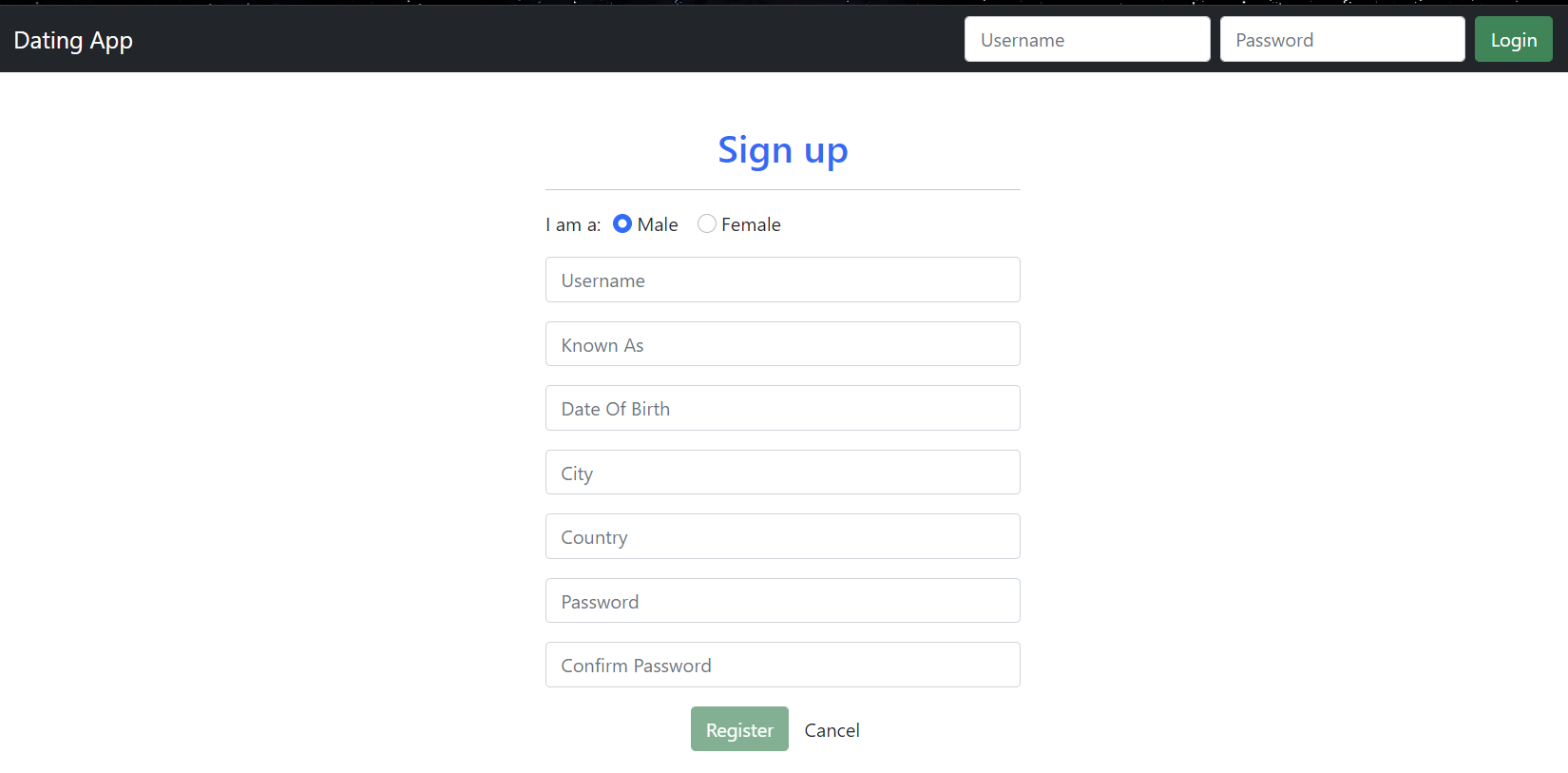


Рисунок 2.2 – Форма регистрации

После регистрации или авторизации обычного пользователя, интерфейс перенаправляет пользователя на домашнюю страницу приложения для обычного пользователя.

**2.1. Пользователь**

На рисунке 2.3 показана домашняя страница для пользователя женского поля.

