НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Р. Е. Алексеева

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА к курсовой работе

Солодун Максим Михайлович

(фамилия, имя, отчество)

Институт	Институт радиоэлектроники и информационных технологий
Кафедра	Графические информационные системы
Группа	23-ИСТ-1-1
День защиты	«27 декабря» 2024 г.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Р.Е. Алексеева

Кафедра Графические информационные системы

Заведующий кафедрой ГИС

(подпись) Филинских А.Д. *(фамилия и. о.)*

(дата)

Разработка web-приложения для организации турниров по видеоиграм
(наименование темы проекта или работы)
ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

КОНСУЛЬТАНТЫ:

1.По

(подпись) *(фамилия и. о.)*

(дата)

2.По

(подпись) *(фамилия и. о.)*

(дата)

3.По

(подпись) *(фамилия и. о.)*

(дата)

РЕЦЕНЗЕНТ

(подпись) *(фамилия и. о.)*

дата)

РУКОВОДИТЕЛЬ:

(подпись) Глумова Е.С. *(фамилия и. о.)*

(дата)

СТУДЕНТ

(подпись) Солодун М.М. *(фамилия и. о.)*

(дата) 23-ИСТ-1-1 *(группа или шифр)*

Проект защищен

Протокол №

С оценкой

					ПЗ-НГТУ-23-ИСТ-1-1-23/08028		
Изм.	*Лист*	*№ докум.*	*Подпись*	*Дата*			
Провер.		Глумова Е.С.			***Разработка web-приложения для организации турниров по видеоиграм***	*Лит.*	*Лист* 1 / *Листов* 74
Разраб.		*Солодун М.М.*					
						НГТУ кафедра ГИС	

Нижегородский государственный технический университет им. Р.Е. Алексеева

Кафедра: Графические информационные системы

УТВЕРЖДАЮ
Зав. кафедрой **Филинских А.Д.**

З А Д А Н И Е
на курсовое проектирование

Студент: Солодун Максим Михайлович

Тема курсового проекта: Разработка web-приложения для организации турниров по видеоиграм

Исходные данные к проекту: Информация из интернета, лекционные материалы, лабораторные работы

Содержание графического материала:

1. Скрины приложения с дизайном
2. Схемы сайта и его функционала

Содержание пояснительной записки:
Перечень вопросов, подлежащих разработке
Анализ исходных данных и разработка ТЗ
Руководство пользователя
Руководство программиста
Листинг

Основная рекомендуемая литература:				
[1] Руководство по Angular 8. Режим доступа https://metanit.com/web/angular2/				
[2] Angular. Режим доступа https://angular.io/docs				
			Руководитель	Глумова Е.С.

«27» декабря 2024 г.

Студент Солодун Максим Михайлович

Дата

Оглавление

Введение

В современном мире видеоигры занимают значительное место в жизни многих людей, представляя собой не только форму развлечения, но и платформу для социальных взаимодействий и профессионального развития. С ростом популярности киберспорта, турниры по видеоиграм стали важным элементом игровой индустрии, привлекая миллионы игроков и зрителей по всему миру. Однако, несмотря на высокий интерес к таким событиям, организация турниров требует значительных усилий и времени. Это связано с необходимостью управления участниками, составлением расписания, и обеспечением справедливости.

Актуальность разработки web-приложения для организации турниров по видеоиграм обусловлена потребностью в удобных, доступных и функциональных инструментах, которые бы автоматизировали основные процессы, связанные с проведением соревнований. Такое приложение может стать полезным как для профессиональных организаторов киберспортивных мероприятий, так и для обычных пользователей, стремящихся создать собственные турниры среди друзей или сообщества.

Целью данной курсовой работы является разработка web-приложения, которое позволит автоматизировать процессы организации турниров по видеоиграм, начиная от регистрации участников и заканчивая предоставлением платформы для турнира. В ходе работы предполагается создать функциональный и интуитивно понятный инструмент, который будет включать возможности управления списком участников, создания своих собственных турниров, принятия участия в уже созданных трунирах и предоставления аналитики по проведённым соревнованиям.

Данная разработка не только способствует развитию киберспортивной культуры, но и предоставляет ценный опыт проектирования и реализации web-

приложений, что является актуальным в условиях растущего спроса на специалистов в области IT и разработки программного обеспечения.

Перечень вопросов, подлежащих разработке:

· Введение;

· Разработка и анализ технического задания;

· Разработка приложения;

· Руководство пользователя;

· Руководство программиста;

· Заключение;

· Список литературы;

· Листинг

1. Анализ исходных данных и разработка ТЗ

1.1 Назначение разработки

Разработать приложения для организации турниров по видеоиграм, которое предназначено для создания команд и турниров, где каждый пользователь может создать свою команду, с которой в последствии сможет участвовать в турнирах.

1.2 Минимальные требования к составу и параметрам технических средств: ЭВМ, внешние устройства.

Для успешной разработки, тестирования и эксплуатации web-приложения для организации турниров по видеоиграм необходимы следующие минимальные технические средства:

Электронно-вычислительная машина (ЭВМ)

- **Процессор**: Двухъядерный процессор с тактовой частотой не менее 2.0 ГГц (например, Intel Core i3 или AMD Ryzen 3).
- **Оперативная память**: Не менее 4 ГБ для разработки и не менее 2 ГБ для пользователей приложения.
- **Накопитель**:
  - Для разработки: Не менее 50 ГБ свободного пространства для установки среды разработки, библиотек, баз данных и хранения исходного кода.
  - Для пользователей: Не менее 10 ГБ свободного пространства для работы с веб-браузером и кэширования данных.
- **Операционная система**: Windows 10, macOS 10.14+, Linux (Ubuntu 20.04 или выше).
- **Сетевое подключение**: Скорость интернета не менее 5 Мбит/с для стабильного подключения к серверу.

Внешние устройства

- **Монитор**:
    - Разрешение: не менее 1366x768 пикселей (рекомендуется Full HD для комфортной работы).
    - Диагональ: от 15 дюймов для удобства просмотра интерфейса приложения.
- **Устройства ввода**:
    - Клавиатура и мышь или трекпад для ввода данных и навигации.

1.3 Требования к информационной и программной совместимости

Для правильной работы с web-приложением необходимо наличие современного браузера, поддерживающего JavaSript и стандарты HTML5/CSS3:

- **Google Chrome** (версия 90 и выше).
- **Mozilla Firefox** (версия 85 и выше).
- **Microsoft Edge** (версия 90 и выше).
- **Safari** (версия 14 и выше).

Все указанные браузеры должны поддерживать:

- JavaScript (включая ES6 и выше).
- Стандарты HTML5 и CSS3.
- API для локального хранилища (localStorage, sessionStorage) для временного сохранения данных.

Также данный проект должен содержать следующие программные инструменты:

- Angular: Основной фреймворк для клиентской части. Требуемая версия — 12 или выше.

- Angular **CLI**: Инструмент командной строки для работы с Angular-приложением (создание, сборка, тестирование).
- TypeScript: используется для разработки клиентской части. Рекомендуемая версия — 4.0 или выше.
- npm: для управления зависимостями проекта
- Visual Studio Code (VS Code): Рекомендуемая среда для разработки.

1.4 Требования к функциональным характеристикам

Программный продукт, разрабатываемый в рамках курсового проекта, должен удовлетворять следующему перечню функциональных требований:

1. Интуитивно понятный пользовательский интерфейс.
2. Наличие форм регистрации и авторизации.
3. Отображение зарегистрированного пользователя.
4. Возможность добавления и удаления команды, и выхода из неё.
5. Возможность создания турнира, участия в нём и выхода из него.

1.5 Выбор и обоснование языков программирования и используемых инструментальных средств

Для реализации web-приложения, предназначенного для организации турниров по видеоиграм, был осуществлен выбор современных языков программирования и инструментальных средств, которые соответствуют требованиям производительности, масштабируемости и удобства разработки. Основными технологиями, применяемыми в проекте, являются Angular, TypeScript, HTML5, SCSS и json-server.

Фреймворк Angular был выбран для разработки клиентской части приложения благодаря его мощному функционалу и модульной архитектуре. Angular предоставляет широкий набор инструментов, включая двустороннее связывание данных, маршрутизацию и шаблоны компонентов, что

значительно ускоряет процесс разработки. Его встроенная поддержка TypeScript позволяет создавать более надежный и читаемый код, а активное сообщество и обширная документация упрощают обучение и решение возникающих задач. Angular также отлично подходит для создания динамических и интерактивных пользовательских интерфейсов, обеспечивая поддержку адаптивного дизайна.

В качестве основного языка программирования был выбран TypeScript. Его преимущества включают статическую типизацию, которая помогает предотвратить множество ошибок на этапе компиляции, и улучшенную читаемость кода за счет использования таких возможностей, как интерфейсы и классы. TypeScript полностью совместим с JavaScript, что делает его использование особенно удобным в экосистеме веб-разработки. Он интегрируется с современными средами разработки, такими как Visual Studio Code, предоставляя инструменты для автодополнения, подсветки синтаксиса и других функций, ускоряющих процесс программирования.

HTML5 был выбран для структурирования веб-страниц приложения. Этот стандарт поддерживает современные возможности, такие как работа с мультимедиа, локальным хранилищем и семантически корректной разметкой, что позволяет создавать удобные и интуитивно понятные пользовательские интерфейсы. HTML5 поддерживается всеми современными браузерами, обеспечивая совместимость и доступность приложения для широкой аудитории пользователей.

SCSS используется для стилизации приложения благодаря его расширенным возможностям по сравнению с традиционным CSS. SCSS поддерживает вложенность селекторов, использование переменных, миксинов и других инструментов, которые упрощают управление стилями и делают код более организованным. Это особенно важно для масштабируемых приложений, где стилизация играет ключевую роль в пользовательском опыте.

Для хранения данных на этапе разработки был выбран json-server. Этот инструмент позволяет быстро и легко эмулировать REST API, обеспечивая возможность работы с данными в формате JSON. Использование json-server экономит время, необходимое для настройки серверной части, и позволяет разработчикам сосредоточиться на клиентской части приложения. При необходимости json-server можно заменить на полноценную серверную инфраструктуру с минимальными изменениями в архитектуре проекта.

Таким образом, выбор Angular, TypeScript, HTML5, SCSS и json-server обусловлен их соответствием современным стандартам веб-разработки, высокой производительностью, удобством использования и возможностью масштабирования. Эти технологии обеспечивают создание надежного и функционального приложения, отвечающего всем требованиям проекта.

2. Внешняя спецификация

2.1 Разработка семантического ядра

Семантическое ядро – ключевые слова, по которым пользователь может найти данное приложение.

Созданное в рамках курсовой работы приложение можно найти по таким ключевым фразам: Турниры по видеоиграм, создать турнир по игре, Create game tournament и так далее.

2.2 Разработка структуры приложения

При загрузке приложения нас встречает главная страница, на которой отображается информация о данном web-приложении и две кнопки: «Создать команду» и «Создать турнир». Если пользователь не авторизован, то при нажатии на любую из них он попадает на страницу регистрации и авторизации. После проведения авторизации пользователь может перейти на страницы создания команд и создания турниров, где пользователь может создавать и удалять их соответственно. Также после авторизации пользователь может зайти в профиль, где может посмотреть всю информацию о себе, своих командах и своих турнирах.

Рисунок 1 – Схема сайта

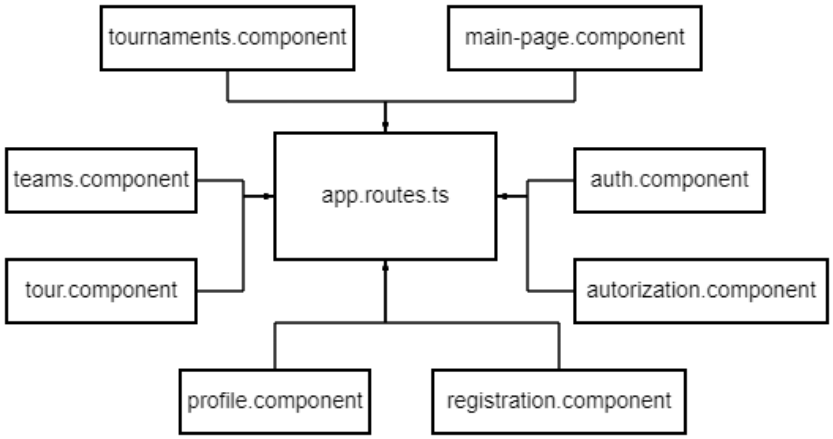


Рисунок 2 – Функциональная схема (1 часть).

