МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное
учреждение высшего образования
**"Южно-Уральский государственный университет
(национальный исследовательский университет)"**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

РАБОТА ПРОВЕРЕНА
Рецензент
Руководитель группы сопровождения
ООО ”ИТ Менеджмент”
_____А.А. Нусратуллин
“___”_____ 2017 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н., профессор
_____ Л.Б. Соколинский
“___”_____ 2017 г.

РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ РЕГИСТРАЦИИ И УЧАСТИЯ В ЛЮБИТЕЛЬСКИХ ЗАЕЗДАХ НА ГОНОЧНЫХ ТРАССАХ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.04.02.2017.115-137.ВКР

Научный руководитель,
кандидат физ.-мат. наук
_____ С.А. Иванов

Автор работы,
студент группы КЭ-217
_____ И.С. Демидович

Ученый секретарь
(нормоконтролер)

_____ О.Н. Иванова
“___”_____ 2017 г.

Челябинск-2017

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность работы

Еще 5 – 7 лет назад в России автомобильный спорт был далеко не популярным. Но со временем ситуация начинает меняться, в данный момент в стране есть 7 – 8 профессиональных спортивных треков. На таких треках проводятся различные чемпионаты, события, а также трек-дни на которых любой желающий может попробовать себя в роли автогонщика, соответственно с определенными требованиями.

Для развития трека нужно создавать специальные условия, проводить рекламные компании, в современности обязательно нужно иметь веб сайт для удобства клиентов. В наше время, когда люди в основном пользуются мобильными устройствами, фирмам требуется не только собственные сайты, но и мобильное приложение. Проведя мониторинг 6-7 треков было выявлено, что почти у всех есть свой сайт, но ни у одного трека нет мобильного приложения. Пользоваться сайтами на мобильных устройствах не всегда удобно, особенно если разметка не оптимизирована под мобильные устройства. Учитывая популярность и массовость мобильных устройств, это является проблемой для пользователей смартфонов.

Создание приложения для мобильных устройств, содержащего основные функции для предоставления услуг гоночного трека, является актуальной задачей.

Цель и задачи

Основной целью данной работы является разработка клиент-серверного мобильного приложения для регистрации в гоночных заездах, которые проводятся в отведенные трек дни. Должна быть реализована авторизация пользователей и дополнительный функционал, такой как новостная лента и календарь событий, проводимых на данном треке. Приложение должно разрабатываться для устройств использующих операционную систему Android. Для того чтобы клиент взаимодействовал с сервером требуется разработать API. Сервер должен обрабатывать данные и записывать их в базу данных.

Для достижения поставленных ранее целей, нужно решить следующие задачи:

1) провести обзор идентичных существующих решений;

2) спроектировать архитектуру клиент – сервер, выявить основные требования к построению данной архитектуры, использовать в разработке стандарт UML;

3) разработать сервер приложения, предоставляющий набор REST API, для взаимодействия с клиентом;

4) разработать Android приложение TrackApp, соответствующее поставленным целям;

5) провести тестирование разработанного приложения;

Структура и объем работы

Данная работа состоит из введения, пяти глав, заключения и списка библиографии. Объем работы составляет 43 страницы, объем библиографии – 22 источника.

В первой главе проведен краткий обзор аналогичных решений, предназначенных для регистрации и участия в любительских заездах.

Вторая глава содержит информацию о разрабатываемом приложении, функциональные и нефункциональные требования, предъявляемые к приложению.

Третья глава содержит архитектуру приложения. Производится обзор статического и динамического поведения архитектуры разрабатываемого приложения.

В четвертой главе представлена реализация приложения, которая состоит из реализации REST API и реализации клиента приложения.

В пятой главе представлено модульное и функцтональное тестирование приложения.

В заключении сделаны выводы о проделанной работе.

1. ОБЗОР АНАЛОГОВ

Проведя мониторинг магазина мобильных приложений Google PlayMarket, схожих по области применения приложений не было найдено. Решения связанные с регистрацией на участие в трек-днях в основном реализованы в виде веб-сайтов.

По моему мнению, лучшим веб-сайтом для бронирования билетов является MoscowRaceway [18], который изображен на рисунке 1. Данный сайт разработан для гоночного трека. Из всех рассмотренных сайтов он предоставляет лучший функционал. Например, просмотр новостей в мире автоспорта, новостей на самом треке, расписания событий, карт треков, возможность бронирования билетов. На сайте правильно составлены формы регистрации, в хорошем формате представлены новости и события. На сайте имеется галерея фотоотчетов, с проводимых событий. Данный сайт выполнен очень качественно и красиво. Развитие ИТ инфраструктуры в данном направлении с каждым днем набирает свою популярность, а ИТ технологии в свою очередь предоставляют все больше различных технологий для реализации подобных проектов. Рассмотренный трек не предоставляет пользователям мобильное приложение, хотя в данный момент это является минусом. Создание приложения в данном направлении имеет смысл, потому что сам автоспорт и в частности треки начали развиваться совсем недавно и пока нет достойных решений, предназначенных для гоночных треков.

Сайт Smolenskring [20] изображен на рисунке 2, он намного проще предыдущего и не имеет большой функциональности. Сайт имеет более новостной характер и не подходит как пример разработки мобильного приложения, для регистрации и участия в любительских заездах на гоночных трассах, так как пользователь никак не может взаимодействовать с ним. Единственная интересная возможность этого сайта в том, что можно смотреть онлайн трансляцию в данный момент времени, но она присутствует и в других похожих веб сайтах, как функциональность в нашем приложении она не будет использоваться.

Сайт Nring [19] изображенный на рисунке 3, как и первый имеет, хороший функционал и дизайн. Главным отличием этого сайта является наличие авторизации, что непременно потребуется в разрабатываемом мобильном приложении.

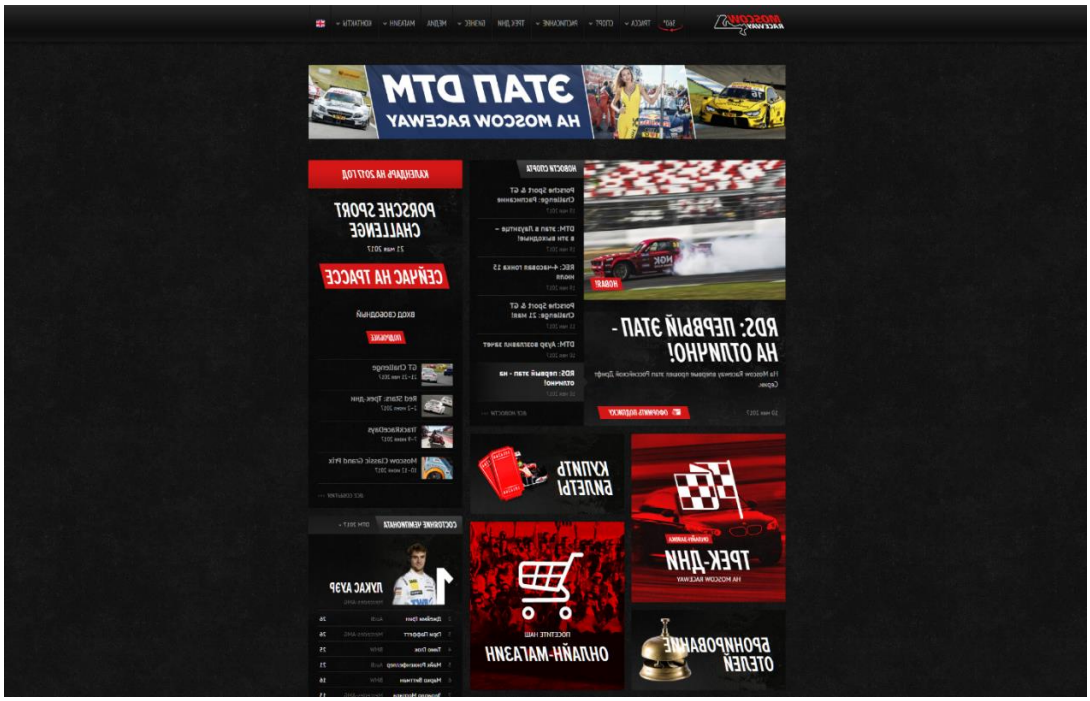

Рис. 1. Веб-сайт MoscowRaceway

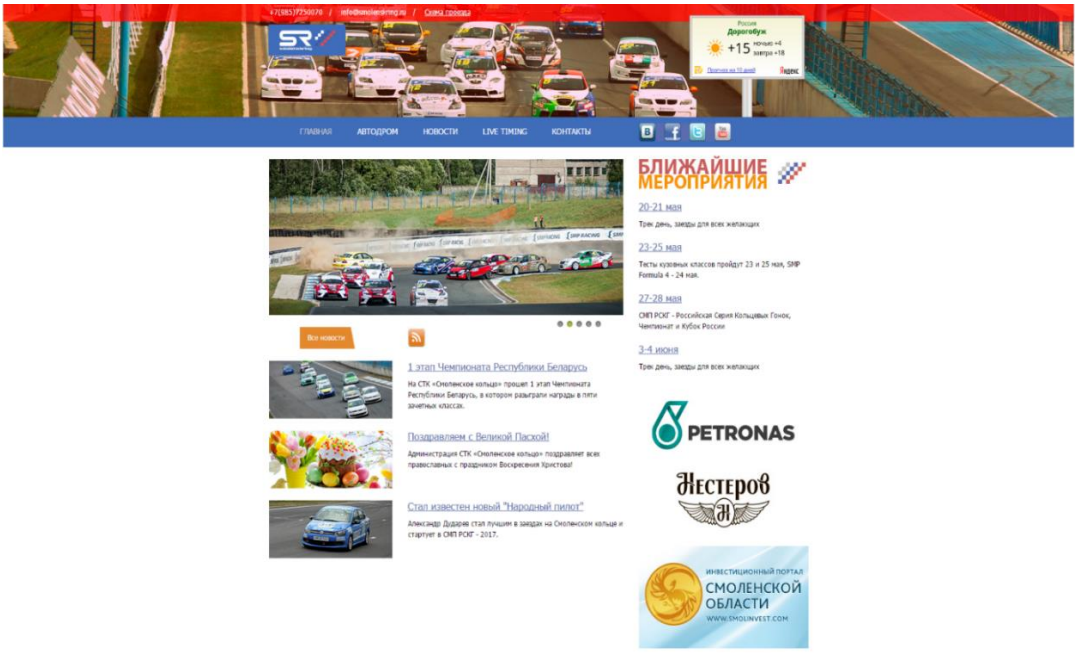

Рис. 2. Веб-сайт Smolenskring

Рис. 3. Веб-сайт Nring

Вывод

На основании рассмотренных аналогичных решений можно выделить преимущества и недостатки. Нужно рассмотреть все необходимые и полезные функции и в будущем реализовать их на платформе Android.