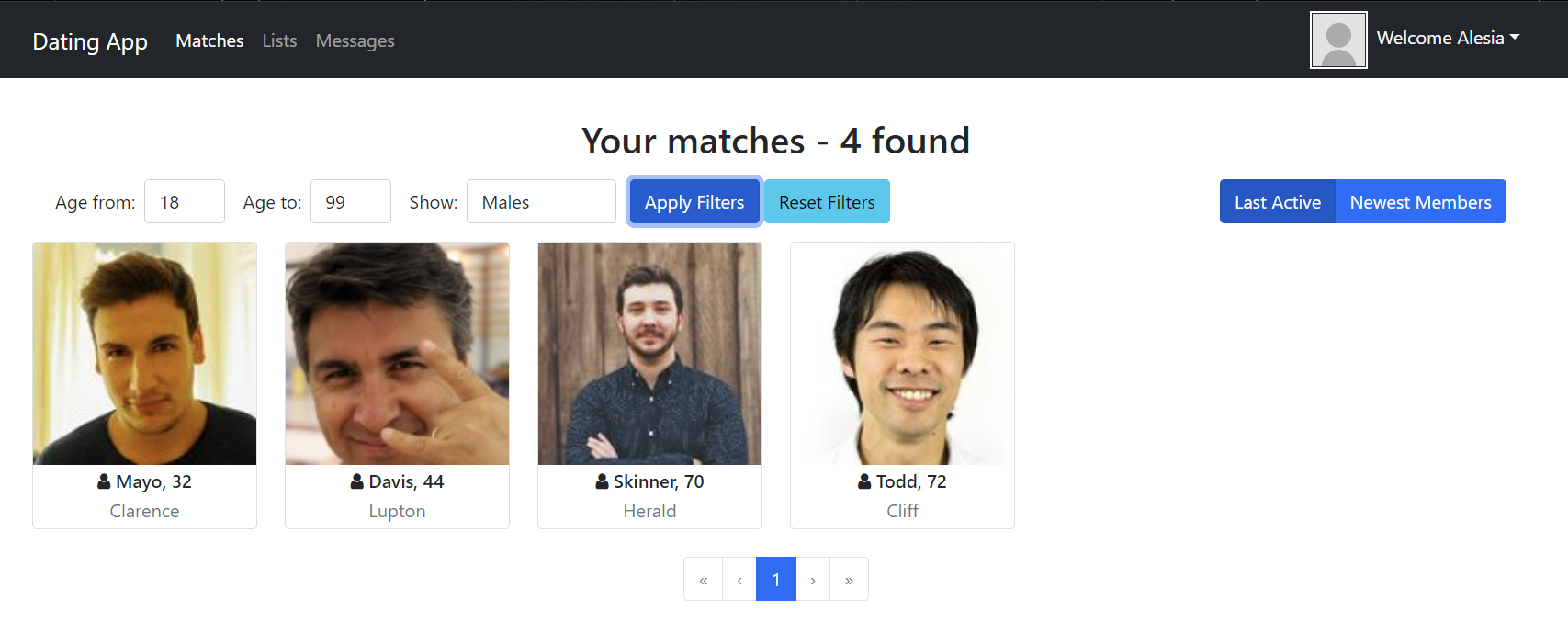


Рисунок 2.3 – Домашняя страница пользователя женского пола.

В приложении в зависимости от пола пользователя показывается по умолчанию показывается список пользователей противоположного пола, сортированные по последней активности на сайте.

**2.2. Модератор**

При авторизации модератора, количество вкладок на шапке меню сайта увеличивается по сравнению с обычным пользователем. Все пользовательские возможности у модератора доступны. В интерфейсе добавляются вкладка «Admin» которая отвечает за управление пользовательской информацией. На рисунке 2.4 представлен интерфейс этой вкладки, также можно увидеть, что по сравнению с рисунком 2.3 (где был интерфейс пользователя), вкладка появилась шапке-меню приложения.