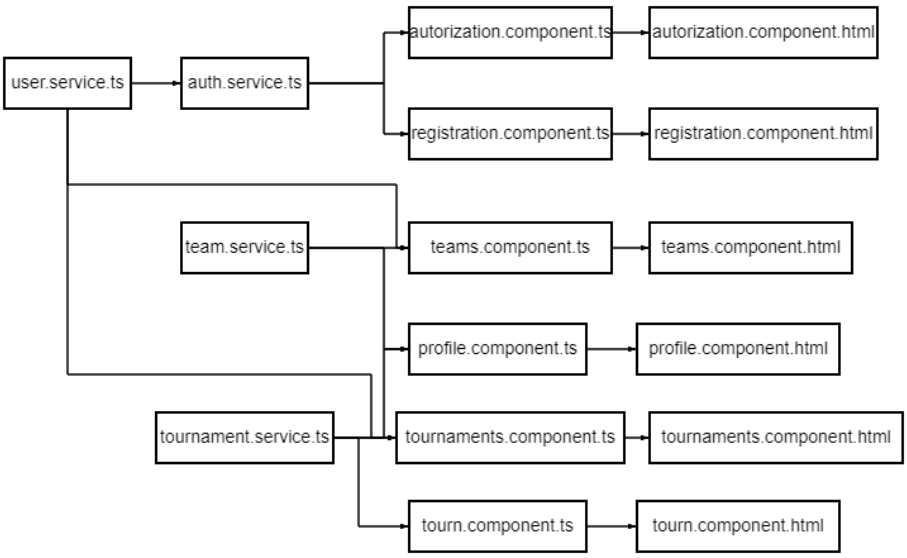


Рисунок 3 – Функциональная схема (2 часть).

3. Руководство пользователя

3.1 Назначение программы

Разработанное приложение предназначено для организации турниров по видеоиграм, что позволило бы игрокам любого типа создавать и участвовать в турнирах со своими друзьями, а также любой организации проводить турниры среди массового количества участников.

3.2 Эргономичность веб-приложения

Эргономичное web-приложения — приложение, созданное с учётом и на основе научных знаний об устройстве и работе человеческого глаза, просматривающего, собирающего (для последующего анализа) информацию с источника излучения определённой спектральной интенсивности, ограниченного по полю обзора. Эргономичное web-приложения обеспечивает необходимые удобства посетителю, сохраняет его силы, здоровье и работоспособность.

Данное приложение имеет такие достоинства как:

- Простота в использовании
- Быстрый доступ к информации
- Интуитивно понятный интерфейс
- Отсутствие избыточности информации

3.3 Описание интерфейса

При запуске приложения пользователь попадает на главную страницу, где он может увидеть всю необходимую информацию о сайте и его предназначении. Также пользователь увидит две кнопки: «Создать свой турнир» и «Создать свою команду», рисунок 4.

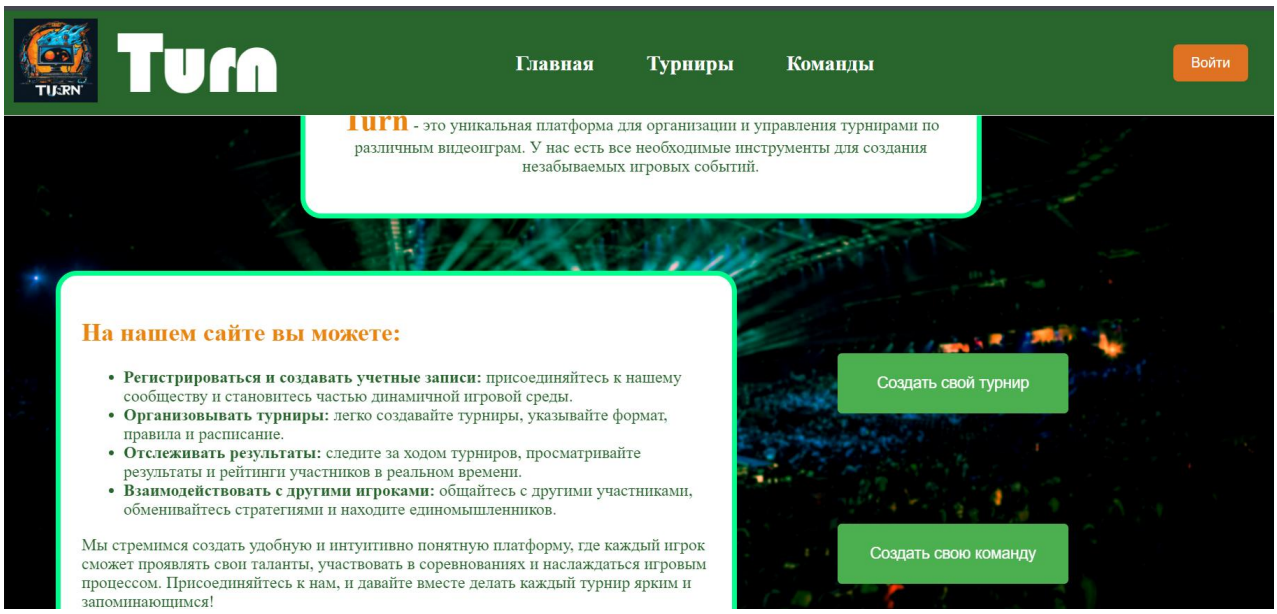


Рисунок 4 – Главная страница.

Если пользователь авторизирован, то при нажатии на соответствующие кнопки пользователь переходит на соответствующие страницы по созданию турнира или команды. А если пользователь не авторизирован, то при нажатии на данные кнопки, ему высвечивается уведомление «Сначала следует пройти авторизацию/регистрацию» и пользователя перебрасывает на формы регистрации, рисунок 5. Если у пользователя нет аккаунта в данном приложении, то для начала нужно пройти регистрацию, а после перейти к форме авторизации. Если же у пользователя есть аккаунт, то он может нажать на кнопку «Уже есть аккаунт? Войти» и пройти авторизацию, рисунок 6.

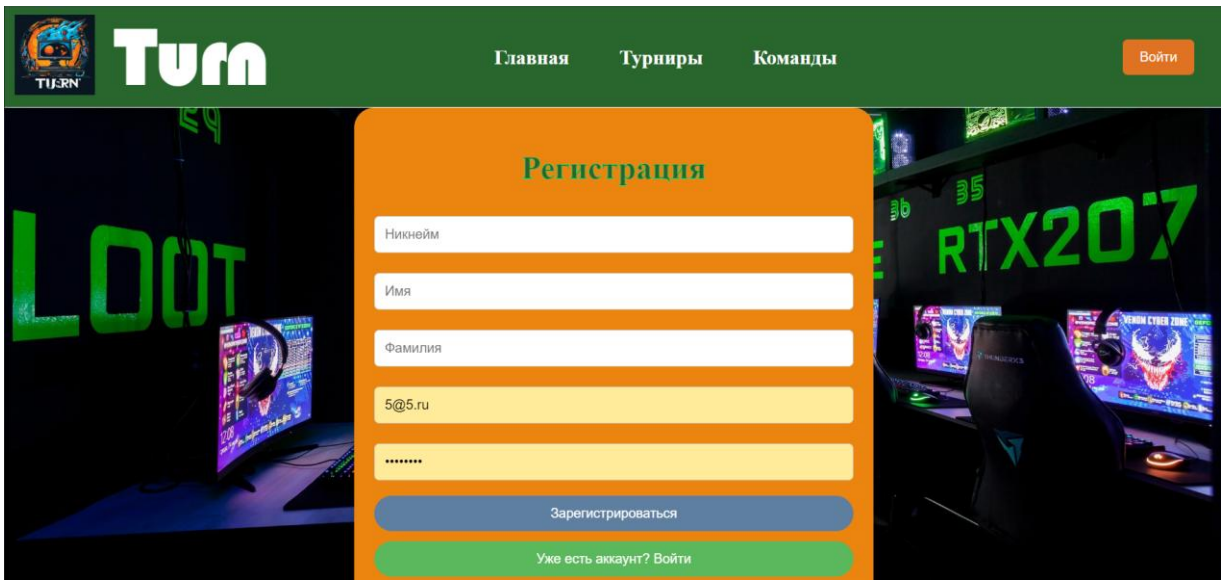


Рисунок 5 – Форма регистрации.

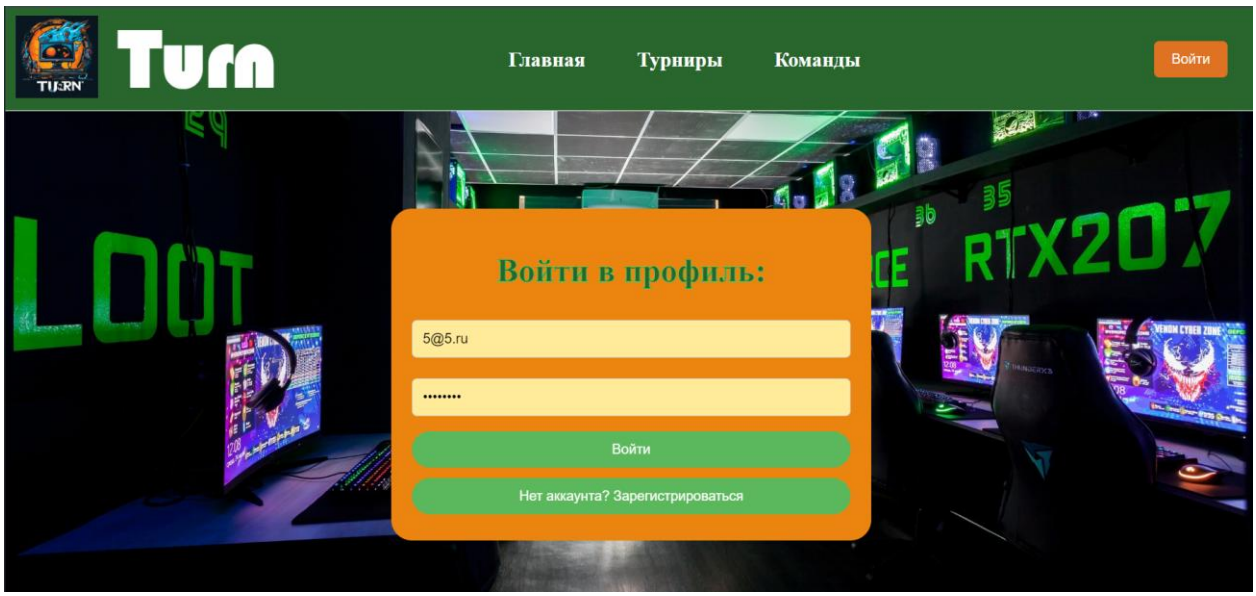


Рисунок 6 – Форма авторизации

После завершения авторизации пользователя перебрасывает на страницу профиля, где будут отображаться вся доступная информация о нём, его командах и турнирах, а также кнопка выхода из профиля, рисунок 7. Там же он может выходить (если он является участником) или удалять (если он является лидером) команды, в которых он присутствует, рисунок 8.

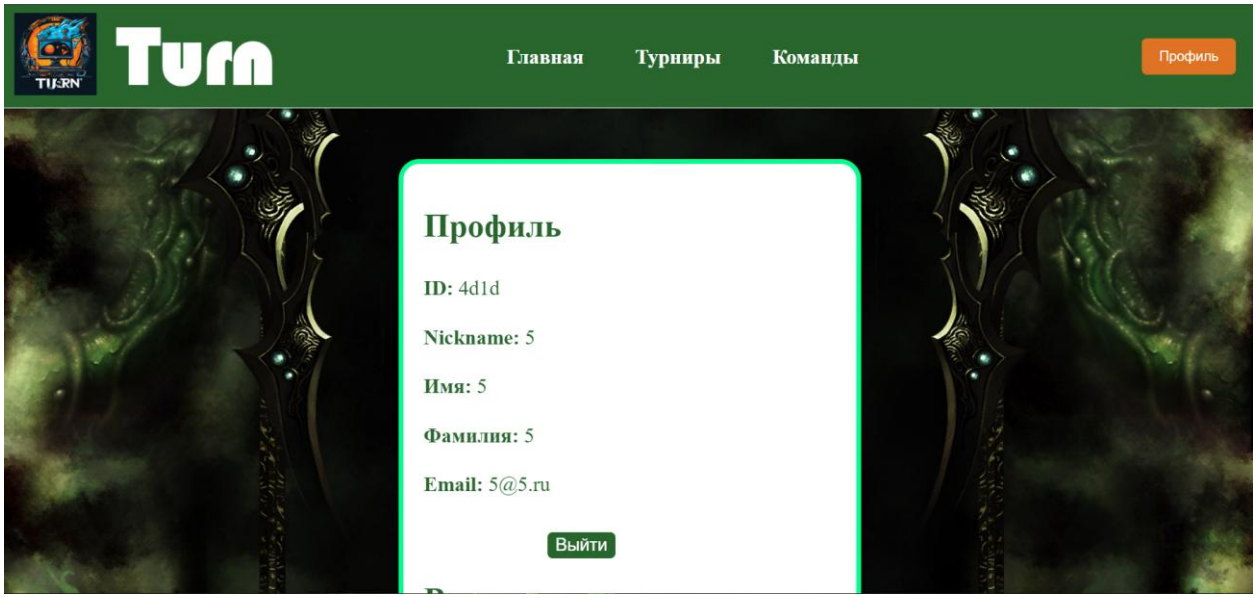


Рисунок 7 – Профиль пользователя с информацией о его личных данных.