2. ТРЕБОВАНИЯ К ПРИЛОЖЕНИЮ

2.1. Общие сведения о приложении

Приложение TrackApp предназначено для регистрации пользователей в любительских заездах на гоночных трассах. Приложение предоставляет пользователям удобный интерфейс для регистрации в любительских заездах, просмотра новостей, событий, проводимых в ближайшее время на гоночном треке.

Приложение должно предоставлять сервис для регистрации, авторизации пользователей и содержать стандартную форму авторизации.

Пользователям будет предоставлен новостной сервис, с помощью которого пользователь сможет просматривать новости, связанные с данным треком и в общем с автоспортом, также он будет содержать информацию по различными турнирам, проведенным на данном треке, выглядеть он будет как мини карточки с новостями.

Календарь событий будет содержать информацию о планируемых событиях на треке, а также стоимость билетов для посещения этого события. В последнее время проводится очень много турниров в различных дисциплинах автоспорта, это могут быть классические заезды, drift турниры набирающие большую популярность в последнее время, гонки на спортивных мотоциклах.

Список всех трасс с подробными их характеристиками позволит лучше познакомиться с трассами чтобы примерно знать маршрут.

Специальная форма позволит забронировать билет для участия в трек днях, форма содержит день проведения события и подробное расписание заездов, пользователь может легко выбрать устраивающий день и время проведения и после выбора подтвердить бронирование билета для участия в трек днях.

Основными преимуществами данного приложения, будет удобный интерфейс, быстродействие, простота в обслуживании, дешевизна в эксплуатации. Приложение в будущем может свободно наращивать функционал.

2.2. Функциональные требования

Функциональные требования предоставляют информацию, о том, как должна быть реализована система, описывают что нужно реализовать в системе

и какие возможности она реализует. Функциональные требования содержат бизнес требования, которые определяют назначение ПО и пользовательские, определяющие набор пользовательских задач, которые должна решать программа. Для приложения TrackApp были выявлены следующие функциональные требования:

1) приложение должно предоставлять интерфейс для авторизации пользователя;

2) приложение должно предоставлять интерфейс для регистрации пользователя;

3) приложение должно предоставлять интерфейс для просмотра новостной ленты;

4) приложение должно предоставлять интерфейс для просмотра календаря событий;

5) приложение должно предоставлять интерфейс для просмотра треков и их характеристик;

6) приложение должно предоставлять интерфейс бронирования билета для участия в трек днях и подтверждения бронирования;

7) приложение должно предоставлять интерфейс для просмотра информации о пользователе.

2.3. Нефункциональные требования

Нефункциональные требования определяют критерии работы системы в целом, а не отдельные сценарии поведения. Нефункциональные требования определяют системные свойства такие как производительность, удобство сопровождения, расширяемость, надежность, средовые факторы эксплуатации. В процессе проектирования мобильного приложения для проведения и регистрации в любительских заездах TrackApp были выявлены следующие функциональные требования:

1) система должна быть реализована на основе архитектуры: клиент – сервер – сервер базы данных;

2) сервер должен разрабатываться с помощью языка программирования Java и развертываться на платформе Tomcat Apache [15];

3) сервер должен предоставлять REST API [8] для манипуляций с данными;

4) клиент мобильного приложения должен быть разработан на языке программирования Java с помощью платформы Android Studio.

2.4. Описание прецедентов

При проектировании приложения было выделено 2 актера, взаимодействующих с системой: пользователь, администратор. Это наглядно показано на диаграмме использования (рис. 4).

Пользователь может создать учетную запись, авторизоваться в приложении с помощью созданной записи и пароля, указанного при регистрации. Пользователь может просматривать новостную ленту, предоставляющую новости автоспорта, календарь событий на котором будут отображаться события, намеченные на ближайшее время. Для пользователя предусмотрена регистрация на участие в трек днях, с выбором даты и времени проведения заезда, который будет проводиться в установленное администрацией время.

Администратор работает с базой данных, в которой хранятся все данные приложения и имеет полные права на управление всеми данными.

Создать учетную запись может любой незарегистрированный в приложении пользователь.

Авторизоваться в приложении может любой зарегистрированный пользователь, пройдя процедуру авторизации приложение.

Любой авторизированный пользователь может просматривать новостную ленту, календарь событий и информацию о треках.

Подать заявку на участие в трек-днях может любой зарегистрированный пользователь.

Администратор может удалять учетные записи пользователей.

Администратор может создавать, редактировать, удалять новости из новостной ленты.

Администратор может создавать, редактировать, удалять календарные события.

Администратор может добавлять, удалять даты трек дней.

Администратор может создавать, редактировать, удалять заявки на участие в трек-днях.

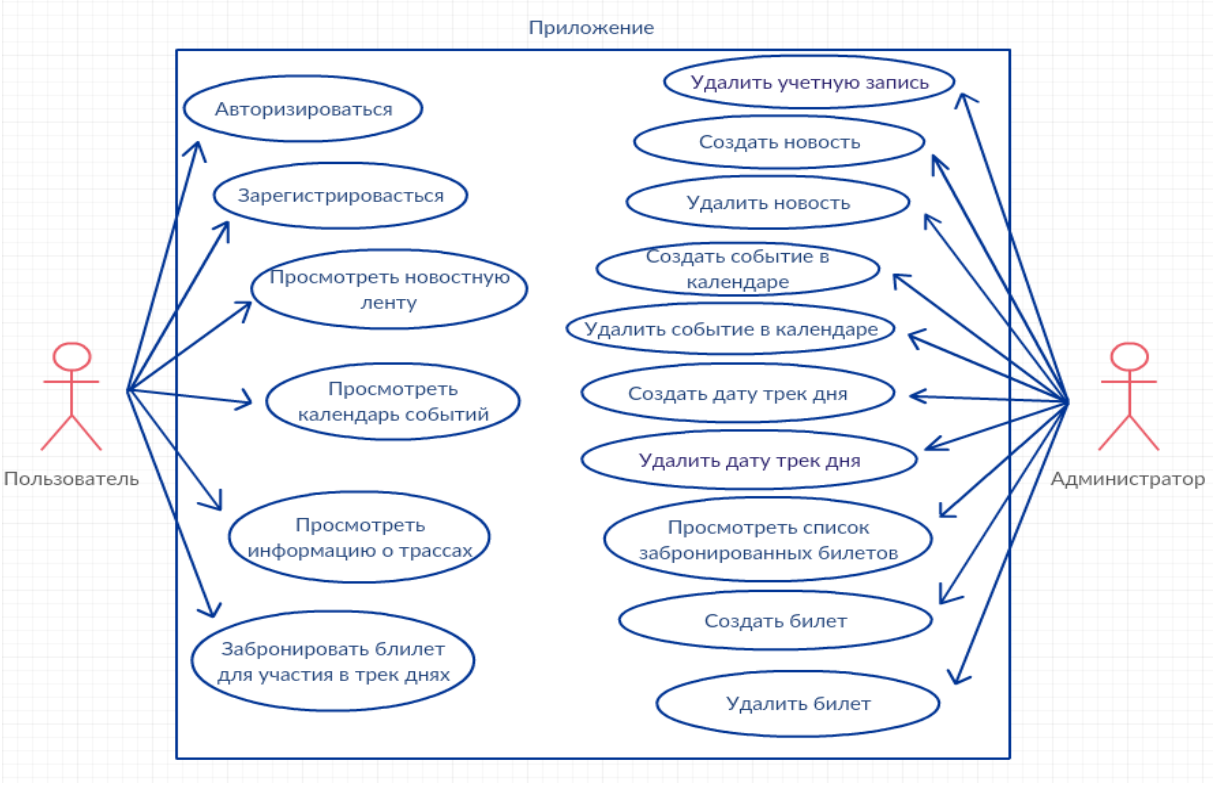

Рис. 4. Диаграмма использования приложения TrackApp

Вывод

С помощью проведения анализа требований было составлено техническое задание, которое содержит функциональные и нефункциональные требования для разрабатываемого приложения, выделены основные пользователи и составлена диаграмма использования приложения.

3. АРХИТЕКТУРА ПРИЛОЖЕНИЯ

Система спроектирована на основе трехзвенной клиент-серверной архитектуры: клиент – сервер – сервер БД. Клиентом является android-приложение. Для схематичного изображения архитектуры была выбрана диаграмма компонентов. Диаграмма компонентов статическая структурная диаграмма, показывает разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами.

Диаграмма компонентов разрабатываемого приложения изображена на рисунке 5.

Компоненты приложения:

1) клиенты;
2) REST-сервис;
3) сервер БД.

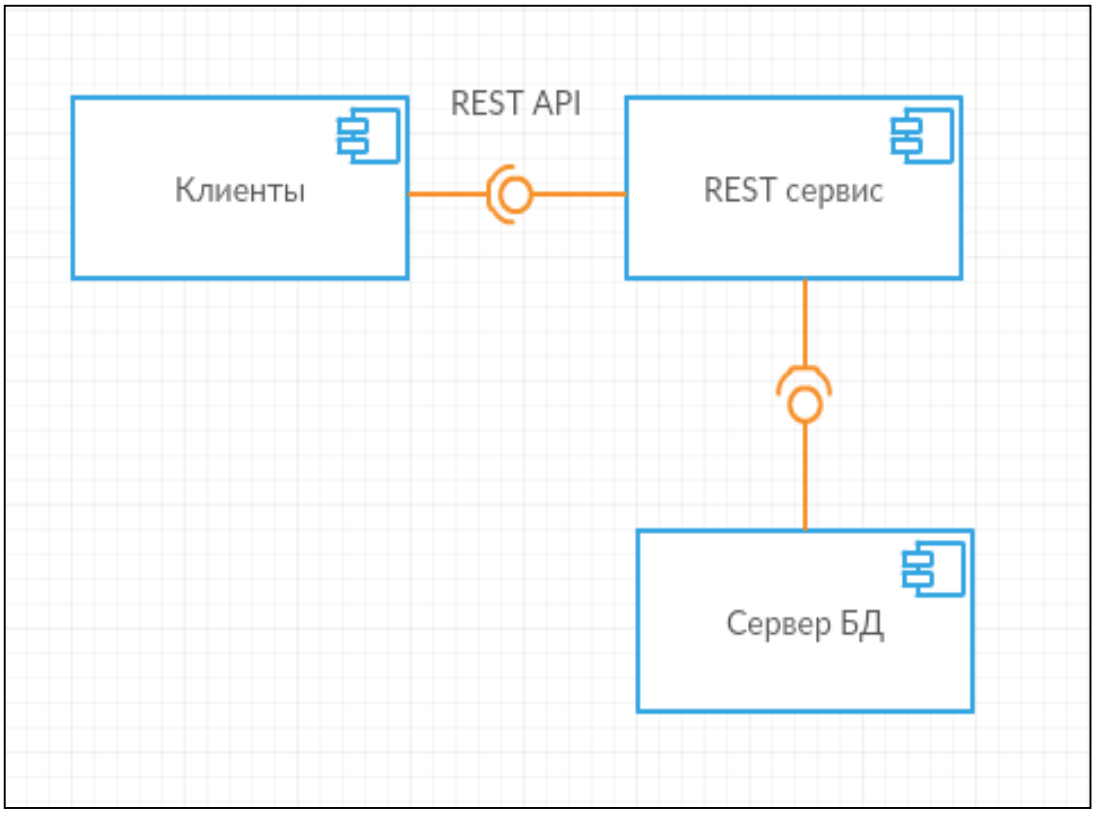

Рис. 5. Диаграмма компонентов приложения TrackApp

Клиенты общаются с сервисом с помощью REST API. Данные передаются посредством протокола HTTP. Основная логика приложения реализована на стороне сервера, операции по управлению учетными записями, новостями, заявками, выполняются с помощью методов, реализованных в REST-сервисах.