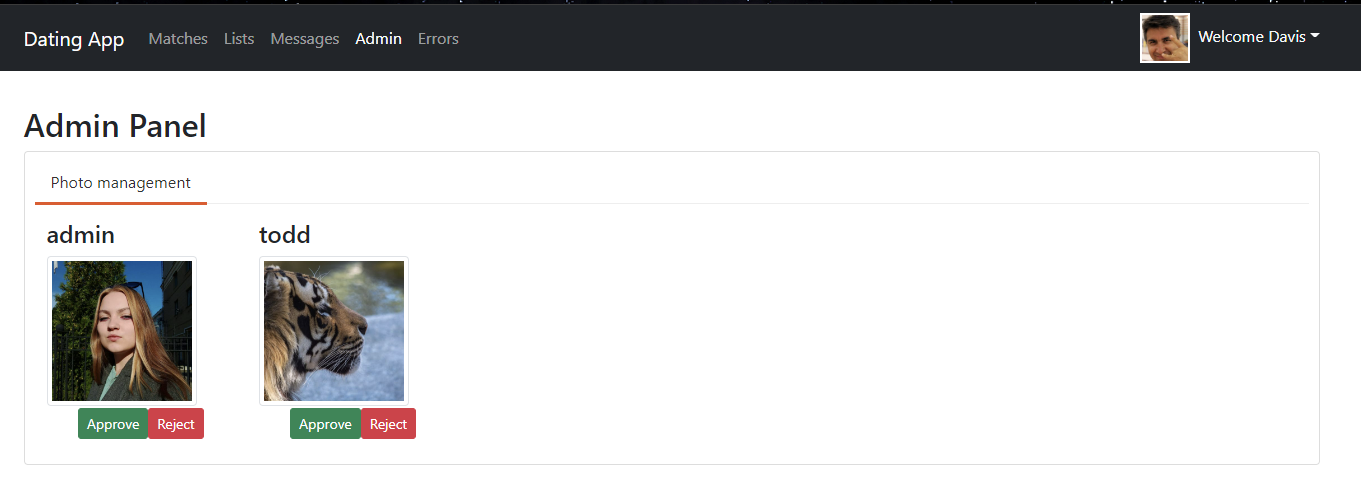


Рисунок 2.4 – Вкладка «Admin» пользователя с модераторскими правами

При нажатии на кнопку «Approve» фотография появляется у пользователя на странице, при нажатии на кнопку «Reject» фотография удаляется.

**2.3. Администратор**

При авторизации администратора, количество вкладок на шапке меню сайта увеличивается по сравнению с обычным пользователем. Все пользовательские и модераторские возможности у администратора доступны. Добавляется вкладка «Admin» которая отвечает за администрирование и управление пользователями. На рисунке 2.5 представлен интерфейс этой вкладки, также можно увидеть, что по сравнению с рисунком 2.3 (где был интерфейс обычного пользователя), вкладка появилась сверху в шапке-меню приложения.