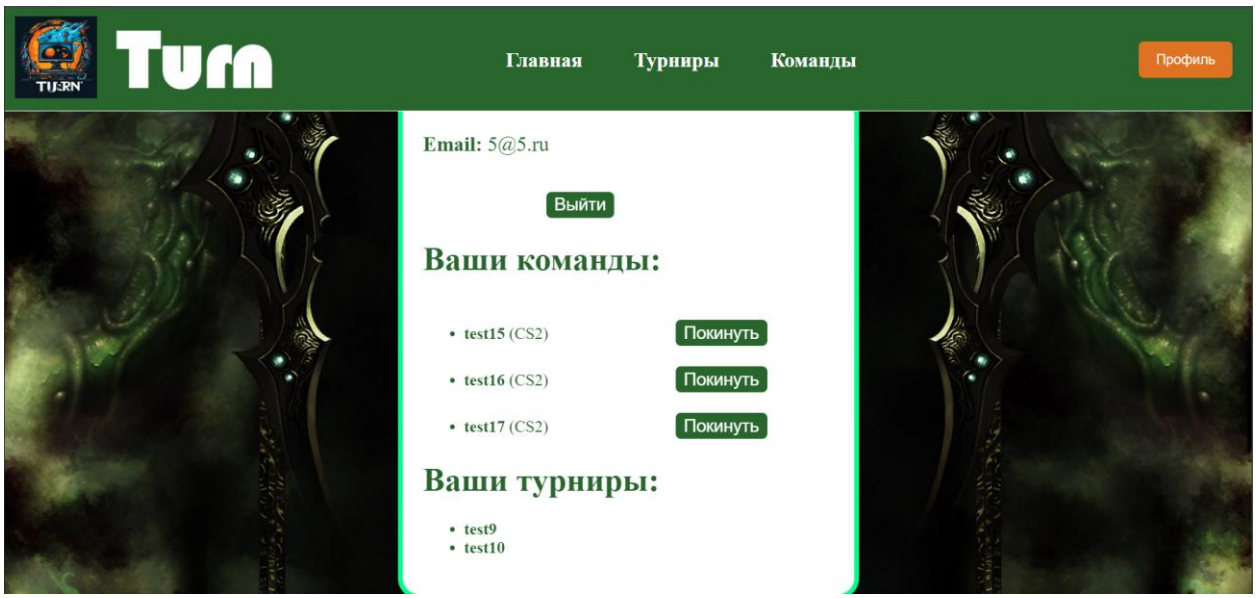


Рисунок 8 – Профиль пользователя с информацией о его командах турнирах.

Далее со страницы профиля пользователь, через навигационное меню в «шапке» web-приложения, может попасть на страницы создания команд и создания турниров по соответствующим кнопкам в меню.

При нажатии на кнопку «Команды» пользователя перенаправляет на страницу создания команды, которая содержит форму для заполнения, рисунок 9. После заполнения формы и нажатия кнопки подтверждения «Создать команды» мы можем наблюдать данную команду в списки наших команд в профиле.

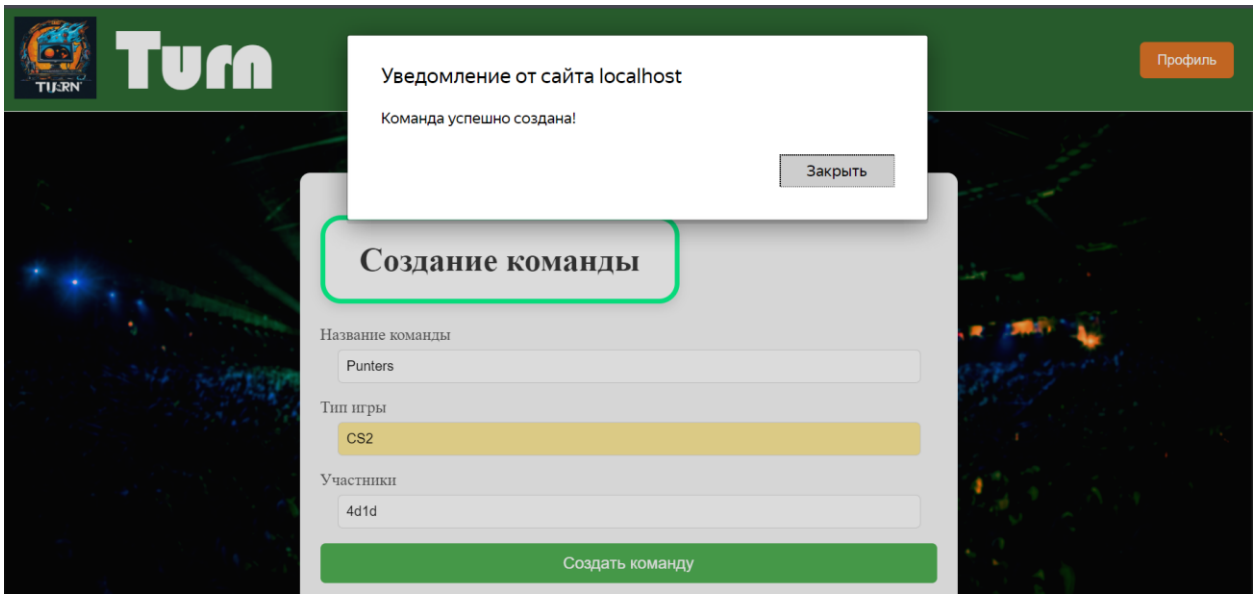


Рисунок 9 – Страница создания команд.

При нажатии в меню на кнопку «Турниры» мы попадаем на страницу турниров, где пользователь может создавать свой турнир аналогично созданию команды, рисунок 10. Так же на данной странице отображаются ближайшие турниры, созданные данным пользователем и другим пользователями, куда текущий пользователь может записать или вычеркнуть команду, при условии, что он её капитан, 11.

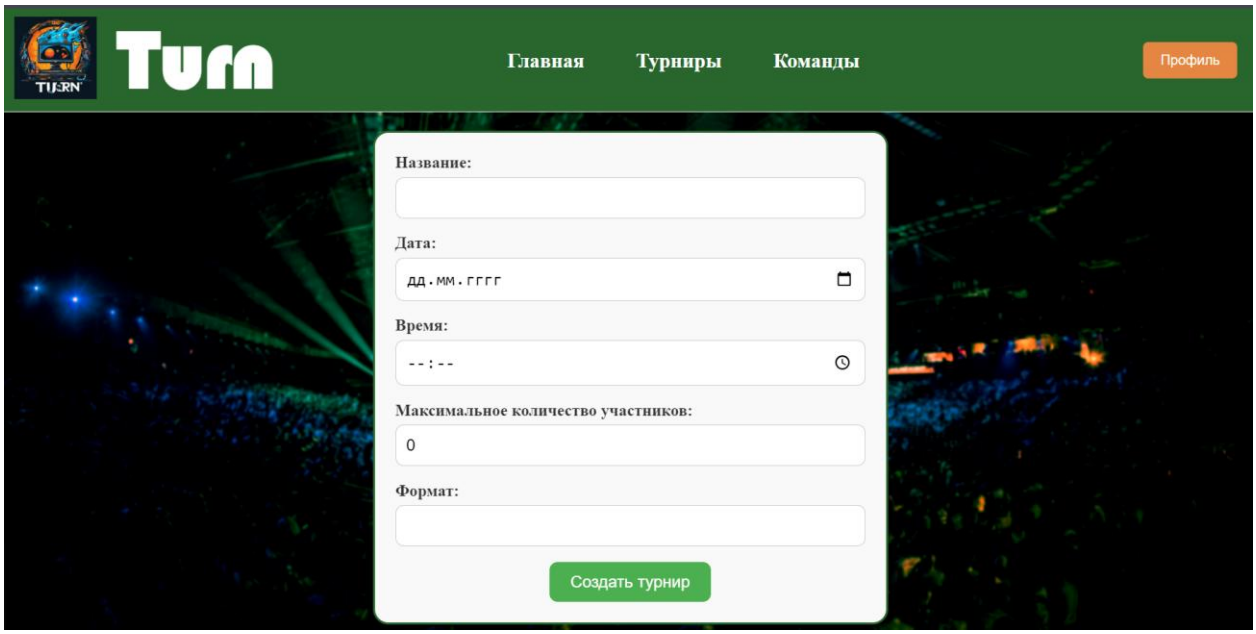


Рисунок 10 – Страница команд, создание команды.

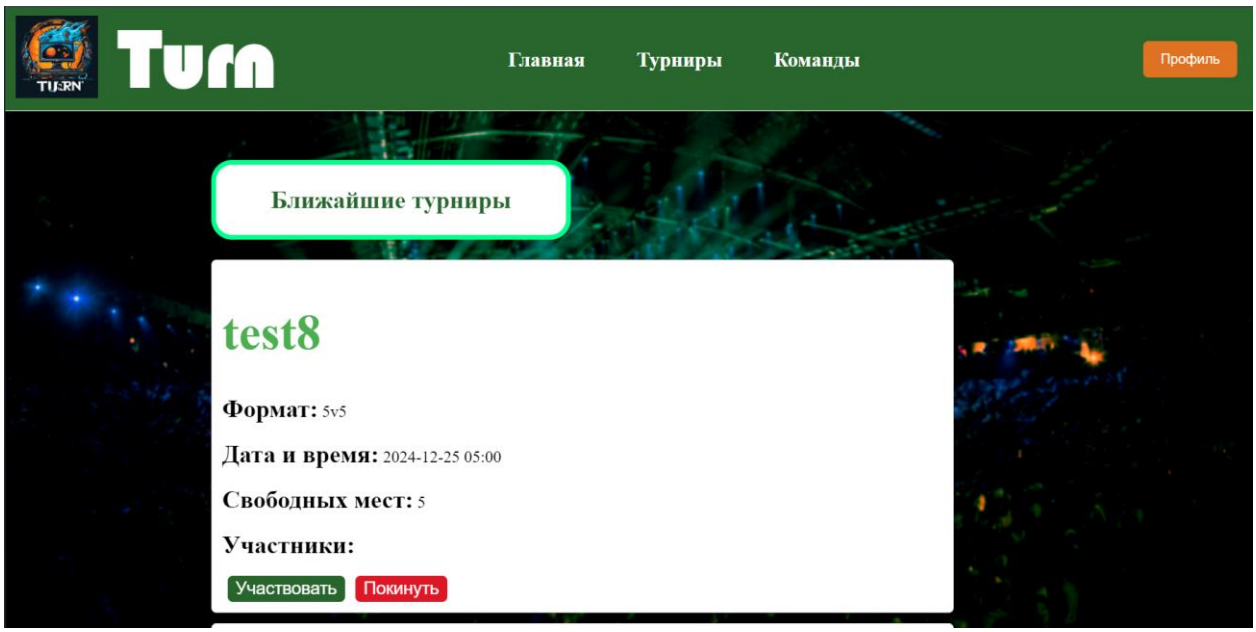


Рисунок 11 – Страница команд, список ближайших турниров.

3.4 Требования ко входных данным

При регистрации поля никнейм, имя, фамилия, почта и пароль не могут быть пустыми, а также в почте должен присутствовать символ «@» и пароль не должен содержать менее 8 символов. При авторизации вводятся только почта и пароль, требования к которым идентичные. При создании турнира каждое из полей так же не может быть пустым, а также поле дата не может содержать прошедшую дату, поле время, при условии, что дата сегодняшняя, не может быть поставлена ранее чем, ближайшие 2 часа от текущего времени, а поле формат обязательно должно содержать букву «v».