Клиенты постоянно взаимодействуют с REST-сервисом для обновления данных, которые содержаться в базе данных приложения (новости, календарные события). Также клиенты постоянно делают запрос серверу на соответствие данных, для авторизации пользователей, регистрации новых пользователей, для создания новых заявок на участие в трек днях.

Для лучшего понимания динамической работы архитектуры, была использована диаграмма последовательности. Диаграмма последовательности описывает взаимодействие групп объектов в различных условиях и их поведение. Проще говоря, диаграмма последовательности моделирует взаимодействие объектов во времени.

Для начала работы с приложением необходимо зарегистрироваться с помощью формы регистрации нового пользователя.

Процесс добавления нового пользователя в систему изображён на рисунке 6 и производится следующим образом:

1) незарегистрированный пользователь в меню регистрации приложения заполняет все необходимые для регистрации поля, нажимает кнопку зарегистрироваться;

2) android приложение отправляет запрос серверу, на добавление нового пользователя с помощью REST API;

3) сервер получает запрос на добавление нового пользователя через REST сервис. С помощью методов, определенных в REST-сервисе, сервер обращается к базе данных для создания нового пользователя в мобильном приложении;

4) сервер БД возвращает ответ о создании записи;

5) сервер приложения возвращает сообщение об успешном создании пользователя в системе;

6) приложение отображает пользователю возвращенное сообщение.

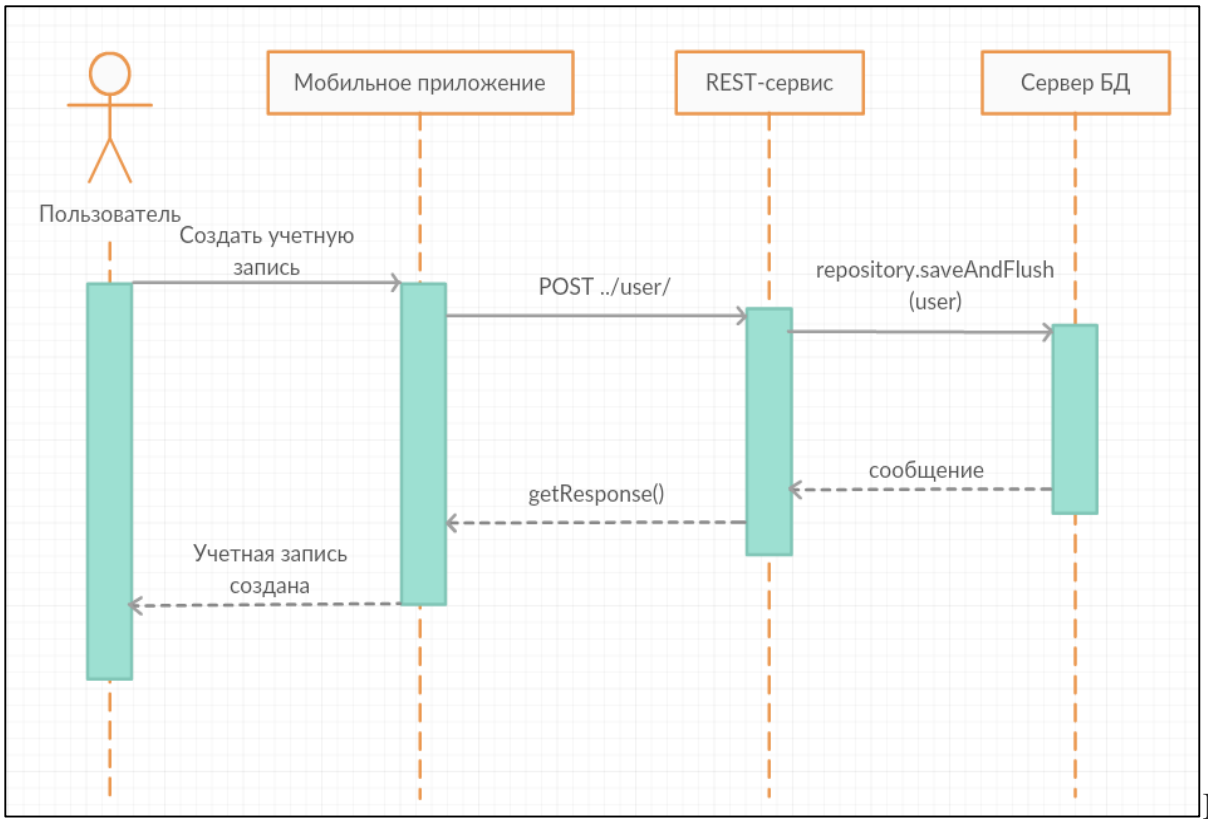

Рис. 6. Процесс добавления нового пользователя

После того как пользователь зарегистрировался, он может войти в приложение. После авторизации, пользователя направляет на главную страницу приложения, на которой отображается новостная лента. Процесс просмотра новостной ленты (рис. 7):

1) приложение отправляет запрос серверу на получение новостей в новостную ленту;

2) сервер получает и обрабатывает запрос на получение новостей через REST сервис;

3) с помощью методов, определенных в REST сервисе, сервер забирает данные о новостях из базы данных;

4) сервер компонует все полученные данные в объект языка java, для последующей отправки этого объекта в android-приложение;

5) android-приложение принимает от сервера объект с данными о новостях, обрабатывает его и представляет графическое представление пользователю.

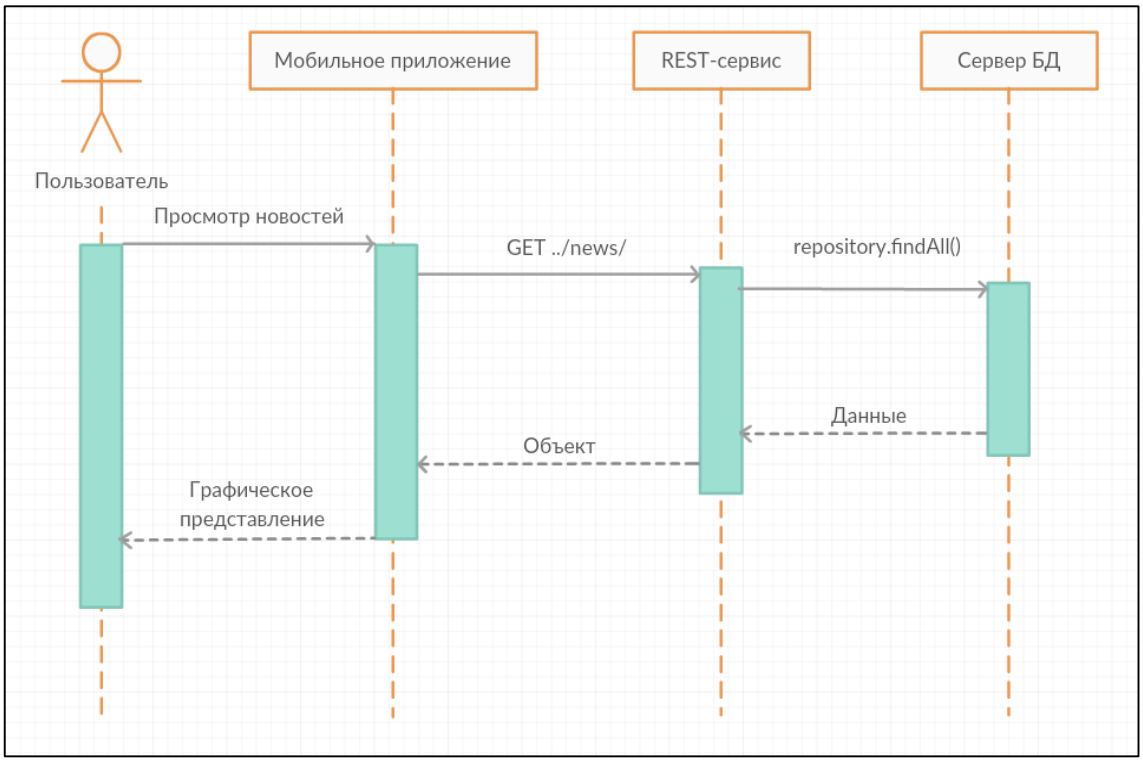

Рис. 7. Процесс работы с приложением, просмотр новостной ленты

В меню приложения пользователь может забронировать билет для участия в трек днях. Процесс бронирования билета выглядит следующим образом (рис. 8):

1) при переходе в меню регистрации билета на участие в трек днях, приложение отправляет запрос серверу, на получение возможных дат проведения трек-дней;

2) сервер получает и обрабатывает запрос на получение списка возможных дат трек-дней, через REST-сервис;

3) с помощью методов, определенных в REST-сервисе, сервер забирает данные о датах трек-дней из базы данных;

4) сервер компонует все полученные данные в объект языка java, для последующей отправки этого объекта в android-приложение;

5) android-приложение принимает от сервера объект с данными трек-дней, обрабатывает его и представляет графическое представление пользователю;

6) приложение собирает данные об авторизованном пользователе;

7) открывается графическое представление, содержащее всю необходимую для регистрации информации;

8) пользователь нажимает кнопку забронировать;

9) приложение отправляет запрос на запись билета для регистрации в трек-дне;

10) сервер возвращает сообщение приложению об успешном бронирование билета в системе;

11) приложение отображает пользователю возвращенное сообщение.