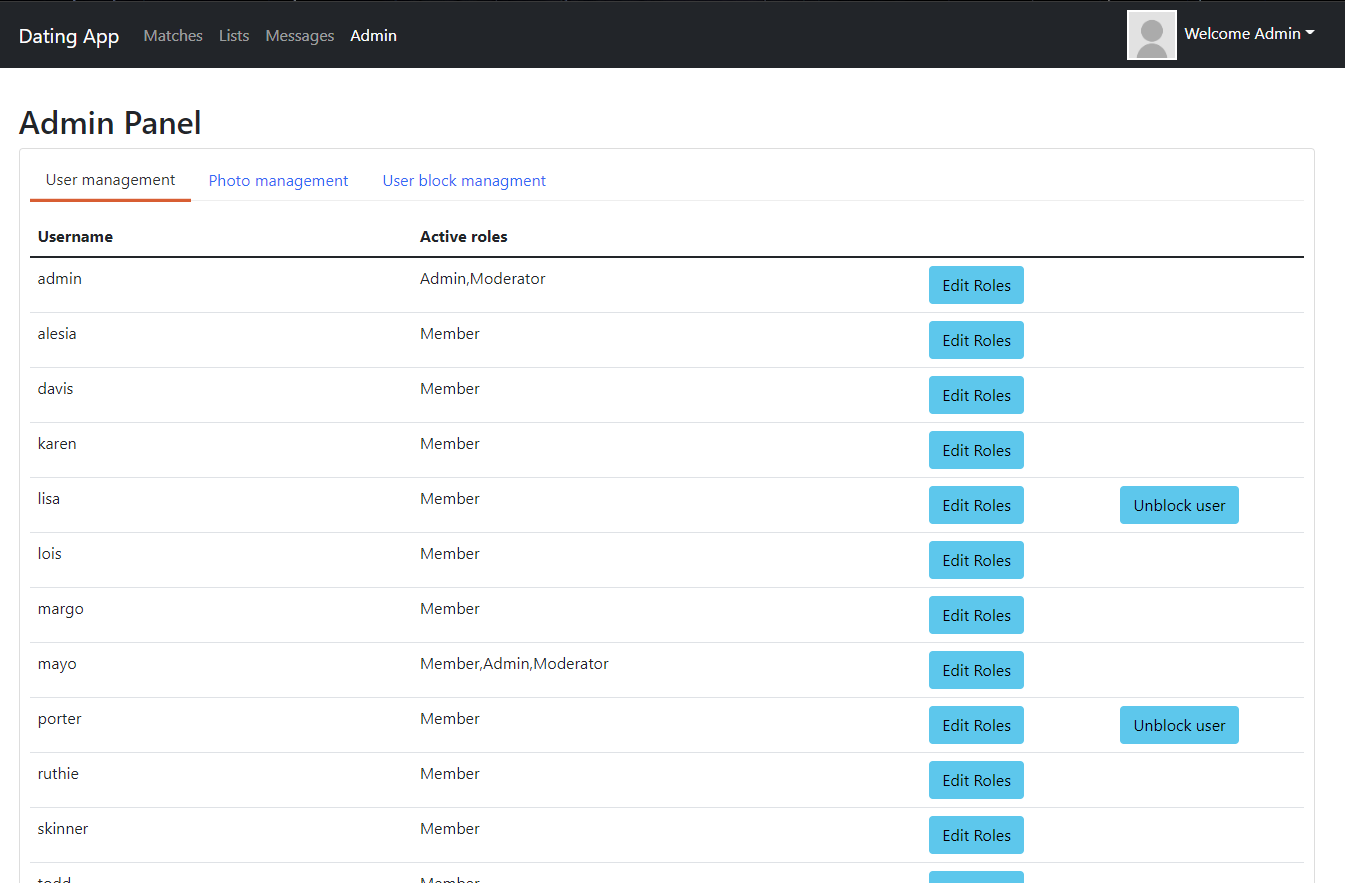


Рисунок 2.5 – Интерфейс вкладки «User managment» страницы «Admin»

На рисунке 2.5 видно, что пользователи с пользовательскими именами «lisa» и «porter» заблокированы и их можно разблокировать, а пользователь с именем «mayo» обладает правами администратора и модератора.

Вкладка «Photo managment» у администратора ничем не отличается от вкладки модератора с тем же названием. На рисунке 2.4 представлен вид вкладки.

Вкладка «User block management» позволяет управлять жалобами на пользователей, интерфейс этой вкладки изображен на рисунке 2.6.

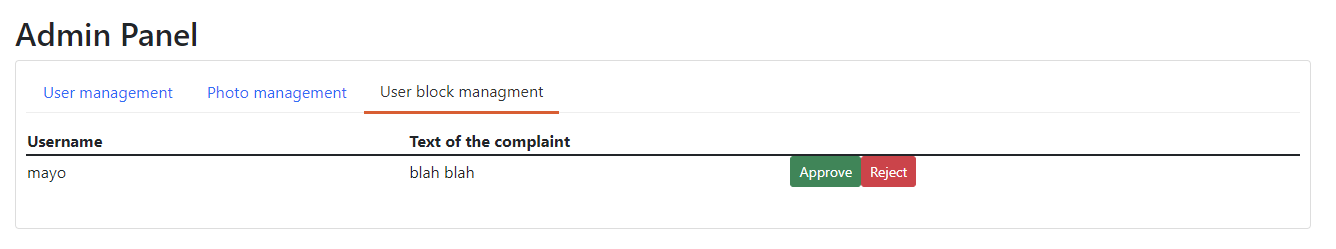


Рисунок 2.6 – Интерфейс вкладки «User block management»

При нажатии на кнопку «Approve» пользователь блокируется, при нажатии на кнопку «Reject» пользователя не блокируют, в обоих случаях заявка удаляется. Разблокировать пользователя можно будет на вкладке «User managment».

Заключение

В результате выполнения работы, первым делом были проанализированы аналоги веб-приложений, которые являются площадками для поиска знакомств. При разработке веб-приложения были учтены как положительные, так и отрицательные черты аналогов.

Для реализации серверной части использовалась платформа ASP.NET Core. ASP.NET Core может работать поверх кроссплатформенной среды .NET Core, которая может быть развернута на основных популярных операционных системах: Windows, Mac OS, Linux. В результате была разработан веб-сервис по архитектуре REST, который обрабатывает клиентские запросы и делает запросы к базе данных.

Была разработана реляционная база данных для веб-приложения, использующая экземпляр PostgreSQL, имеющая в своем наборе 9 таблиц.

С помощью фреймворка Angular была разработана клиентская часть приложения. Запросы к серверу выполняются с помощью библиотек «HttpClient», «HubConnection» и «HubConnectionBuilder».

Были созданы дополнительные конфигурационные файлы, позволяющие быстро и легко администрировать, расширять функционал приложения.

В результате проведенной работы, был создан проект, который удовлетворяет потребностям огромного круга людей. В современном мире роль приложений для знакомств является одной из главных.

# Список литературы

1. Руководство по ASP.NET Core 6 [Электронный ресурс] – Режим доступа: https://metanit.com / – Дата доступа: 01.12.2022
2. PostgreSqL Documentation [Электронный ресурс] – Режим доступа: <https://www.postgresql.org/docs/> – Дата доступа: 09.11.2022
3. Введение в SignalR [Электронный ресурс] – Режим доступа:  [https://docs.microsoft.com/](%20https://docs.microsoft.com/) – Дата доступа: 2.11.2022
4. Руководство по Entity Framework Core [Электронный ресурс] – Режим доступа:  [https://metanit.com/](%20https://metanit.com/sharp/entityframeworkcore/) – Дата доступа: 8.11.2022