4. Руководство программиста

4.1 Организация ввода и вывода данных в программе

В данном проекте ввод данных осуществляются путём введения значений в поля формы, а их отправка серверу происходит после подтверждения введённых значений нажатием кнопки.

В большей части форм данного приложения реализован реактивный подход, так как этот подход в формах обеспечивает мгновенное обновление пользовательского интерфейса при изменении данных, улучшая интерактивность и удобство взаимодействия с приложением. Этот подход упрощает валидацию и управление состоянием форм, позволяя динамически адаптироваться к действиям пользователя.

В качестве сервера был выбран json-server, так как он является отличным решением для хранения данных на этапе разработки, так как позволяет быстро настроить REST API без необходимости создания полноценного серверного приложения. Он поддерживает формат JSON, широко используемый для обмена данными, и обеспечивает гибкость для последующего перехода на полноценную серверную инфраструктуру.

Пример ввода данных расположен на рисунке ниже.

```
<div id="fon">
    <div class="team-creation-container">
        <h1>Создание команды</h1>
        <form [formGroup]="teamForm" (ngSubmit)="onSubmit()">
            <div>
                <label for="name">Название команды</label>
                <input id="name" formControlName="name" type="text" placeholder="Введите название команды" />
            </div>

            <div>
                <label for="gameType">Тип игры</label>
                <input id="gameType" formControlName="gameType" type="text" placeholder="Введите тип игры" />
            </div>

            <div>
                <label for="members">Участники</label>
                <input
                    id="members"
                    formControlName="members"
                    type="text"
                    placeholder="Введите участников через запятую (макс. 5)"
                />
            </div>

            <button type="submit" [disabled]="teamForm.invalid">Создать команду</button>
        </form>
    </div>

</div>
```

Рисунок 12 – Примере ввода данных в форме создания команд, файл teams.compoent.ts

4.2 Описание компонентов

Данного приложение использует новую версию Angular, которая поддерживает структуру standalone: true, поэтому она не нуждается в обязательном использовании модулей. Следовательно, опишем все компоненты системы, а именно: какую роль они выполняют в системе, для чего они нужны и с чем взаимодействуют.

Основные компоненты системы:

1) *MainPageComponent:* отображает основную информацию о приложении и является стартовой точкой работы нашего приложения.

- Этот компонент представляет собой контейнер с ознакомительной информацией о сайте, а также маршруты для перенаправления на основные функции страницы через кнопки.
- Взаимодействует с TeamsComponent и TournamentComponent.

2) *AuthComponent:* содержит сервисы для управления состоянием пользователя (авторизирован/нет) и управления данными пользователя, а также содержит RegistrationComponent и AutorizationComponent.

- Этот компонент представляет собой группировку логику для организации регистрации и авторизации.
- Взаимодействует с RegistrationComponent и AutorizationComponent

3) *AutorizationComponent:* представляет собой функционал для авторизации пользователей.

- Этот компонент содержит форму для авторизации пользователя, а также сервис для обработки данных формы.
- Взаимодействует с AuthComponent.

4) *RegistrationComponent:* представляет собой функционал для регистрации пользователя.

- Этот компонент содержит форму для регистрации пользователя, а также сервис для обработки данных пользователя.
- Взаимодействует с AuthComponent.

5) *ProfileComponent:* представляет собой контейнер для вывода информации о пользователе.

- Этот компонент содержит контейнер, в котором описаны все основные данные о пользователе, его командах и турнирах после авторизации.
- Взаимодействует с TeamsComponent и TournamentsComponent

6) *TeamsComponent:* представляет собой функционал для создания команд.

- Этот компонент содержит формы для создания объектов команд пользователя, а также сервис для управления информацией о команде и добавление ID команды каждому её участнику.
- Взаимодействует с AuthComponent.

7) *TournamentsComponent:* представляет собой функционал для создания турниров и записи на них.

- Этот компонент содержит формы для создания объектов турниров пользователя, а также сервис для управления информацией о турнире и добавление команды пользователя на данный турнир.
- Взаимодействует с AuthComponent и TournComponent.

Переиспользуемые компоненты:

1) *HeaderComponent:* является «шапкой» сайта и реализует навигационное меню на сайте, с помощью которого пользователь может удобно перемещаться между страницами, присутствует на протяжении всей работы приложения вверху страницы.

2) *FooterComponent:* является «подвалом» сайта и содержит все необходимые контактные данные о администрации сайта, присутствует на протяжении всей работы приложения снизу страницы.

3) *TournComponent:* является переиспользуемым компонентом для отображения информации о турнирах, а также сервис для записи на каждый отдельный турнир.

4.3. Модели и сервисы

Для начала опишем модели всех объектов данного приложения, так как именно они будут хранится на сервере. Первостепенно опишем модель пользователя, которая будет содержать его ID, никнейм, имя, фамилия, почту, пароль, а также массив ID команда, в которых пользователь принимает участие, либо является их лидером, и массив турниров, рисунок 13.

```
export interface User {
    id?: number;          // Идентификатор
    nickname: string,
    name: string;        // Имя пользователя
    surname: string;
    email: string;       // Email пользователя
    password: string;    // Пароль пользователя
    teams: string[],
    turns: string[]
  }
```

Рисунок 13 – Модель пользователя user.model.ts

Так же опишем модель команды, которая содержит следующие свойства: ID команды, название, тип игры, массив ID участников команды и ID капитана команды, рисунок 14.

```
export interface Team {
    id?: number;              // Идентификатор команды
    name: string;             // Название команды
    gameType: string;         // Тип игры
    members: string[];        // Участники команды
    creatorId?: number;        // ID капитана команды
  }
```

Рисунок 14 – Модель команды team.model.ts

Так же опишем модель турнира, которая будет содержать следующую информацию о турнире: ID турнира, название, дата и время турнира, формат,

список ID команд участников, максимальное количество участников и ID пользователя, создавшего данный турнир, рисунок 15.

```
export interface Tournament {
    id?: number;          // Идентификатор турнира
    name: string;         // Название турнира
    date: string;         // Дата турнира в формате ISO
    time: string;         // Время турнира
    format: string;       // Формат турнира
    participants: string[]; // Список ID команд-участников
    max_num: number;
    creatorId?: number;    // ID пользователя, создавшего турнир
}
```

Рисунок 15 – Модель турнира tournament.model.ts

На основе этих трёх моделей опишем сервисы для сохранения и выгрузки информации о данных объектах на сервер.

Для начала перейдём к сервису user.service.ts по работе с сервером db.json. На данный момент на сервере будут хранится объекты модели пользователя user.service.ts, а связующим звеном между ними будет являться файл user.service.ts, рисунок 16, 17. Он реализует функции получения пользователей, добавления, обновления, авторизации и так далее.

```
export class UserService {
  private apiUrl = 'http://localhost:3000/users'; // URL JSON Server
  private currentUserId: number | undefined = undefined;

  constructor(private http: HttpClient) {}

  // Получить всех пользователей
  getUsers(): Observable<User[]> {
    return this.http.get<User[]>(this.apiUrl);
  }

  // Добавить нового пользователя
  addUser(user: User): Observable<User> {
    return this.http.post<User>(this.apiUrl, user);
  }

  login(email: string, password: string): Observable<User | null> {
    return this.http.get<User[]>(`${this.apiUrl}?email=${email}`).pipe(
      switchMap(async (users) => {
        const user = users[0];
        if (user && (await bcrypt.compare(password, user.password))) {
          return user;
        }
        return null;
      })
    );
  }
```

Рисунок 16 - Файл user.service.ts (1 часть)

```
setCurrentUserId(id: number): void {
  this.currentUserId = id;
}

// Получить пользователя по ID (тип string)
getUserById(userId: string): Observable<User> {
  return this.http.get<User>(`${this.apiUrl}/${userId}`);
}

// Получить текущего пользователя
getCurrentUser(): Observable<User | null> {
  if (this.currentUserId !== null) {
    return this.http.get<User>(`${this.apiUrl}/${this.currentUserId}`);
  }
  return of(null);
}

updateUser(user: User): Observable<User> {
  return this.http.put<User>(`http://localhost:3000/users/${user.id}`, user);
}

updateUserTeams(userId: string, teamId: number): Observable<User> {
  return this.getUserById(userId).pipe(
    switchMap((user) => {
      const updatedTeams = [...(user.teams || []), teamId]; // Обновляем массив команд
      return this.http.patch<User>(`${this.apiUrl}/${userId}`, { teams: updatedTeams });
    })
  );
}
```

Рисунок 17 - Файл user.service.ts (2 часть)

Сервис для работы с командами, как и в прочем любой другой сервис представляет собой проводника между сервером и приложением. Он содержит такие методы, как: получение всех команд, создание новой команды, удаление команды, обновление данных команды, обновление данных команды, получение команды по ID и получение нескольких команд по их ID. Для осуществления стандартных запросов к серверу в данном файле подключим класс «HttpClient», который реализует функции: get, post, delete, patch, рисунок 17, 18.

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable, forkJoin } from 'rxjs';
import { Team } from '../team.model';
import { User } from '../../../auth/models/user.model';

@Injectable({
  providedIn: 'root',
})
export class TeamService {
  private apiUrl = 'http://localhost:3000/teams'; // URL для хранения данных команд

  constructor(private http: HttpClient) {}

  // Получение всех команд
  getTeams(): Observable<Team[]> {
    return this.http.get<Team[]>(this.apiUrl);
  }

  // Создание новой команды
  createTeam(team: Team): Observable<Team> {
    return this.http.post<Team>(this.apiUrl, team);
  }

  // Удаление команды
  deleteTeam(id: number): Observable<void> {
    return this.http.delete<void>(`${this.apiUrl}/${id}`);
  }
```

Рисунок 17 – Файл сервиса team.service.ts (1 часть)

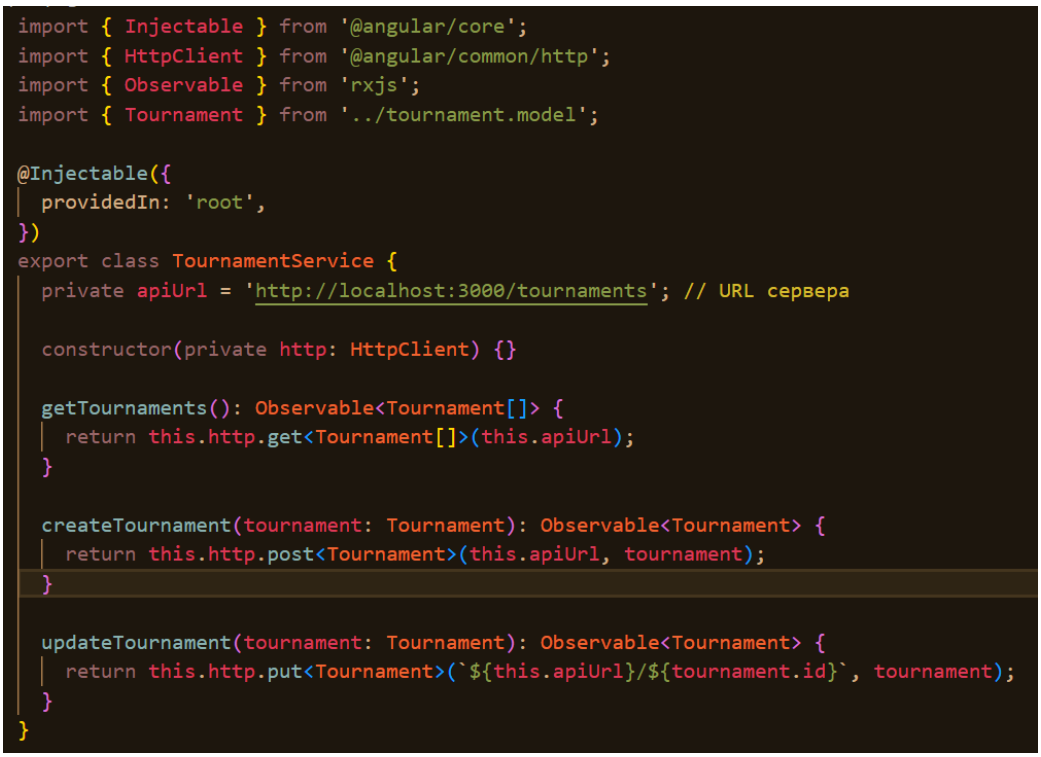
```
// Обновление данных команды
updateTeam(teamId: number, updatedData: Partial<Team>): Observable<Team> {
  return this.http.patch<Team>(`${this.apiUrl}/${teamId}`, updatedData);
}

// Получить команду по ID
getTeamById(teamId: string): Observable<Team> {
  return this.http.get<Team>(`${this.apiUrl}/${teamId}`);
}

// Получить несколько команд по их ID
getTeamsByIds(ids: string[]): Observable<Team[]> {
  const query = ids.map((id) => `id=${id}`).join('&');
  return this.http.get<Team[]>(`${this.apiUrl}?${query}`);
}
```

Рисунок 18 - Файл сервиса team.service.ts (2 часть)

Если говорить о сервисе для турниров tournament.service.ts, то он всё так же реализует функции для обмена данными между основным приложением и сервером. Он реализует такие функции, как: получение всех турниров, создание турнира и обновления турнира, рисунок 19.