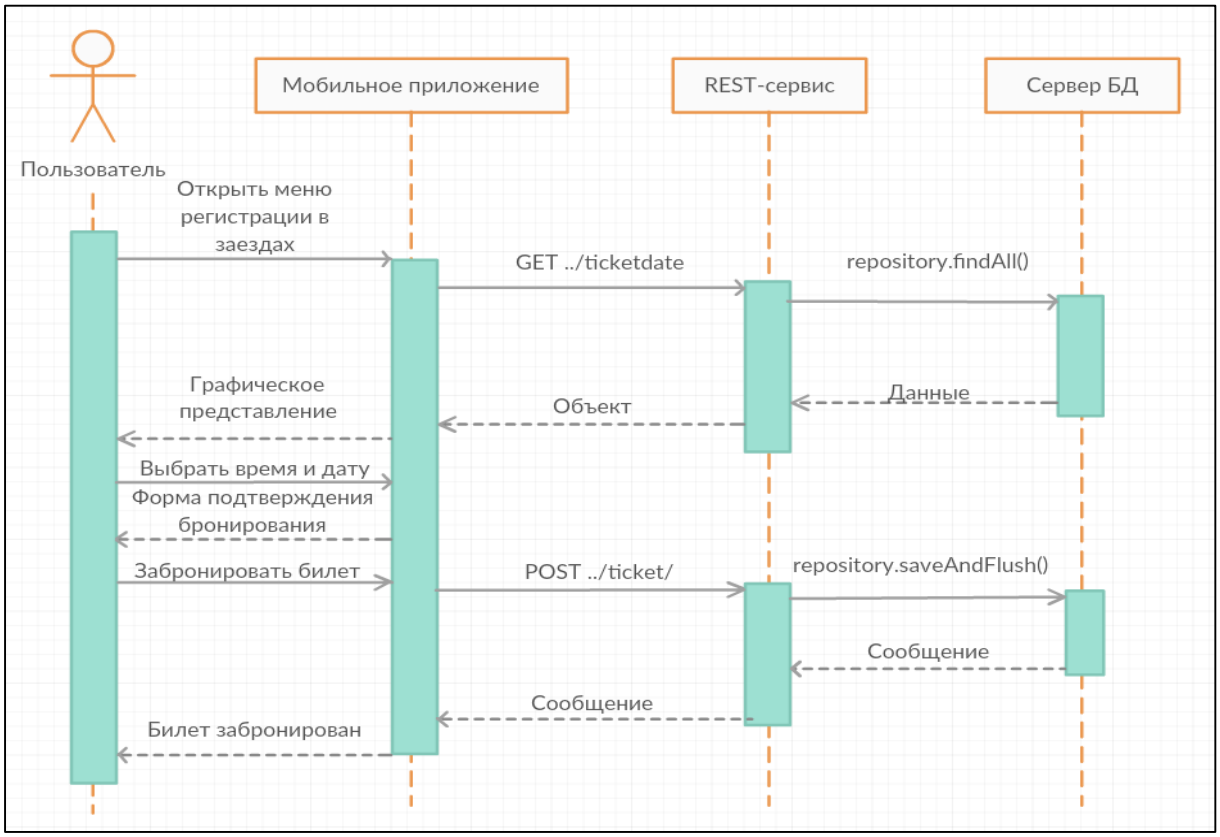

Рис. 8. Процесс работы с приложением, бронирование билета на участие в трек-днях

Вывод

В итоге для приложения была спроектирована архитектура, которая состоит из трех частей – это клиент, сервер на основе REST сервисов, сервер базы данных. Преимущество данной архитектуры в слабосвязанных компонентах, это позволит в будущем с легкостью расширять функционал.

4. РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ.

4.1. База данных

Все данные мобильного приложения будут храниться на сервере базы данных. База данных приложения базируется на PostgreSQL.

PostgreSQL [6] является не просто реляционной, а объектно-реляционной СУБД. Это дает некоторые преимущества, такие как поддержка пользовательских объектов, управление их поведением, типами данных, функциями, операциями, доменами и индексами. Это делает работу с базой данных более гибкой и надежной, а также позволяет работать с сложными структурами данных.

REST-сервис работает с базой данных, с помощью библиотеки Spring data и провайдера hibernate [5]. Главная задача провайдера hibernate представление данных базы данных в виде объектов. Все это происходит не просто на уровне таблиц, а на уровне отношений между таблицами. Данный провайдер также предназначен для управления всеми операциями по работе с данными, такими как запись, удаление, обновление. Hibernate позволяет просто и удобно общаться с базой данных.

На сервере с помощью так называемых сущностей entity описывается представление таблицы в виде объекта языка java, создаются таблицы при запуске сервера, можно создавать всю базу данных заново, также можно работать уже с готовой базой правильно описав необходимые таблицы их свойства и отношения. На клиенте приложения сущности будут иметь такое-же представление, как и на сервере.

Все это позволит управлять данными с помощью объектов, это очень удобно и очень сильно уменьшает количество кода приложения и сервера, также hibernate позволяет использовать запросы SQL, то есть если не получается выполнить какой-либо запрос с помощью методов предоставляемых hibernate, есть возможность создать SQL запрос.

Сущности будут выглядеть следующим образом.

Сущность News изображена на рисунке 9.

```java
@Entity
@Table(name = "firstData")
public class News {

    @Id
    @GeneratedValue(generator = "increment")
    @GenericGenerator(name = "increment", strategy = "increment")
    private long id;

    @Column (name = "title", nullable = false, length = 50)
    private String title;

    @Column (name = "remindDate", nullable = false)
    @Temporal(TemporalType.TIMESTAMP)
    private Date remindDate;

    @Column (name = "img", nullable = false)
    private String img;

    @Column(name = "info", nullable = false)
    private String info;

    public News(){
    }
```

Рис.9. Сущность News

Представление сущности news в PostgreSQL:

```
news
ID BIGINT PK
title character varying (50)
date timestamp without time zone
img character varying (255)
info character varying (255)
```

Аннотация @Entity говорит о том, что данный класс является сущностью. Аннотация @Table задает имя таблицы, @GeneratedValue @GenericGenerator создают поле, которое генерируется автоматически и автоматически инкрементируется. @Column создает колонку с указаными в свойствах параметрами такими как тип данных, заполнение, допустимое количество символов. @Temporal специально предназначена для хранения данных типа дата.

Сущность Users изображена на рисунке 10.

```java
@Entity
@Table(name = "Users")
public class User {

    @Id
    @GeneratedValue(generator = "increment")
    @GenericGenerator(name = "increment", strategy = "increment")
    private long id;

    @Column(name = "name", nullable = false)
    private String name;

    @Column(name = "password", nullable = false)
    private String password;

    public User(String name, String password) {

        this.name = name;
        this.setPassword(password);
    }
```

Рис.10 Сущность “Users”

Представление сущности users в PostgreSQL:

```
users
ID BIGINT PK
name character varying (255)
password character varying (255)
email character varying (255)
phone character varying (255)
```

По такому же сценарию создаются и другие сущности. Каждой сущности соответствует свой контроллер, репозиторий и сервис. Сущностей может создаваться неограниченное количество.

База данных настраивается с помощью специального класса, который конфигурирует ее на сервере. Аннотация @Configuration помечает класс как конфигурацию. @EnableJpaRepositories эта аннотация говорит spring контексту о том, что необходимо использовать JPA [13], а именно spring data который основан на паттерне repository и выбирается пакет, в котором будут находиться все репозитории. Аннотация @EnableTransactionManagement включает поддержку транзакций. Транзакционность предоставляет возможность отката определенных транзакций, то есть если происходят какие-либо действия с базой данных, например,

обновление каких-либо данных и в результате добавления этой записи при изменении данных происходит ошибка или сбой в базе данных, в этом случае произойдет откат данных на стороне базы данных и данные не пострадают. @ComponentScan указывает путь, по которому нужно искать все репозитории и конфигурации.

С помощью аннотации @Resource вставляется объект Envirement с пакета org.springframework.core.env с его помощью предоставляется доступ к property файлам и получаются некоторые значения помещенные в них. @PropertySource указывает конкретный путь к файлу настроек базы данных.

DataSource является связующим звеном между базой данных и библиотекой Spring data [11], а точнее с JDBC драйвером. В DataSource для инициализации EntityManager с помощью которого происходит работа с базой данных, требуется конфигурация бинов. Для конфигурации бина указывается метод который возвращает данный объект, метод аннотируется - @Bean чтобы spring понял, что данный метод бин, после этого описывается развертывание бина. В качестве DataSource используется библиотека от apache dbcp2 [4]. Для конфигурации устанавливаются определенные параметры. Инициализация DataSource предназначена для подготовки инициализации основного бина, который предназначен непосредственно для работы с базой данных. Название этого бина LocalContainerEntityManagerFactoryBean. Данный бин предоставляет возможность автоматического создания бинов и создания таблиц на основе бинов Entity. Для поддержки этой возможности инициализируется объект EntityManager.

4.2. REST API.

Взаимодействие клиента и сервера происходит с помощью REST API. REST API базируется на протоколе HTTP. Все клиенты подключаются к одному API.

REST API предоставляет набор методов для работы приложения. В приложении используются методы:

1) GET позволяет получить данные;
2) POST позволяет записать данные;

3) DELETE позволяет удалить данные;

4) PUT позволяет обновить данные.

На сервере созданы REST-сервисы которые работают с данными приложения и базы данных. Клиенты и сервер передают между собой объекты языка java [22] сериализуя (переводя данные в последовательность битов) их в формат JSON, а при получении десериализуюя (восстанавливая данные до начального состояния из битовой последовательности) их обратно в объекты языка java. На сервере каждой сущности Entity соответствует свой контроллер, репозиторий и сервис.

4.2.1. Контроллер

Контроллер по факту является сервлетом, то есть классом, который общается с приложением через HTTP протокол посредством HTTP методов, базируясь на запросе (Request) и ответе (Response). При создании класс аннотируется @RestController. Аннотация @RequestMapping говорит серверу как можно попасть на данный контроллер, эту аннотацию не обязательно указывать перед классом, ее можно использовать перед любым методом находящемся в контроллере, чтобы обратиться именно к нему. В параметре аннотации value указывается путь обращения к контроллеру, в параметре method тип метода обращения (GET, POST, DELETE, PUT). Чтобы приложение было гибким, в контроллере не реализуется бизнес логика, она выносится в сервисы. Аннотация @Autowired используется для инициализации объекта в контроллере, а именно объекта сервиса, который манипулирует данными и выполняет конкретную часть бизнес логики мобильного приложения, реализованную именно для данного сервиса. Контроллер соответствует хранилищу (репозиторию), сервису и сущности. По сути контроллер является входной и выходной точкой нашего сервера и конкретно определенного сервиса, так как каждый сервис предоставляет свою функциональность.

4.2.2. JSON конвертер

По стандарту spring не знает, как передать в клиент объект типа JSON, а в данной реализации объекты имеют представление java объектов. Для того чтобы

конвертировать java объект можно использовать готовые библиотеки, предназначенные для конвертации данных, в нашем случае используется Jackson.

Чтобы реализовать конвертер переопределяется метод configureMessageConverters в веб-конфигурации. После этого создается и инициализируется экземпляр класса MappingJackson2HttpMessageConverter который хранит в себе классы разных конвертеров. С помощью maven добавляется зависимость для динамического подключения библиотеки Jackson. После этого для конвертера устанавливается ObjectMapper, из пакета добавленной библиотеки Jackson и также устанавливается SupportedMediaTypes в котором мы обращаемся к методу Collections.singletonList в данном методе устанавливается тип конвертируемых данных MediaType.APPLICATION_JSON так как необходимо конвертировать данные в JSON объект.

Пример JSON конвертера [10] изображен на рисунке 11.

```
public class WebConfig extends WebMvcConfigurerAdapter {
    @Override
    public void configureMessageConverters(List<HttpMessageConverter<?>> converters) {
        MappingJackson2HttpMessageConverter converter = new MappingJackson2HttpMessageConverter();
        converter.setObjectMapper(new ObjectMapper());
        converter.setSupportedMediaTypes(Collections.singletonList(MediaType.APPLICATION_JSON));

        converters.add(converter);
    }
```

Рис. 11. JSON конвертер

В клиенте нужно использовать идентичную библиотеку чтобы принять данные. Сериализация и десериализация данных производится автоматически с помощью возможностей библиотеки Jackson.

На рисунке 12 изображен пример передаваемых данных в формате JSON.

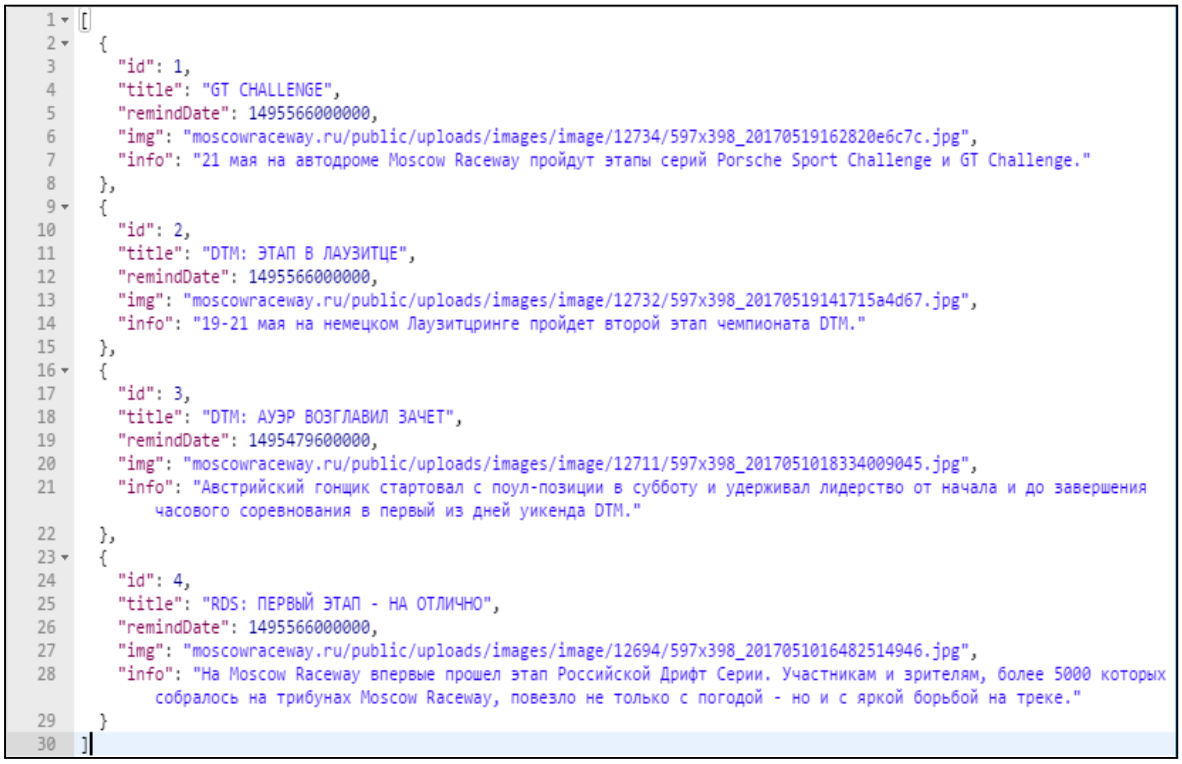

Рис. 12. Пересылаемые данные в формате JSON

Таким образом выглядят объекты языка java сконвертированные в JSON. На клиенте приложения они принимаются и с помощью библиотеки Jackson конвертируются обратно в java-объекты.

4.2.3. Хранилище

Reposytory [5] либо DAO являются слоем объектов которые предоставляют доступ к данным. Для реализации DAO используется EntitManager который находится в конфигурации базы данных, и с помощью него производится работа с базой данных, но так как используется spring framework подход немного отличается от стандартного, в котором используется DAO. Spring Data предоставляет набор готовых реализаций для создания DAO, но в spring данный слой называется не DAO а Repository.

Для каждого Entity которое создано на сервере нужно создать Repository который позволит оперировать объектами в базе данных. Репозиторий создается с таким же именем как у сущности и наследуется от JpaRepository. JpaRepository

это интерфейс spring data framework который предоставляет стандартный набор методов JPA, для работы с базой данных.

При создании репозиториев нужно придерживаться нескольких правил:

1) имя репозитория должно начинаться с имени сущности. Это необязательное правило, но его желательно придерживаться для удобства и для того чтобы код был понятней;

2) второй generic в JpaRepository должен быть оберточным типом того типа, которым является ID нашей сущности. Это правило должно выполняться обязательно;

3) первый generic в JpaRepository должен быть объектом сущности для которой создается данный репозиторий, это означает что spring data должен предоставить реализацию методов для работы с этой сущностью. Это правило должно выполняться обязательно;

4) необходимо наследовать свой интерфейс от JpaRepository, либо, библиотека Spring-data не предоставит реализацию для нашего репозитория. Это правило должно выполняться обязательно.

4.2.4. Сервисы

Репозитории нужны для обеспечения доступа к данным базы данных PostgreSQL. Но использовать напрямую репозитории для получения данных на пользовательский интерфейс не принято и считается плохим тоном, для этих целей были придуманы сервисы (Services). Сервис – это класс языка java, который предоставляет с себя основную бизнес-логику. В основном сервис использует готовые репозитории или другие сервисы, для того чтобы предоставить конечные данные для пользовательского интерфейса. Сервисы вызываются в контроллерах и в конкретных методах предоставляется необходимая функциональность, реализованная в сервисах. В соответствующем контроллере обычно присутствует такое же количество методов, как и в сервисе, на каждый метод в контроллере прописывается свой путь, по которому к данному методу можно обратиться с помощью HTTP методов. Для разделения логики, удобства, правильности кода создается класс сервиса и соответствующий ему интерфейс, для обращения к

сервису из контроллера. После создания интерфейса он реализовывается в классе сервиса и реализовываются все методы, которые входят в него. После реализации интерфейса, сервис может реализовывать разную логику, могут подключаться разные репозитории, их может быть несколько, можно подключать java mail сервисы, либо другие сервисы. В общем здесь выполняется вся бизнес логика для приложения, а контроллеры просто вызывают сервисы. Используя данный подход получается читабельные контроллеры и довольно структурированная бизнес логика. В будущем бизнес логика может свободно наращиваться и приложение свободно и без проблем может масштабироваться.

Пример контроллера и сервиса для получения данных новостной ленты изображен на рисунке 13 и 14.

```
@RestController
@RequestMapping("/get")
public class NewsController {

    @Autowired
    private NewsService service;

    @RequestMapping(value = "/news", method = RequestMethod.GET)
    @ResponseBody
    public List<News> getAllData(){
        return service.getAll();
    }
    @RequestMapping(value = "/news/{id}", method = RequestMethod.GET)
    @ResponseBody
    public News getNewsByID(@PathVariable("id") long newsID) {
        return service.getByID(newsID);
    }
    @RequestMapping(value = "/news/save", method = RequestMethod.POST)
    @ResponseBody
    public News saveNews(@RequestBody News news) {
        service.save(news);
    }
    @RequestMapping(value = "/reminders/{id}", method =
            RequestMethod.DELETE)
    @ResponseBody
    public void delete(@PathVariable long id) {
        service.remove(id);
    }

}
```

Рис. 13 Контроллер новостей

```java
@Service
public class NewsServiceImpl implements NewsService {

    @Autowired
    private NewsRepository repository;

    public List<News> getAll() {
        return repository.findAll();
    }

    public News getByID(long id) {
        return repository.findOne(id);
    }

    public void save(News news) {
        repository.saveAndFlush(news);
    }

    public void remove(long id) {
        repository.delete(id);
    }
}
```

Рис. 14. Сервис новостей

Остальная часть сервера реализуется идентичным образом с помощью контроллеров, репозиториев, сервисов и сущностей.

4.3. Реализация клиента приложения

Для реализации клиента приложения используется IDE Android Studio. Данная IDE была выбрана из-за ряда преимуществ:

1) разработчиком IDE является google inc. Которая выпускает саму платформу. Это дает удобную интеграцию функций для новых версий android;

2) IDE имеет встроенный android SDK [17], что дает нам возможность работать со всеми версиями API, притом все необходимые настройки будут установлены автоматически;

3) конструктор интерфейсов. Все визуализировано, имеется возможность просмотреть отображение экрана на примере любого устройства, соблюдаются все характеристики выбранного устройства. Визуализация будет выглядеть как на конкретной версии операционной системы;

4) представление структуры проекта;

5) удобный дизайн;

6) удобное отслеживание ошибок.

4.3.1. Хранение данных

Для хранения наборов данных, которые обычно представлены в виде ключ-значение, в android приложении обычно используют файл настроек. Эти настройки реализуются классом SharedPreferences [2]. Объекты этого класса представляют методы для обмена и хранения данных, и дают возможность выбрать необходимый тип файла частный или общий. В данных файлах настроек возможно хранение таких типов данных как: String, StringSet, Integer, Float, Long, Boolean. Основное назначение данных файлов, обмен данными и хранение контекста приложения и пользовательских настроек.

Для передачи данных между страницами приложения использовался метод putExtra, он фактически является словарем данные в котором представлены в виде пары ключ-значение, вызов данного метода осуществляется с помощью класса Intent [3]. Метод putExtra поддерживает передачу числовых, строковых и логических значений.

При реализации приложения в зависимости от определенных условий использовались файлы настроек или объекты класса intent. Эти методы применялись к данным, которые часто требовались для использования в приложении. К таким данным относятся хранящиеся данные о пользователе, они используются для предоставления доступа пользователю к приложению. Для предоставления данных о пользователе в необходимых формах, например, форме регистрации для участия в любительском заезде на гоночной трассе.