```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Tournament } from '../tournament.model';

@Injectable({
  providedIn: 'root',
})
export class TournamentService {
  private apiUrl = 'http://localhost:3000/tournaments'; // URL сервера

  constructor(private http: HttpClient) {}

  getTournaments(): Observable<Tournament[]> {
    return this.http.get<Tournament[]>(this.apiUrl);
  }

  createTournament(tournament: Tournament): Observable<Tournament> {
    return this.http.post<Tournament>(this.apiUrl, tournament);
  }

  updateTournament(tournament: Tournament): Observable<Tournament> {
    return this.http.put<Tournament>(`${this.apiUrl}/${tournament.id}`, tournament);
  }
}
```

Рисунок 19 - Файл сервиса tournament.service.ts

Функции каждого из данных сервисов используются по всему коду в различных файлах, а новые функции добавляются по мере их надобности. Они используются в основном для получения объектов с сервера и использования их в нуждах функционала проекта. Также они используются для обновления данных об объектах, которые хранит сервер.

4.4 Структура программы.

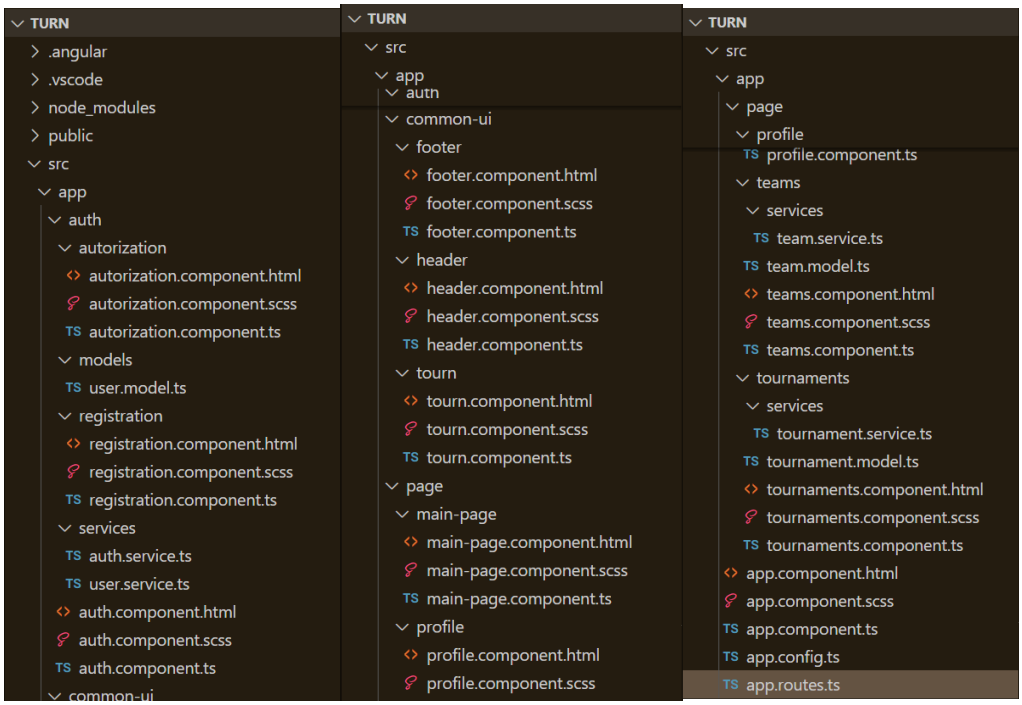


Рисунок 20 – Структура программы (1 часть).

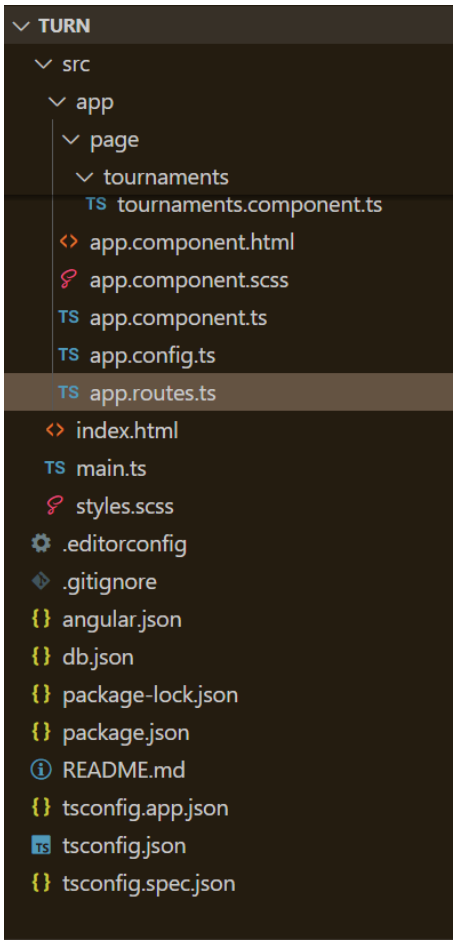


Рисунок 21 - Структура программы (2 часть).

4.5 Описание взаимодействия между сущностями web-приложения

Для осуществления логического взаимодействия между компонентами в приложении используются сервисы, которые передают информации от одного компонента к другому, чтобы создать слаженную систему.

Для осуществления маршрутного взаимодействия в приложении сервис Router. Все пути для него написаны в файле app.routes.ts, рисунок 22.

Рисунок 22 – Файл app.routes.ts

В файлах сервисов он применяется с использованием метода .navigate(), в котором прописывается путь к компоненту, на который нужно сделать переход, рисунок 23.

```
this.router.navigate(['/prof']); // Перенаправление на список команд
```

Рисунок 23 – Пример перехода к другому компоненту.

Заключение

В ходе разработки веб-приложения для организации турниров по видеоиграм была успешно реализована функциональность, позволяющая пользователям удобно и эффективно управлять процессами, связанными с проведением турниров. Приложение предоставляет пользователям возможность регистрироваться, создавать турниры, создавать команды, а также принимать участие в турнирах разных форматов. Это создает благоприятные условия для организации и участия в турнирах, делая процесс максимально доступным и комфортным.

В процессе реализации проекта все поставленные задачи были выполнены в полном объеме. Мы разработали архитектуру приложения, которая обеспечивает надежное функционирование всех процессов. С помощью интеграции различных технологий и инструментов, таких как базы данных, серверные и клиентские технологии, было достигнуто высокое качество разработки, что, в конечном итоге, позволяет пользователю сосредоточиться на организации турниров, не беспокоясь о технических аспектах.

Интуитивно понятный пользовательский интерфейс стал одним из ключевых аспектов нашего приложения. Систематизированное меню, ясные инструкции и удобная навигация позволяют пользователям на интуитивном уровне выполнять все необходимые действия без затруднений.

Данное приложение служит важным инструментом для любителей видеоигр и организаторов турниров, обеспечивая платформу для взаимодействия и соперничества. Оно позволяет игрокам находить новые турниры, управлять своим участием, а организаторам — эффективно управлять процессами, связанными с проведением мероприятий. В результате, приложение не только способствует развитию киберспорта, но и объединяет игроков, создавая сообщества и поддерживая дух соревнования в мире видеоигр.

Список литературы

1) Руководство по Angular [электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/246881/ (Дата обращения: 22.11.2024)

2) Руководство по TypeScript [электронный ресурс]. – Режим доступа: https://habr.com/ru/companies/macloud/articles/557996/ (Дата обращения: 24.11.2024)

3) Форум с вопросами и ответами для разработчиков [электронный ресурс]. – Режим доступа: https://stackoverflow.com/ (Дата обращения: 27.11.2024)

4) Семь принципов создания современных веб-приложений [электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/242429/ (Дата обращения: 27.11.2024)

5) Шпаргалка по JSON-server [электронный ресурс]. – Режим доступа: https://my-js.org/docs/cheatsheet/json-server/ (Дата обращения: 30.11.2024)

6) Создание фейкового REST API с помощью json-server [электронный ресурс]. – Режим доступа: https://code.tutsplus.com/ru/fake-rest-api-up-and-running-using-json-server--cms-27871t (Дата обращения: 01.12.2024)

Приложение

Рисунок 24 – Форма регистрации

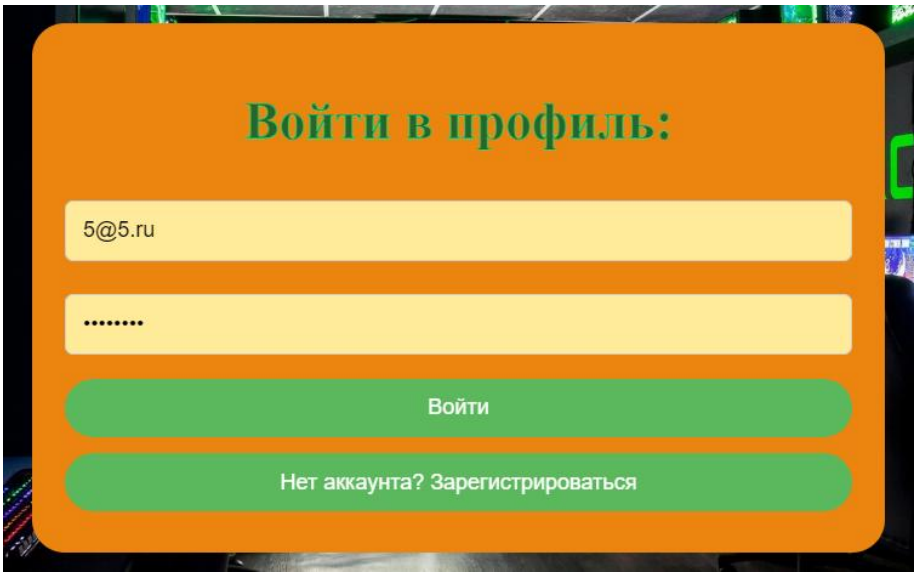
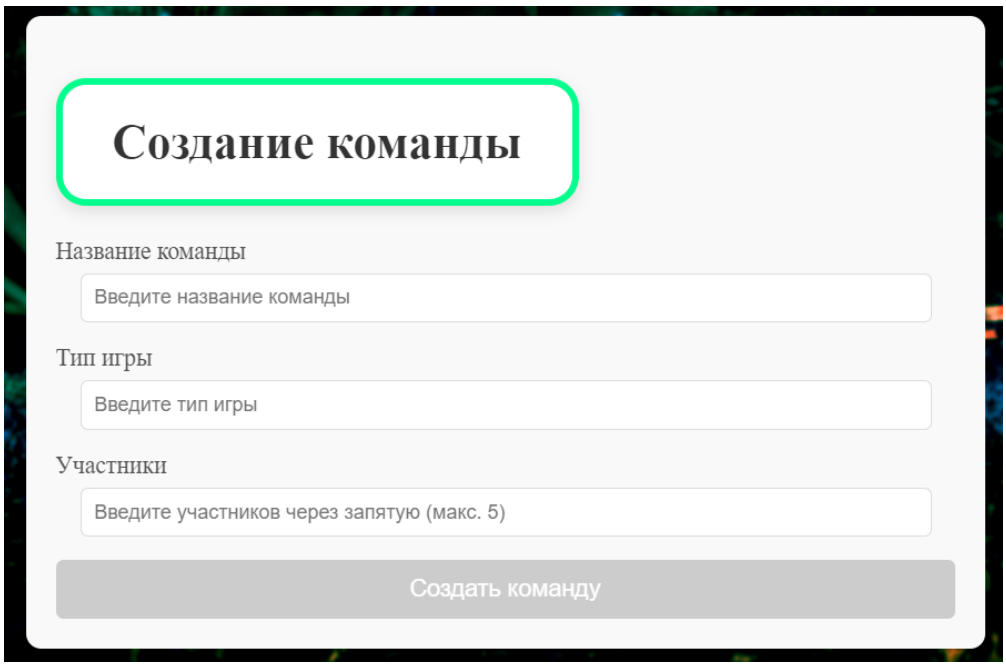


Рисунок 25 – Форма авторизации



Рисинок 26 – Форма создание команды

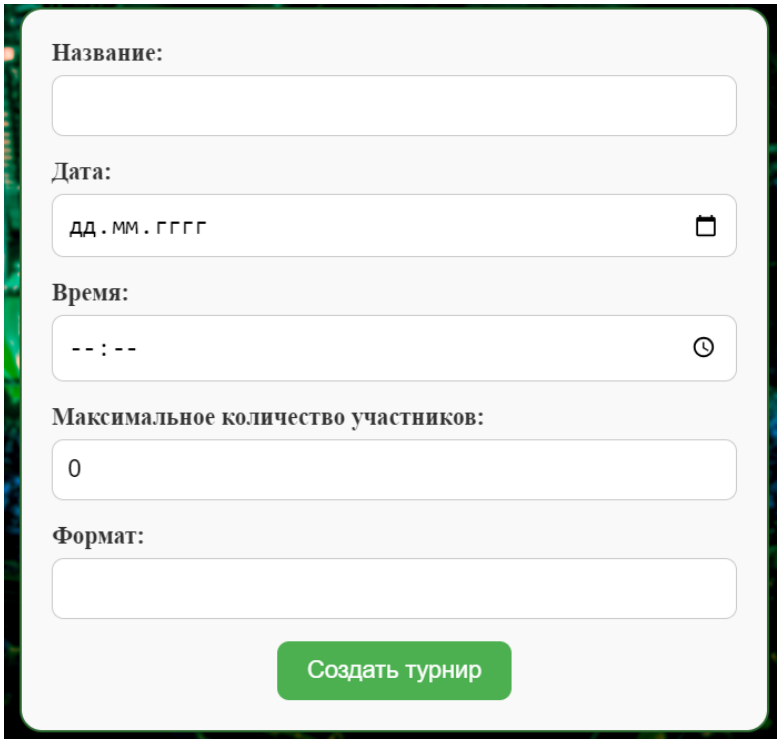


Рисунок 27 – Форма создания турнира

authorization.component.ts:

```
import { Component, EventEmitter, Output, OnInit } from '@angular/core';
import { FormGroup, FormControl, Validators, FormsModule, ReactiveFormsModule } from
'@angular/forms';
import { CommonModule } from '@angular/common';
import { Router } from '@angular/router';
import { UserService } from '../services/user.service';
import { AuthService } from '../../auth/services/auth.service';
```

```
@Component({
  selector: 'app-autorization',
  standalone: true,
  imports: [CommonModule, FormsModule, ReactiveFormsModule],
  templateUrl: './autorization.component.html',
  styleUrl: './autorization.component.scss'
})
export class AutorizationComponent implements OnInit{
  @Output() toggle = new EventEmitter<void>();

  switchToRegister() {
    this.toggle.emit();
  }

  logform!: FormGroup;

  constructor(private userService: UserService, private router: Router, private authService:
AuthService) {}

  ngOnInit(): void {
    // Инициализация формы авторизации
    this.logform = new FormGroup({
      email: new FormControl('', [Validators.required, Validators.email]),
      password: new FormControl('', [Validators.required, Validators.minLength(8)]),
    });
  }

  onSubmit(): void {
    if (this.logform.invalid) {
      return;
    }

    const { email, password } = this.logform.value;

    // Проверка пользователя в базе данных
    this.userService.login(email, password).subscribe({
      next: (user) => {
        if (user) {
          console.log('Login successful:', user);
          // Перенаправление на главную страницу
          if (user.id !== undefined) {
            this.userService.setCurrentUserId(user.id); // Проверка на `undefined`
          }
          this.authService.login()
          this.router.navigate(['/prof']);
        } else {
          alert('Неверный email или пароль');
        }
      },
      error: (error) => {
```

```
        console.error('Error during login:', error);
        alert('Произошла ошибка при авторизации');
      },
      complete: () => {
        console.log('Login process completed.');
      },
    });

  }
}
```

user.model.ts:

```
export interface User {
    id?: number;        // Идентификатор
    nickname: string,
    name: string;      // Имя пользователя
    surname: string;
    email: string;     // Email пользователя
    password: string;  // Пароль пользователя
    teams: string[],
    turns: string[]
  }
```

registration.component.ts:

```
import { Component, EventEmitter, Output, OnInit } from '@angular/core';
  import { FormGroup, FormControl, Validators, ReactiveFormsModule,
FormsModule } from '@angular/forms';
  import { UserService } from '../services/user.service';
  import { User } from '../models/user.model';
  import { CommonModule } from '@angular/common';
  import * as bcrypt from 'bcryptjs'; // Импорт bcryptjs
  @Component({
    selector: 'app-registration',
    standalone: true,
    imports: [CommonModule, FormsModule, ReactiveFormsModule],
    templateUrl: './registration.component.html',
    styleUrls: ['./registration.component.scss']
  })
  export class RegistrationComponent implements OnInit {
```

```
  @Output() toggle = new EventEmitter<void>();

  // Метод для переключения на страницу входа
  switchToLogin() {
    this.toggle.emit();
  }

  regform!: FormGroup;  // Экземпляр формы

  constructor(private userService: UserService) {}

  ngOnInit(): void {
    // Инициализация формы с валидаторами
    this.regform = new FormGroup({
      nickname: new FormControl('', [Validators.required]),
      name: new FormControl('', [Validators.required]),
      surname: new FormControl('', [Validators.required]),
      email: new FormControl('', [Validators.required, Validators.email]),
      password: new FormControl('', [Validators.required,
Validators.minLength(8)])
    });
  }



  // Метод для отправки данных формы
  async onSubmit() {
    if (this.regform.invalid) {
      return;  // Если форма не валидна, ничего не отправляем
```

```
    }

    const { nickname, name, surname, email, password } = this.regform.value;
    // Хэширование пароля
    const hashedPassword = await bcrypt.hash(password, 10);

    const newUser: User = {
      nickname,
      name,
      surname,
      email,
      password: hashedPassword,
      teams: [],
      turns: []
    };

    // Отправка нового пользователя на сервер
    this.userService.addUser(newUser).subscribe((addedUser) => {
      console.log('User added successfully:', addedUser);
    });

    // Очистка формы после отправки
    this.regform.reset();
  }
 }
```

auth.service.ts:

```
export class AuthService {
   private isAuthenticated;
   constructor() {
```

```
    if (localStorage.getItem('user') == null) {
      this.isAuthenticated = false;
    }
    else {
      this.isAuthenticated = true;
    }
  }

  login() {
    this.isAuthenticated = true;
  }

  logout() {
    this.isAuthenticated = false;
    window.localStorage.clear();
  }

  isLoggedIn(): boolean {

    return this.isAuthenticated;
  }
}
```

user.service.ts:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable, of } from 'rxjs';
import { User } from '.././models/user.model';
import { map, switchMap} from 'rxjs/operators';
import * as bcrypt from 'bcryptjs'; // Импорт bcryptjs
```

```
@Injectable({
  providedIn: 'root',
})
export class UserService {
  private apiUrl = 'http://localhost:3000/users'; // URL JSON Server
  private currentUserId: number | undefined = undefined;

  constructor(private http: HttpClient) {}

  // Получить всех пользователей
  getUsers(): Observable<User[]> {
    return this.http.get<User[]>(this.apiUrl);
  }

  // Добавить нового пользователя
  addUser(user: User): Observable<User> {
    return this.http.post<User>(this.apiUrl, user);
  }

  login(email: string, password: string): Observable<User | null> {
    return this.http.get<User[]>(`${this.apiUrl}?email=${email}`).pipe(
      switchMap(async (users) => {
        const user = users[0];
        if (user && (await bcrypt.compare(password, user.password))) {
          return user;
        }
        return null;
      })
```

```
    );
  }

  setCurrentUserId(id: number): void {
    this.currentUserId = id;
  }

  // Получить пользователя по ID (тип string)
  getUserById(userId: string): Observable<User> {
    return this.http.get<User>(`${this.apiUrl}/${userId}`);
  }

  // Получить текущего пользователя
  getCurrentUser(): Observable<User | null> {
    if (this.currentUserId !== null) {
      return this.http.get<User>(`${this.apiUrl}/${this.currentUserId}`);
    }
    return of(null);
  }

  updateUser(user: User): Observable<User> {
    return this.http.put<User>(`http://localhost:3000/users/${user.id}`, user);
  }

  updateUserTeams(userId: string, teamId: number): Observable<User> {
    return this.getUserById(userId).pipe(
      switchMap((user) => {
        const updatedTeams = [...(user.teams || []), teamId]; // Обновляем массив
команд
```

```
      return this.http.patch<User>(`${this.apiUrl}/${userId}`, { teams: updatedTeams });
    })
  );
 }
}
```

auth.component.ts:

```
import { Component } from '@angular/core';
import { RegistrationComponent } from './registration/registration.component';
import { AutorizationComponent } from './autorization/autorization.component';
import { CommonModule } from '@angular/common';
import { Router } from '@angular/router';
import { UserService } from './services/user.service';

@Component({
 selector: 'app-auth',
 standalone: true,
 imports: [RegistrationComponent, AutorizationComponent, CommonModule],
 templateUrl: './auth.component.html',
 styleUrl: './auth.component.scss'
})
export class AuthComponent {
 isRegistering = true;

 toggleForm(isRegistering: boolean) {
  this.isRegistering = isRegistering;
 }
}
```

header.component.ts:

```
import { Component } from '@angular/core';
import { Router, RouterLink } from '@angular/router';
import { AuthService } from '../../auth/services/auth.service';

import { CommonModule } from '@angular/common';



@Component({
  selector: 'app-header',
  imports: [CommonModule, RouterLink],
  templateUrl: './header.component.html',
  styleUrl: './header.component.scss'
})
export class HeaderComponent {
  constructor(private authService: AuthService) { }



  auth() {
    return this.authService.isLoggedIn()
  }
}
```

footer.component.ts:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-footer',
  imports: [],
  templateUrl: './footer.component.html',
```

```
  styleUrl: './footer.component.scss'
})
export class FooterComponent {

}
```

tourn.component.ts:

```
import { Component, Input, OnInit } from '@angular/core';
import { CommonModule,  } from '@angular/common';
import { Tournament } from '../../page/tournaments/tournament.model';
import { UserService } from '../../auth/services/user.service';
import { TeamService } from '../../page/teams/services/team.service';
import { Team } from '../../page/teams/team.model';
import { TournamentService } from '../../page/tournaments/services/tournament.service';
import { switchMap } from 'rxjs/operators';
import { forkJoin, of } from 'rxjs';
import { catchError } from 'rxjs/operators';


@Component({
  selector: 'app-tourn',
  imports: [CommonModule],
  templateUrl: './tourn.component.html',
  styleUrl: './tourn.component.scss'
})
export class TournComponent implements OnInit {
  @Input() tournament!: Tournament; // Турнир, который нужно отобразить
  participantTeams: Team[] = [];
  constructor(
```

```
  private userService: UserService,
  private teamService: TeamService,
  private tournamentService: TournamentService
) {}

ngOnInit(): void {
  this.loadParticipantTeams();
}

loadParticipantTeams(): void {
  if (this.tournament.participants.length === 0) {
    this.participantTeams = []; // Если участников нет, оставляем массив пустым
    return;
  }

  this.teamService.getTeamsByIds(this.tournament.participants).subscribe({
    next: (teams) => {
      this.participantTeams = teams; // Сохраняем команды-участники
    },
    error: (err) => {
      console.error('Ошибка при загрузке команд-участников:', err);
    }
  });
}

onParticipate(): void {
  this.userService.getCurrentUser().subscribe((user) => {
    if (user) {
      const userTeamIds = user.teams || [];
```

```
// Загружаем команды пользователя
this.teamService.getTeamsByIds(userTeamIds).subscribe((teams) => {
  // Фильтруем команды, где пользователь является создателем
  const userCreatedTeams = teams.filter(
    (team) => String(team.creatorId) === String(user.id)
  );
  console.log('Все команды пользователя:', teams);
  console.log('ID текущего пользователя:', user.id);
  if (userCreatedTeams.length === 0) {
    alert('Вы не являетесь лидером ни одной из команд в которых вы состоите');
    return;
  }