Когда пользователь проходит авторизацию, все необходимые данные пользователя полученные от сервера, такие как имя пользователя, e-mail, идентификатор пользователя, записываются в файл настроек и по необходимости вызываются и используются в приложении.

При регистрации пользователя в трек днях из файла настроек передается вся необходимая информация о пользователе и заполняется форма регистрации билета, после чего пользователь может забронировать билет.

4.3.2. Взаимодействие с REST сервисами

Для взаимодействия с Restful web-сервисом клиент приложения использует библиотеку Spring for Android [13] RestTemplate. Для того чтобы клиент мог принимать данные от REST сервиса создается класс представителя, который определяет поля. Класс создается полностью идентичным образом как на сервере, то есть класс должен быть полной копией “Entity” которая находится на сервере, соответственно без аннотаций, которые описывают что класс - это сущность. Подключается библиотека spring for android RestTemplate с помощью сборщика проектов gradle. RestTemplate дает возможность использовать библиотеку Jackson, которая позволяет очень удобно работать с JSON объектами. Jackson является очень мощным обработчиком для java. Так как все передаваемые данные имеют формат JSON библиотека Jackson тоже подключается.

Для получения и обработки получаемых от REST-сервиса данных используется класс AsyncTask [1]. AsyncTask позволяет забыть про работу с потоками, он предоставляет возможность работать напрямую с данными, лишь распределяя по переопределенным методам функционал и получаемые данные. Для использования AsyncTask создается класс наследник, переопределяются необходимые методы, и в необходимом месте обычно при инициализации графического объекта или в обработчике нажатия кнопки вызывается метод execute для начала работы метода. Класс не просто наследуется, он наследуется от типизированной версии AsyncTask <Void, Void, List<NewsDTO>>. Унаследованная версия типизируется тремя типами:

1) класс хранящий информацию для обработки задачи;

2) тип объектов, использующихся для индикации процесса выполнения задачи;

3) тип результата задачи.

Типы наследуемого класса могут представляться разными классами. В нашем приложении в основном потребуется класс списка получаемых объектов и отправляемых объектов. Для сервиса новостей будет использоваться тип

List<NewsDTO> потому что он будет возвращать приложению список переданных REST-сервисом объектов. Так как не нужно будет получать информацию об обработке задачи и идентифицировать процесс выполнения задачи для первых параметров устанавливается тип Void.

AsyncTask в себе содержит четыре метода, которые возможно переопределить:

1) doInBackground – метод выполняющийся в фоновом потоке, данный метод возвращает результат своей работы;

2) onPreExecute – метод вызывающийся из главного потока перед запуском doInBackground;

3) onProgressUpdate – метод позволяющий предоставлять информацию о выполняемом фоновом потоке;

4) onPostExecute – выполняется из главного потока после того как завершит свою работу метод doInBackground.

В приложении используется метод doInBackground для того чтобы принимать и обрабатывать данные отправленные REST сервисом. После того как данные будут обработаны будет исползован метод onPostExecute, чтобы передать уже обработанные и готовые для использования данные в приложение.

В методе doInBackground выполняются все операции по получению или отправке данных, а также их обработке. Первое что нужно использовать это шаблонный класс RestTemplate который предоставляется Spring. RestTemplate используется создания HTTP запроса но так ка все получаемые данные имеют формат json используется конвертер который нам предоставляет библиотека Jackson. Запрос на получение данных выполняется с помощью метода getForObject. Первым параметром метода getForObject является адрес сервиса, который установлен в контроллере нужного сервиса. Вторым параметром является тип принимаемых данных. После получения нужных данных нужно их обработать чтобы привести к нужному для получения виду. В методе onPostExecute получаются уже готовые данные возвращенные методом doInBackground они вставляются в при-

ложение. Для отправки данных используется метод postForObject. Первым параметром метода является адрес сервиса, который установлен в контроллере нужного нам сервиса, вторым параметром является передаваемый REST-сервису объект, третьим параметром является тип возвращенного ответа сервера. Данные методы класса AsyncTask позволяют с легкостью работать с веб-сервисами, не думая о работе с потоками. Также spring framework сильно облегчает работу разработчикам так как отлично работает с библиотекой Jackson. Не нужно сериализовывать десериализовывать данные, все это происходит автоматически.

Пример получения и записи данных с помощью класса AsyncTask изображены на рисунке 15 и 16.

```
class PostDataTask2 extends AsyncTask<Void, Void, String> {

    @Override
    protected String doInBackground(Void... unused) {
        template.getMessageConverters().add(new MappingJackson2HttpMessageConverter());
        String msg  = template.postForObject(URL2, user, String.class);
        return msg;
    }
    @Override
    protected void onPostExecute(String s) {
        String msg = s;
        Toast toast = Toast.makeText(getApplicationContext(),msg,Toast.LENGTH_LONG);
        toast.show();
    }
}
```

Рис. 15. Запись данных с помощью AsyncTask

```
private class NewsTask extends AsyncTask<Void, Void, List<NewsDTO>> {

    @Override
    protected List<NewsDTO> doInBackground(Void... params) {
        RestTemplate template = new RestTemplate();
        template.getMessageConverters().add(new MappingJackson2HttpMessageConverter());
        NewsDTO[] mydatas = template.getForObject(Constants.URL.GET_NEWS, NewsDTO[].class);
        List<NewsDTO> list = new ArrayList<>();
        for(int i = mydatas.length-1 ; i >= 0  ; i--){
            list.add(mydatas[i]);
        }
        return list;
    }

    @Override
    protected void onPostExecute(List<NewsDTO> newsDTO) {
        adapter.setData(newsDTO);
    }
}
```

Рис. 16. получение данных с помощью AsyncTask

4.3.3. Система сборки проекта gradle

Для сборки проекта использовался gradle. Gradle [21] является инструментом сборки, который интегрирован в среду разработки Android studio. В Android Studio сборка проекта осуществляется автоматически в фоновом режиме, сборка основана на наборе конфигурационных файлов. Gradle помогает разработчику легко управлять зависимостями приложения и устанавливать их из репозитория, также есть возможность создавать разные варианты сборок.

Файл конфигурации build.gradle содержит в себе информацию о версии приложения, требования к версии Android. На рисунке 17 изображен исходный код файла конфигурации, разрабатываемого приложения.

Файл конфигурации также включает в себя список используемых сторонних библиотек. На рисунке 18 изображен файл конфигурации, включающий используемые сторонние библиотеки, которые используются в разрабатываемом приложении.

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 25
    buildToolsVersion '25.0.2'
    defaultConfig {
        applicationId "com.example.raven.demoapp"
        minSdkVersion 15
        targetSdkVersion 25
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
    packagingOptions {
        exclude 'META-INF/notice.txt'
        exclude 'META-INF/license.txt'
        exclude 'META-INF/NOTICE'
        exclude 'META-INF/LICENSE'
    }
}
```

Рис. 17. Код файла конфигурации build.gradle

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support.constraint:constraint-layout:1.0.2'
    compile 'com.android.support:appcompat-v7:25.3.1'
    compile 'com.android.support:design:25.3.1'
    compile 'com.android.support:cardview-v7:25.3.1'
    compile 'com.android.support:recyclerview-v7:25.3.1'
    compile 'org.springframework.android:spring-android-rest-template:1.0.1.RELEASE'
    compile 'com.fasterxml.jackson.core:jackson-databind:2.8.7'
    compile 'com.squareup.picasso:picasso:2.5.1'
    testCompile 'junit:junit:4.12'
}
```

Рис. 18. Список подключаемых библиотек в build.gradle

4.3.4. Интерфейс приложения

Разработанное приложения позволяет пользователю регистрироваться в заездах, которые проводятся трек-дни. Пользователь имеет возможность просматривать календарь событий трека, новостную ленту и характеристик трасс. Приложение предназначено для любителей автоспорта, оно должно облегчить процесс регистрации в заездах и предложить некоторую полезную функциональность.

При запуске приложения пользователь может авторизоваться в приложение с помощью данных своей учетной записи или создать учетную запись, заполнив форму (рис. 19) регистрации пользователя в соответствии с условиями регистрации. Проверка пользователя сервером выполняется с помощью специальных средств фреймворка spring, точнее spring security. Если все введенные данные корректные, пользователь переходит на главную страницу приложения. При входе вся необходимая информация о пользователе сохраняется для возможности дальнейших операций.

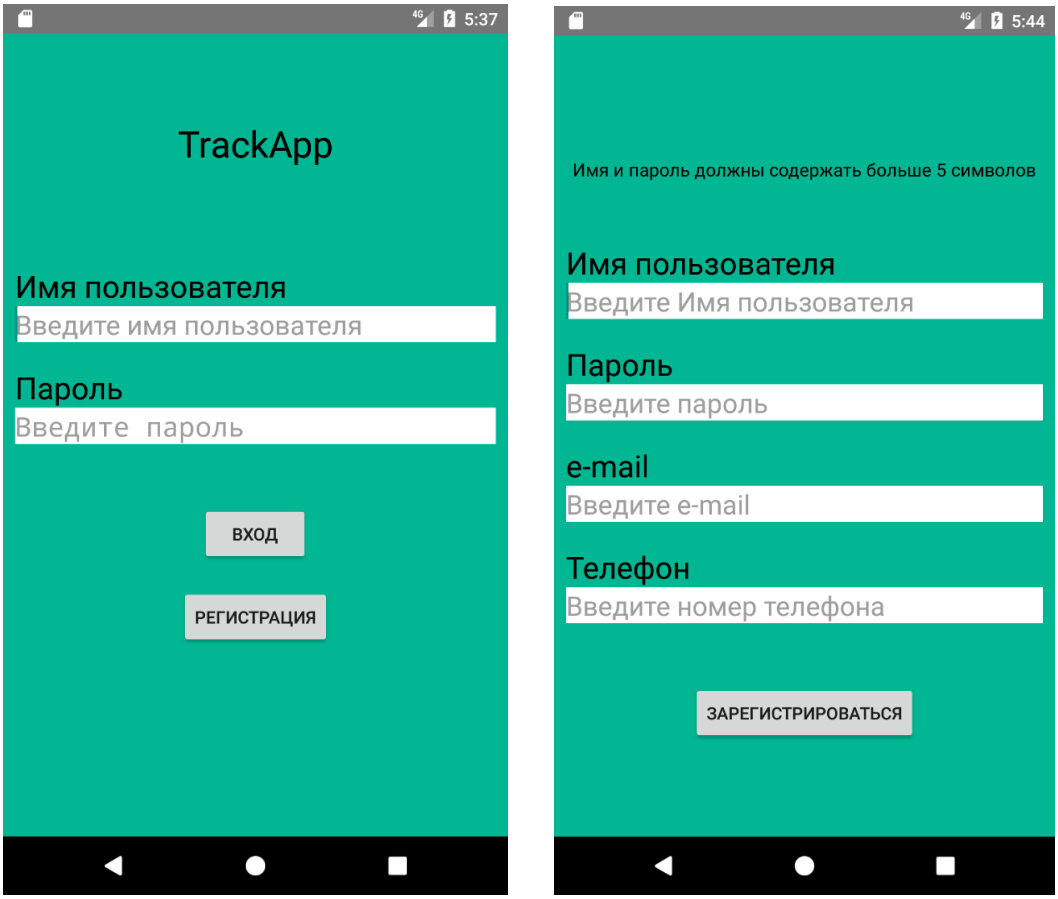

Рис. 19. Меню авторизации и регистрации

На главной странице (рис. 20) пользователю отображаются новости автоспорта и конкретно данного трека, новости имеют заголовок, фото или картинку и описание самой новости. При переходе на вкладки данные обновляются динамически, после добавления администратором новости, она будет отображена при переходе на меню новостей.

Для навигации в приложение предназначено боковое меню (рис.20). Меню нажимается по нажатию кнопки, либо с помощью бокового свайпа. Каждый пункт меню предоставляет свою функциональность, имеет название и иконку.

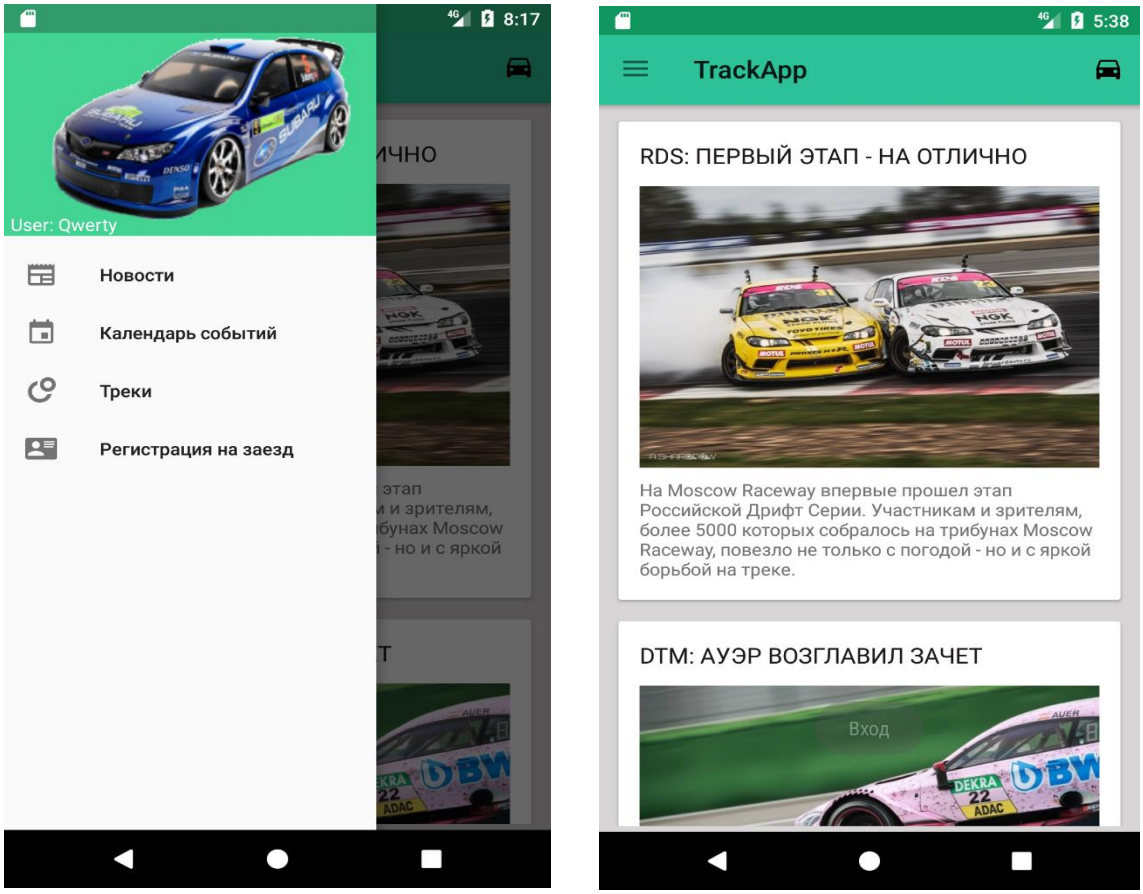

Рис. 20. Меню навигации и начальная страница

С помощью календаря событий (рис. 21) пользователь может получить информацию о датах проведения событий, которые будут проведены на треке в ближайшее время, данный календарь отображает не только проведение трекдней. Календарь событий может информировать пользователей о любых событиях. Например, о соревнованиях по дрифту, мото-заездах и других возможных

событиях на треке. Также в календарные события содержат сопутствующую информацию о событие.

В меню треки (рис. 21) пользователь может ознакомится со всеми трассами трека и узнать их конфигурацию, перед тем как совершить заезд, будет не лишним ознакомиться с трассой.

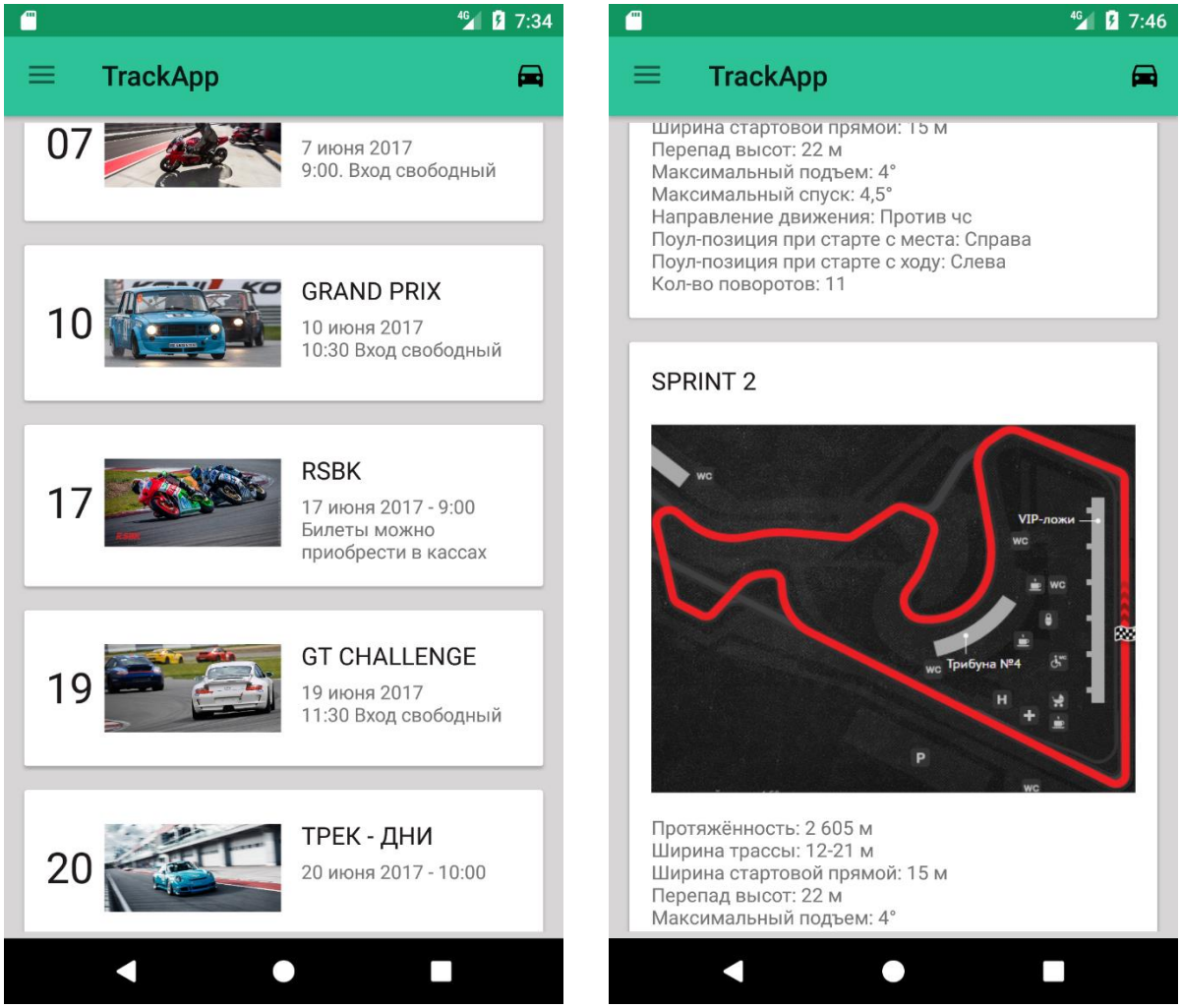

Рис. 21. Меню календарь событий и треки

Для регистрации в трек днях предназначена форма (рис. 22). В форме у пользователя есть возможность выбрать необходимую дату трек-дня и время заезда. Все данные формы заполняются динамически. В меню можно выбрать только возможные даты проведения заездов, так как они назначаются организаторами мероприятий. После выбора необходимой даты и времени заезда пользователя перенаправит в форму подтверждения бронирования билета.

В форме подтверждения автоматически заполняются необходимые поля. Поля содержат информацию о пользователе: имя, почта, телефон; информацию про заезд: дату проведения, номер и время заезда, и дату бронирования билета. Чтобы подтвердить бронирование билета, пользователь должен нажать кнопку забронировать, после чего система оповестит пользователя о результате операции.