  // Проверяем, зарегистрировал ли пользователь уже одну из своих команд
  const userCreatedTeamAlreadyInTournament = this.tournament.participants.some((participantId) =>
    userCreatedTeams.some((team) => String(team.id) === String(participantId))
  );

  if (userCreatedTeamAlreadyInTournament) {
    alert('Вы уже добавили одну из своих команд в этот турнир');
    return;
  }

  // Предлагаем выбрать команду
```

```
const selectedTeamIndex = prompt(
  `Выберите команду для участия:\n${userCreatedTeams
    .map((team, index) => `${index + 1}. ${team.name}`)
    .join('\n')}`
);

const selectedIndex = parseInt(selectedTeamIndex || '', 10) - 1;

if (!isNaN(selectedIndex) && userCreatedTeams[selectedIndex]) {
  const selectedTeam = userCreatedTeams[selectedIndex];

  // Проверяем, является ли команда уже участником турнира
  if (this.tournament.participants.includes(String(selectedTeam.id!))) {
    alert('Данная команда уже является участником данного турнира');
  } else {
    if (this.tournament.max_num === 0){
      alert('Все места уже заняты. Стоит поискать другой турнир для участия.')
    } else{

      // Добавляем ID турнира в turns всех участников команды
      this.addTournamentToTeamMembers(selectedTeam.id!.toString(), String(this.tournament.id!));

      // Добавляем ID команды в список участников турнира
      this.tournament.participants.push(selectedTeam.id!.toString());

      this.tournament.max_num = this.tournament.max_num - 1;
      // Обновляем турнир на сервере
```

```
            this.tournamentService.updateTournament(this.tournament).subscribe({
              next: () => {
                alert(`Команда "${selectedTeam.name}" теперь учавствует в данном трунире. За 2 часа до начала турнира с вами свяжутся наши администраторы с нашего официального email по контактной информации лидера вашей команды. Желаем вам удачного проведения турнира!`);
              },
              error: (err: any) => {
                console.error('Ошибка при обновлении турнира:', err);
              },
            });
          }
        }
      }
    });
  }
}

addTournamentToTeamMembers(teamId: string, tournamentId: string): void {
  this.teamService.getTeamById(teamId).subscribe((team) => {
    if (team && team.members) {
      // Формируем массив запросов для обновления участников команды
      const memberUpdates = team.members.map((memberId) => {
        return this.userService.getUserById(memberId).pipe(
          switchMap((user) => {
            // Проверяем, есть ли турнир уже в turns
            if (user.turns.includes(tournamentId)) {
              console.log(`Турнир уже добавлен пользователю ${user.id}`);
```

```
            return of(user); // Возвращаем текущего пользователя без изменений
          }
          const updatedTurns = [...user.turns, tournamentId];
          return this.userService.updateUser({ ...user, turns: updatedTurns }).pipe(
            catchError((error) => {
              console.error(`Ошибка обновления пользователя ${user.id}:`, error);
              return of(null); // Возвращаем null в случае ошибки
            })
          );
        })
      );
    });

    // Выполняем обновление всех участников параллельно
    forkJoin(memberUpdates).subscribe({
      next: (results) => {
        console.log('Участники команды успешно обновлены:', results);
      },
      error: (err) => {
        console.error('Ошибка при обновлении участников команды:', err);
      },
    });
  }
 });
}

leave(): void {
  this.userService.getCurrentUser().subscribe((user) => {
    if (user) {
```

```
const userTeamIds = user.teams || [];

// Загружаем команды пользователя
this.teamService.getTeamsByIds(userTeamIds).subscribe((teams) => {
  // Фильтруем команды, где пользователь является создателем
  const userCreatedTeams = teams.filter(
    (team) => String(team.creatorId) === String(user.id)
  );

  // Проверяем, зарегистрировал ли пользователь уже одну из своих команд
  const userCreatedTeamInTournament = userCreatedTeams.find((team) =>
    this.tournament.participants.includes(String(team.id))
  );

  if (!userCreatedTeamInTournament) {
    alert('Вы не регистрировали ни одну из своих команд на данный турнир');
    return;
  }

  const selectedTeam = userCreatedTeamInTournament;

  // Удаляем ID команды из списка участников турнира
  this.tournament.participants = this.tournament.participants.filter(
    (participantId) => participantId !== String(selectedTeam.id)
  );

  // Увеличиваем доступное количество мест
```

```
        this.tournament.max_num = this.tournament.max_num + 1;

        // Удаляем ID турнира из turns всех участников команды
        this.removeTournamentFromTeamMembers(selectedTeam.id!.toString(),
String(this.tournament.id!));

        // Обновляем турнир на сервере
        this.tournamentService.updateTournament(this.tournament).subscribe({
          next: () => {
            alert(`Команда "${selectedTeam.name}" успешно покинула данный
турнир.`);
          },
          error: (err: any) => {
            console.error('Ошибка при обновлении турнира:', err);
          },
        });
      });
    }
  });
 }

 removeTournamentFromTeamMembers(teamId: string, tournamentId: string):
void {
   this.teamService.getTeamById(teamId).subscribe((team) => {
     if (team && team.members) {
       // Формируем массив запросов для обновления участников команды
       const memberUpdates = team.members.map((memberId) => {
         return this.userService.getUserById(memberId).pipe(
           switchMap((user) => {
```

```
            // Удаляем турнир из turns, если он существует
            if (!user.turns.includes(tournamentId)) {
              console.log(`Турнир отсутствует у пользователя ${user.id}`);
              return of(user); // Возвращаем текущего пользователя без изменений
            }
            const updatedTurns = user.turns.filter((turnId) => turnId !==
tournamentId);
            return this.userService.updateUser({ ...user, turns: updatedTurns }).pipe(
              catchError((error) => {
                console.error(`Ошибка обновления пользователя ${user.id}:`, error);
                return of(null); // Возвращаем null в случае ошибки
              })
            );
          })
        );
      });

      // Выполняем обновление всех участников параллельно
      forkJoin(memberUpdates).subscribe({
        next: (results) => {
          console.log('Участники команды успешно обновлены:', results);
        },
        error: (err) => {
          console.error('Ошибка при обновлении участников команды:', err);
        },
      });
    }
  });
}
```

```
}
```

main-page.component.ts:

```
import { Component } from '@angular/core';
import { AuthService } from '../../auth/services/auth.service';
import { CommonModule } from '@angular/common';
import { Router, RouterLink} from '@angular/router';
@Component({
  selector: 'app-main-page',
  imports: [CommonModule, RouterLink],
  templateUrl: './main-page.component.html',
  styleUrl: './main-page.component.scss'
})
export class MainPageComponent {

  constructor(private authService:AuthService, private router: Router){}
  auth() {
    return this.authService.isLoggedIn()
  }

  go_toAuth(){
    this.router.navigate(['/auth']);
    alert('Сначала следует пройти регистрацию/авторизацию');
  }
}
```

profile.component.ts:

```
import { Component, OnInit} from '@angular/core';
import { CommonModule } from '@angular/common';
```

```
import { UserService } from '../../auth/services/user.service';
import { User } from '../../auth/models/user.model';
import { AuthService } from '../../auth/services/auth.service';
import { Router } from '@angular/router';
import { Team } from '../teams/team.model';
import { Tournament } from '../tournaments/tournament.model';
import { TeamService } from '../teams/services/team.service';
import { TournamentService } from '../tournaments/services/tournament.service';

@Component({
  selector: 'app-profile',
  standalone: true,
  imports: [CommonModule],
  templateUrl: './profile.component.html',
  styleUrl: './profile.component.scss'
})


export class ProfileComponent implements OnInit {
  currentUser: User | null = null; // Данные текущего пользователя
  loading: boolean = true; // Флаг для отображения загрузки
  error: string | null = null; // Сообщение об ошибке
  userTeams: Team[] = [];
  userTurns: Tournament[] = [];
  userNickname: string | undefined; // ID текущего пользователя

  constructor(
    private userService: UserService,
    private authService: AuthService,
```

```
    private router: Router,
    private teamService: TeamService,
    private turnService: TournamentService
  ) {}

  ngOnInit(): void {
    this.loadUser();
  }

  loadUser(): void {
    this.userService.getCurrentUser().subscribe({
      next: (user) => {
        if (user) {
          this.currentUser = user;
          this.loadUserTeams(); // Загружаем команды, где пользователь является участником
          this.loadUserTurns();
        } else {
          this.error = 'Пользователь не найден.';
        }
        this.loading = false; // Отключаем индикатор загрузки
      },
      error: (err) => {
        this.error = 'Произошла ошибка при загрузке данных пользователя.';
        console.error('Error loading user:', err);
        this.loading = false; // Отключаем индикатор загрузки
      },
    });
  }
```

```
  private loadUserTeams(): void {
    if (!this.currentUser || !this.currentUser.teams || this.currentUser.teams.length === 0) {
      this.userTeams = []; // Если у пользователя нет команд, инициализируем пустой массив
      return;
    }

    const teamIds = this.currentUser.teams; // Получаем массив ID команд пользователя

    this.teamService.getTeams().subscribe({
      next: (teams) => {
        // Фильтруем команды по ID из массива teams пользователя
        this.userTeams = teams.filter((team) => teamIds.includes(team.id?.toString() || ''));
      },
      error: (err) => {
        this.error = 'Ошибка при загрузке команд.';
        console.error('Error loading teams:', err);
      },
    });
  }

  private loadUserTurns(): void {
    if (!this.currentUser || !this.currentUser.turns || this.currentUser.turns.length === 0) {
```

```
    this.userTurns = []; // Если у пользователя нет команд, инициализируем пустой массив
    return;
   }

   const turnIds = this.currentUser.turns; // Получаем массив ID команд пользователя

   this.turnService.getTournaments().subscribe({
    next: (turns) => {
     this.userTurns = turns.filter((turn) => turnIds.includes(turn.id?.toString() || ''))
    },
    error: (err) => {
     this.error = 'Ошибка при загрузке турниров.';
     console.error('Error loading turns:', err);
    }
   });
  }

  exit(): void {
   this.authService.logout();
   this.router.navigate(['main-page']);
  }

  leave(teamId: number | undefined){
   if (!this.currentUser || !teamId) {
    console.error('Неверные данные для выхода из команды.');
    return;
   }
```

```
    // Найти команду по ID
    this.teamService.getTeams().subscribe({
      next: (teams) => {
        const team = teams.find((t) => t.id === teamId);
        if (!team) {
          console.error('Команда не найдена.');
          return;
        }

        if (team.creatorId === this.currentUser?.id) {
          // Если текущий пользователь является создателем команды, удаляем
команду
          this.teamService.deleteTeam(teamId).subscribe({
            next: () => {
              // Удаляем ID команды из свойства `teams` текущего пользователя
              this.updateUserTeams(teamId, true);
              alert('Вы успешно удалили команду.');
              this.loadUserTeams();
            },
            error: (err) => {
              console.error('Ошибка при удалении команды:', err);
              alert('Ошибка при удалении команды.');
            },
          });
        } else {
          // Если пользователь не является создателем команды
          // Удаляем пользователя из `members` команды
```

```
        const updatedMembers = team.members.filter((memberId) => memberId !==
this.currentUser?.id?.toString());
        this.teamService.updateTeam(teamId, { members: updatedMembers
}).subscribe({
          next: () => {
            // Удаляем ID команды из свойства `teams` текущего пользователя
            this.updateUserTeams(teamId, false);
            alert('Вы успешно покинули команду.');
            this.loadUserTeams();
          },
          error: (err) => {
            console.error('Ошибка при обновлении команды:', err);
            alert('Ошибка при покидании команды.');
          },
        });
      }
    },
    error: (err) => {
      console.error('Ошибка при загрузке команды:', err);
      alert('Ошибка при обработке команды.');
    },
  });
 }

 private updateUserTeams(teamId: number, isTeamDeleted: boolean): void {
  if (!this.currentUser) {
    console.error('Текущий пользователь не найден.');
    return;
  }
```

```
    const updatedTeams = this.currentUser.teams.filter((id) => id !== teamId.toString());