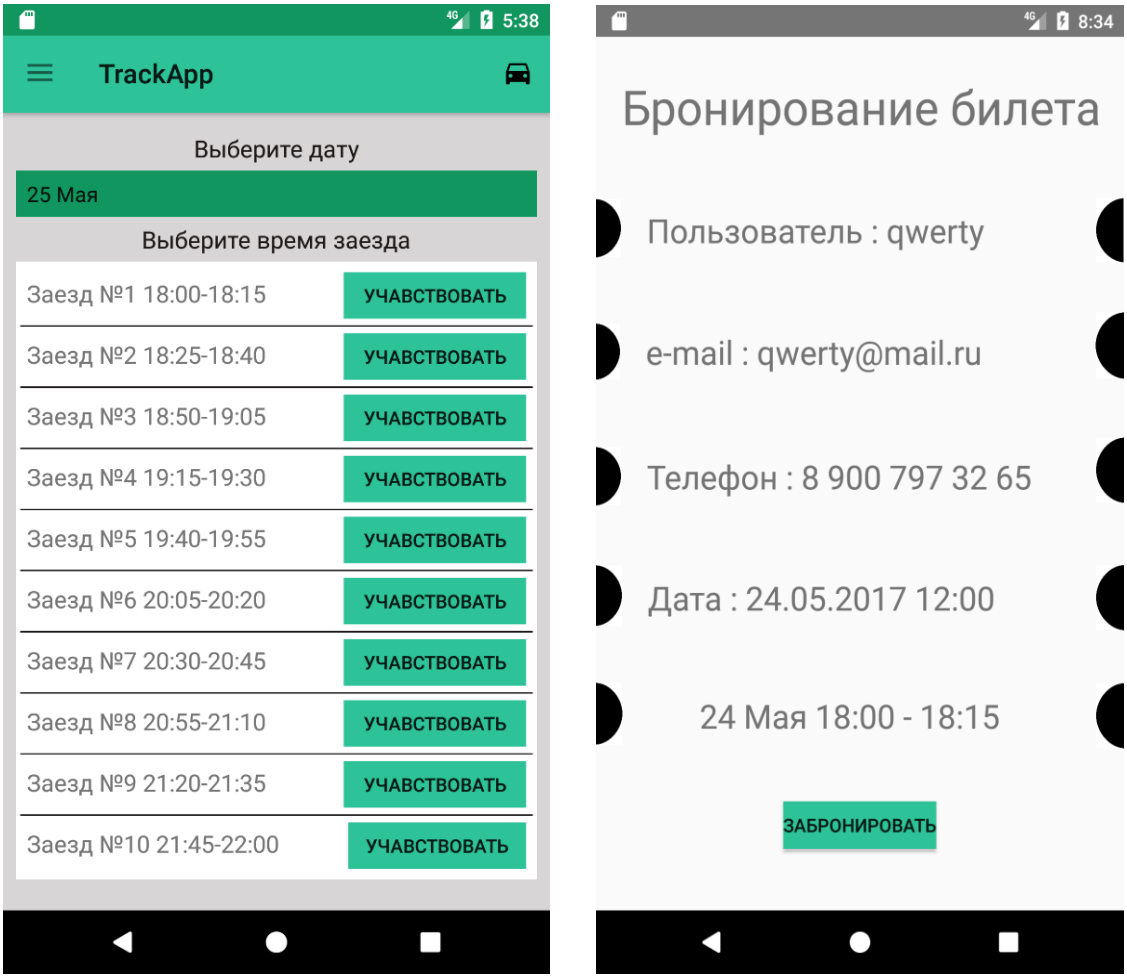

Рис. 22. Меню бронирования билета

Вывод

В результате разработано мобильное приложение TrackApp базирующееся на операционной системе Android. Разработанное мобильное приложение полностью соответствует функциональным и нефункциональным требованиям, установленным для данной системы.

5. ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

Тестирование REST-сервиса проводилось с помощью автоматизированных средств тестирования. Модульные тесты были написаны при помощи библиотеки spring-boot-starter-test. Тесты созданы для каждого метода REST-сервисов. Все вызываемые методы находятся в контроллерах, поэтому тестирование выполняется именно в них. Так как помощью аннотации @Autowired в контроллер инжектируется сервис, появляется возможность использовать его внутри класса контроллера. В классе тестирования такой возможности нет. Для решения этой проблемы используется библиотека mockito. Библиотека mockito поставляется вместе с spring-boot-starter-test. Перед классом тестирования прописывается аннотация @RunWith(MockitoJunitRunner.class), которая является стандартной для JUnit тестов, внутри аннотации подключается mockito. После подключения mockito появляется возможность создания макета сервиса и с помощью аннотации @Mock создается макет сервиса. Для инжектирования макета сервиса используется аннотация @InjectMocks, после чего создается экземпляр класса тестируемого контроллера со всеми входящими в него зависимостями и инжектируются заданные макеты. После выполненных манипуляций, начинается тестирование контроллера. Проверяется вызов содержащихся в сервисе методов. Проверка вызова методов выполняется с помощью Mockito.verify, таким образом тестируется сервисы. На рисунке 23 приведен код тестирования контроллера новостного сервиса. В результате тестирования контроллеров REST сервисов все написанные модульные тесты для каждого сервиса были пройдены успешно.

Тестирование вызова HTTP методов проводилось с помощью HTTP – клиента Postman. Postman предоставляет возможность создавать простые и сложные HTTP-запросы. Удобное меню приложения позволяет сохранять все требуемые запросы. Также он позволяет просматривать результат запроса, который возвращает сервер. Таким образом были протестированы все HTTP методы сервисов.

```
package com.example.raven.app.server.controller;

import ...

@RunWith(MockitoJUnitRunner.class)
public class NewsControllerTest {
    @Mock
    private NewsService service;
    @InjectMocks
    NewsController sut;

    @Test
    public void testGetAllData() throws Exception {
        //prepare
        when(service.getAll()).thenReturn(ImmutableList.<News>of());
        //testing
        List<News> listOfNews = sut.getAllData();
        //validate
        verify(service).getAll();
    }

}
```

Рис. 23. Тестирование контроллера новостного сервиса

Тестирование пользовательского интерфейса мобильного приложения проводилось вручную на физическом устройстве. При тестировании были найдены ошибки отображения интерфейса, ошибки перехода между страницами и фрагментами. Тестировался весь основной функционал приложения такой как авторизация, регистрация, бронирование билетов. Тестирование проводилось в течение длительного времени. Все найденные ошибки при тестировании были своевременно устранены.

Для проведения тестирования совместной работы мобильного клиента и сервера приложения было проведено интеграционное тестирование сервиса. Основная задача интеграционного тестирования – поиск дефектов, связанных с ошибками в реализации и интерпретации интерфейсного взаимодействия между модулями. В результате проведения интеграционного тестирования было выявлено, что разработанные по отдельности компоненты корректно взаимодействуют друг с другом, представляя собой единый программный продукт.

ЗАКЛЮЧЕНИЕ

В данной научно-исследовательской работе было предложено решение по разработке приложения для регистрации и участия в любительских гоночных заездах, основанное на использовании микро сервисной архитектуры. Были изучены технологии spring, hibernate.

Было проведено исследование по поиску аналогичных уже существующих решений для регистрации и участия в любительских гоночных заездах, были рассмотрены их основные характеристики и назначение, а также выявлены принципиальные недостатки.

В данной работе также было дано детальное описание разрабатываемого сервиса, сформулированы функциональные и нефункциональные требования к разрабатываемой системе. Были выявлены основные актеры, взаимодействующие с системой, и действия, которые можно совершать над системой. На основании этих данных была построена диаграмма прецедентов, а также дано детальное описание каждого указанного на диаграмме актера и прецедента.

Была спроектирована архитектура разрабатываемой системы, состоящая из трех основных компонентов: мобильное приложение, REST-сервис и сервер БД. Все компоненты являются слабосвязанными, и изменение одного из них не влияет на другие.

В результате была выполнена реализация всех компонентов приложения в соответствии с функциональными и нефункциональными требованиями, предъявляемыми к системе. В том числе был реализован REST-сервис, предоставляющий необходимый REST API для работы клиентских приложений.

Было проведено модульное тестирование компонентов сервиса, интеграционное тестирования для проверки совместной работы компонентов и тестирование интерфейса приложения.

ЛИТЕРАТУРА

1. Android Development: AsyncTask. [Электронный ресурс] URL: https://developer.android.com/reference/android/os/AsyncTask.html (дата обращения: 20.04.2017).

2. Android Development: SharedPreferences. [Электронный ресурс] URL:https://developer.android.com/reference/android/content/SharedPreferences.html (дата обращения: 18.04.2017).

3. Android Development: Intent. [Электронный ресурс] URL: https://developer.android.com/reference/android/content/Intent.html (дата обращения: 18.04.2017).

4. Apache Commons. [Электронный ресурс] URL: https://commons.apache.org/proper/commons-dbcp/apidocs/index.html (дата обращения: 20.12.2016).

5. JPA provider Hibernate. [Электронный ресурс] URL: http://hibernate.org/orm/documentation/5.2 (дата обращения: 20.12.2016).

6. Obe R., Hsu L. PostgreSQL: Up and Running. – USA: O'Reilly, 2012. – 168 с.

7. Picaso library for Android. [Электронный ресурс] URL: http://square.github.io/picasso (дата обращения: 22.04.2017).

8. Richardson L., Ruby S. RESTful Web Services. – USA: O'Reilly, 2007. – 440 с.

9. Sonatype Company, Maven: The Definitive Guide. – USA: O'Reilly, 2008. – 468 с.

10. Spring: MessageConverter. [Электронный ресурс] URL: http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/converter/json/MappingJackson2HttpMessageConverter.html (дата обращения: 28.04.2017).

11. Spring: REST-service. [Электронный ресурс] URL: https://spring.io/guides/gs/rest-service (дата обращения: 28.04.2017).

12. Spring: Spring for Android. [Электронный ресурс] URL: http://projects.spring.io/spring-android (дата обращения: 28.04.2017).

13. Spring: spring MVC. [Электронный ресурс] URL: https://docs.spring.io/spring-boot/docs/current/reference/html/howto-spring-mvc.html (дата обращения: 28.04.2017).

14. Spring-security: PasswordEncoder. [Электронный ресурс] URL: http://docs.spring.io/spring-security/site/docs/current/apidocs/org/springframework/security/crypto/bcrypt/BCryptPasswordEncoder.html (дата обращения: 28.04.2017).

15. Tomcat Apache. [Электронный ресурс] URL: http://tomcat.apache.org/tomcat-9.0-doc/deployer-howto.html (дата обращения: 03.05.2017).

16. Walls C., Spring in Action. – USA: Manning, 2015. – 745 с.

17. Дейтел П., Дейтел Х., Уолд А. Android 6 for Programmers: An App-Driven Approach. Пер. с англ. – СПб.: Питер, 2016. – 512. с.

18. Официальный сайт гоночного трека Moscow-raceway. [Электронный ресурс] URL: http://moscowraceway.ru (дата обращения: 20.12.2016).

19. Официальный сайт гоночного трека Nring. [Электронный ресурс] URL: http://nring.ru (дата обращения: 20.12.2016).

20. Официальный сайт гоночного трека Smolenskring. [Электронный ресурс] URL: http:// smolenskring.ru (дата обращения: 20.12.2016).

21. Официальный веб-сайт инструмента сборки Android-проектов Gradle. [Электронный ресурс] URL: http://gradle.org/getting-started-android (дата обращения: 08.04.2017).

22. Шилдт Г. Java 8. Полное руководство; 9-е изд.: Пер. с англ. – М.: ООО "И.Д. Вильяме", 2015. – 1376 с.