    // Если команда удалена, передаём обновлённые данные
    const userData = isTeamDeleted ? { ...this.currentUser, teams: updatedTeams } : { ...this.currentUser, teams: updatedTeams };

    this.userService.updateUser(userData).subscribe({
      next: (updatedUser) => {
        this.currentUser = updatedUser; // Обновляем текущего пользователя
      },
      error: (err) => {
        console.error('Ошибка при обновлении команд пользователя:', err);
        alert('Не удалось обновить список команд пользователя.');
      },
    });
  }
}
```

team.service.ts:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable, forkJoin } from 'rxjs';
import { Team } from '../team.model';
import { User } from '../../../auth/models/user.model';

@Injectable({
  providedIn: 'root',
})
```

```
export class TeamService {
  private apiUrl = 'http://localhost:3000/teams'; // URL для хранения данных команд

  constructor(private http: HttpClient) {}

  // Получение всех команд
  getTeams(): Observable<Team[]> {
    return this.http.get<Team[]>(this.apiUrl);
  }

  // Создание новой команды
  createTeam(team: Team): Observable<Team> {
    return this.http.post<Team>(this.apiUrl, team);
  }

  // Удаление команды
  deleteTeam(id: number): Observable<void> {
    return this.http.delete<void>(`${this.apiUrl}/${id}`);
  }


  getUsersByIds(ids: string[]): Observable<any[]> {
    const requests = ids.map((id) => this.http.get<any>(`http://localhost:3000/users/${id}`));
    return forkJoin(requests); // Выполняет запросы параллельно
  }

  // Обновление данных команды
```

```
  updateTeam(teamId: number, updatedData: Partial<Team>): Observable<Team>
{
    return this.http.patch<Team>(`${this.apiUrl}/${teamId}`, updatedData);
  }

  // Получить команду по ID
  getTeamById(teamId: string): Observable<Team> {
    return this.http.get<Team>(`${this.apiUrl}/${teamId}`);
  }

  // Получить несколько команд по их ID
  getTeamsByIds(ids: string[]): Observable<Team[]> {
    const query = ids.map((id) => `id=${id}`).join('&');
    return this.http.get<Team[]>(`${this.apiUrl}?${query}`);
  }
}
```

team.model.ts:

```
export interface Team {
    id?: number;           // Идентификатор команды
    name: string;          // Название команды
    gameType: string;      // Тип игры
    members: string[];     // Участники команды
    creatorId?: number;    // ID капитана команды
  }
```

teams.component.ts:

```
import { Component, OnInit } from '@angular/core';
import { FormGroup, FormControl, Validators, FormsModule,
ReactiveFormsModule } from '@angular/forms';
import { CommonModule } from '@angular/common';
```

```
import { TeamService } from './services/team.service';
import { UserService } from '../../auth/services/user.service';
import { Team } from './team.model';
import { Router } from '@angular/router';

@Component({
  selector: 'app-teams',
  standalone: true,
  imports: [CommonModule, FormsModule, ReactiveFormsModule],
  templateUrl: './teams.component.html',
  styleUrl: './teams.component.scss'
})
export class TeamsComponent implements OnInit{
  teamForm!: FormGroup;
  currentUserId!: number | undefined;

  constructor(private teamService: TeamService, private userService: UserService,
private router: Router) {}

  ngOnInit(): void {
    // Получение текущего пользователя
    this.userService.getCurrentUser().subscribe(user => {
      if (user !== null){
        this.currentUserId = user.id;
      }

    });

    // Инициализация формы
```

```
  this.teamForm = new FormGroup({
    name: new FormControl('', [Validators.required, Validators.maxLength(50)]),
    gameType: new FormControl('', [Validators.required]),
    members: new FormControl('', [Validators.required]),
  });
}

onSubmit(): void {
  if (this.teamForm.invalid || this.currentUserId === null) {
    alert('Форма некорректна или пользователь не авторизован');
    return;
  }

  const { name, gameType, members } = this.teamForm.value;
  // Разделяем строку с ID участников на массив, очищая от лишних пробелов
const userIds = members
.split(',')
.map((id: string) => id.trim()) // Удаляем лишние пробелы
.filter((id: string) => id.length > 0); // Убираем пустые строки

console.log('Проверка userIds:', userIds);

// Проверяем, существуют ли пользователи с указанными ID
this.teamService.getUsersByIds(userIds).subscribe({
  next: (users) => {
    console.log('Полученные пользователи:', users);
    if (users.length !== userIds.length) {
      alert('Некоторые пользователи не найдены!');
      return;
```

```
    }

    const newTeam: Team = {
      name,
      gameType,
      members: userIds, // Используем массив ID
      creatorId: this.currentUserId,
    };

    // Сохранение команды
    this.teamService.createTeam(newTeam).subscribe({
      next: (createdTeam) => {
        // Добавляем ID команды каждому участнику
        userIds.forEach((userId: string) => {
          this.userService.updateUserTeams(userId, createdTeam.id!).subscribe({
            next: () => {
              console.log(`Пользователь ${userId} обновлен`);
            },
            error: (error) => {
              console.error(`Ошибка при обновлении пользователя ${userId}:`,
error);
            },
          });
        });

        alert('Команда успешно создана!');
        this.router.navigate(['/prof']); // Перенаправление на список команд
      },
      error: (error) => {
```

```
          console.error('Ошибка при создании команды:', error);
          alert('Произошла ошибка при создании команды');
        },
      });
    },
    error: (error) => {
      console.error('Ошибка при проверке пользователей:', error);
      alert('Произошла ошибка при обработке списка участников');
    },
  });
}
}
```

tournament.servce.ts:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Tournament } from '../tournament.model';

@Injectable({
  providedIn: 'root',
})
export class TournamentService {
  private apiUrl = 'http://localhost:3000/tournaments'; // URL сервера

  constructor(private http: HttpClient) {}

  getTournaments(): Observable<Tournament[]> {
    return this.http.get<Tournament[]>(this.apiUrl);
  }
```

```
  createTournament(tournament: Tournament): Observable<Tournament> {
    return this.http.post<Tournament>(this.apiUrl, tournament);
  }

  updateTournament(tournament: Tournament): Observable<Tournament> {
    return this.http.put<Tournament>(`${this.apiUrl}/${tournament.id}`,
tournament);
  }
}
```

tournament.model.ts:

```
export interface Tournament {
    id?: number;         // Идентификатор турнира
    name: string;        // Название турнира
    date: string;        // Дата турнира в формате ISO
    time: string;        // Время турнира
    format: string;      // Формат турнира
    participants: string[]; // Список ID команд-участников
    max_num: number;
    creatorId?: number;   // ID пользователя, создавшего турнир
}
```

tournaments.component.ts:

```
import { Component, OnInit } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule, NgForm, NgModel } from '@angular/forms';
import { Tournament } from './tournament.model';
import { TournamentService } from './services/tournament.service';
import { TournComponent } from '../../common-ui/tourn/tourn.component';
import { UserService } from '../../auth/services/user.service';
```

```
@Component({
  selector: 'app-tournaments',
  imports: [CommonModule, FormsModule, TournComponent],
  templateUrl: './tournaments.component.html',
  styleUrl: './tournaments.component.scss',
})
export class TournamentsComponent implements OnInit {
  tournaments: Tournament[] = []; // Список всех турниров
  currentUserId!: number | undefined;
  today: string = ''; // Для минимальной даты
  currentTime: string = new Date().toISOString().split('T')[1]?.slice(0, 5) || '';
  constructor(private tournamentService: TournamentService, private userService: UserService) {}



  newTournament: Tournament = {
    name: '',
    date: '',
    time: '',
    max_num: 0,
    format: '',
    participants: [],
    creatorId: undefined, // ID текущего пользователя будет присвоен при создании
  };
```

```
ngOnInit(): void {
  this.loadTournaments();

  this.userService.getCurrentUser().subscribe(user => {
    if (user !== null){
      this.currentUserId = user.id;
    }

  });

  // Устанавливаем минимальную дату как сегодняшнюю
  const now = new Date();
  this.today = now.toISOString().split('T')[0];
}

loadTournaments(): void {
  this.tournamentService.getTournaments().subscribe({
    next: (tournaments: Tournament[]) => {
      this.tournaments = tournaments;
    },
    error: (err: any) => {
      console.error('Ошибка при загрузке турниров:', err);
    },
  });
}

createTournament(event: Event, form: any): void {
  event.preventDefault();
```

```
    if (form.invalid){
      return;
    }
    const creatorId = this.currentUserId; // Подставьте текущего пользователя
    const tournament = { ...this.newTournament, creatorId };
    console.log(tournament);
    this.tournamentService.createTournament(tournament).subscribe({
      next: (createdTournament) => {
        this.tournaments.push(createdTournament);
        this.newTournament = { name: '', date: '', time: '', max_num: 0, format: '',
participants: [], creatorId: undefined };
      },
      error: (err) => {
        console.error('Ошибка при создании турнира:', err);
      },
    });
  }

  // Проверка времени на валидность (не ранее чем через 2 часа)
  validateTime(model: NgModel): boolean {
    const inputDate = new
Date(`${this.newTournament.date}T${this.newTournament.time}`);
    const currentDate = new Date();
    const twoHoursLater = new Date(currentDate.getTime() + 2 * 60 * 60 * 1000);

    if (this.newTournament.date === this.today) {
      // Если дата сегодняшняя, проверяем, что время позже чем через 2 часа
      if (inputDate.getTime() < twoHoursLater.getTime()) {
        model.control.setErrors({ invalidTime: true });
```

```
        return false;
      }
    }

    // Если дата не сегодняшняя, считаем время корректным
    return true;
  }
}
```

app.component.ts:

```
import { Component } from '@angular/core';
import { RouterOutlet, Event, NavigationEnd, Router} from '@angular/router';
import { HeaderComponent } from './common-ui/header/header.component';
import { MainPageComponent } from './page/main-page/main-page.component';
import { FooterComponent } from './common-ui/footer/footer.component';
import { AuthComponent } from './auth/auth.component';
/* import {} */
@Component({
  selector: 'app-root',
  standalone: true,
  imports: [RouterOutlet, HeaderComponent, FooterComponent],
  templateUrl: './app.component.html',
  styleUrl: './app.component.scss'
})
export class AppComponent {
  title = 'Turn';
  constructor(private router: Router) {
    this.router.events.subscribe((event: Event) => {
      if (event instanceof NavigationEnd) {
        window.scrollTo(0, 0); // Прокрутка страницы в начало
```

```
      }
    });
  }
  ngOnInit(): void {
    this.router.navigate(['/main-page']);
  }
}
```

app.config.ts:

```
import { ApplicationConfig, provideZoneChangeDetection } from '@angular/core';
import { provideRouter } from '@angular/router';

import { routes } from './app.routes';
import { provideHttpClient } from '@angular/common/http';
import { AuthService } from './auth/services/auth.service';

export const appConfig: ApplicationConfig = {
  providers: [provideZoneChangeDetection({ eventCoalescing: true }), provideRouter(routes), provideHttpClient(),AuthService]
};
```

app.routes.ts:

```
import { RouterModule, Routes } from '@angular/router';
import { MainPageComponent } from './page/main-page/main-page.component';
import { AuthComponent } from './auth/auth.component';
import { AutorizationComponent } from './auth/autorization/autorization.component';
import { NgModule } from '@angular/core';
import { AuthService } from './auth/services/auth.service';
import { UserService } from './auth/services/user.service';
import { ProfileComponent } from './page/profile/profile.component';
```

```
import { TournamentsComponent } from './page/tournaments/tournaments.component';
import { TeamsComponent } from './page/teams/teams.component';
import { TournComponent } from './common-ui/tourn/tourn.component'
import { RegistrationComponent } from './auth/registration/registration.component';
export const routes: Routes = [

    {path:'', component: MainPageComponent},
    {path:'main-page',component:MainPageComponent},
    {path:'auth', component:AuthComponent},
    {path:'autoriz', component:AutorizationComponent},
    {path:'registr', component:RegistrationComponent},
    {path:'prof', component: ProfileComponent},
    {path:'tour', component: TournamentsComponent},
    {path:'teams', component: TeamsComponent},
    {path:'tourn', component: TournComponent}

];

@NgModule({
    imports: [RouterModule.forRoot(routes)],
    exports: [RouterModule],
    providers: [AuthService, UserService]
  })

  export class SystemRoutingModule { }
```

main.ts:

```
import { bootstrapApplication } from '@angular/platform-browser';
import { appConfig } from './app/app.config';
```

```
import { AppComponent } from './app/app.component';

bootstrapApplication(AppComponent, appConfig)
  .catch((err) => console.error(err));
```