МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационные системы и технологии

Специальность 1-98 01 03 «Программное обеспечение информационной

безопасности мобильных систем»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

Разработка программного средства «Event-manager»

Выполнил студент Стефаненко Максим Сергеевич

(Ф.И.О.)

Руководитель проекта асс. Колмаков М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой д.к.т.н., Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультант асс. Колмаков М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтроллер асс. Колмаков М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2020

**Реферат**

Пояснительная записка курсового проекта содержит 20 страниц, 13 изображений, 7 источников литературы, 2 приложения.

Основными целями курсового проекта являются: разработка программного средства – «Event-manager».

Пояснительная записка состоит из введения, пяти разделов, заключения.

Во введении представлена общая информация, дающая представление о предстоящей работе, определены цели.

В первом разделе представлена постановка задачи.

Во втором разделе рассматривается разработка архитектуры проекта.

В третьем разделе рассматривается разработка функциональной модели приложения.

В четвертом разделе представлено руководство пользователя.

В заключении представлены итоги курсового проектирования и задачи, которые были решены в ходе разработки программного средства.

Содержание

[Введение 4](#_Toc41248625)

[1. Постановка задачи. 5](#_Toc41248626)

[2. Архитектура проекта 6](#_Toc41248627)

[3. Схема базы данных 7](#_Toc41248628)

[4. Программная реализация приложения 8](#_Toc41248629)

[4.1 Реализация сервера 8](#_Toc41248630)

[4.2 Реализация мобильного приложения 9](#_Toc41248631)

[4.2.1 Реализация REST 10](#_Toc41248633)

[5. Руководство пользователя 13](#_Toc41248634)

[5.1 Android приложение 13](#_Toc41248635)

[Заключение 19](#_Toc41248636)

[Список использованных источников 20](#_Toc41248637)

[Приложение А 21](#_Toc41248638)

[Приложение B 23](#_Toc41248639)

# Введение

Мобильное приложение — это программное обеспечение, специально разработанное под конкретную мобильную платформу (iOS, Android, Windows Phone и т. д.). Предназначено для использования на смартфонах, фаблетах, планшетах, умных часах и других мобильных устройствах.

Основные плюсы мобильных приложений: интерфейс программы создан конкретно под работу на мобильном устройстве через сенсорный экран или кнопки,

удобная и понятная для пользователей гаджетов навигация, мобильное меню, лучшее взаимодействие с пользователем через сообщения, пуш-уведомления, напоминания. Приложение может выполнять функции даже в фоновом режиме, чего нельзя сказать о сайте. Для работы с программой не нужно открывать браузер, а многие приложения поддерживают ряд функций и при отключенном интернете.

Темой курсового проекта является разработка программного средства «Event-manager» - мобильного приложения, основной задачей которого является создание различных событий, просмотра пользователей, которые записались к вам на событие, и возможности записаться на разнообразные события, созданные другими пользователями.

Целью курсового проекта является проектирование и реализация программного средства «Event-manager».

# Постановка задачи.

ASP.NET (Active Server Pages для .NET) — платформа разработки [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5), в состав которой входит: [веб-сервисы](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81), программная инфраструктура, модель программирования, от компании [Майкрософт](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D0%B9%D0%BA%D1%80%D0%BE%D1%81%D0%BE%D1%84%D1%82). ASP.NET входит в состав платформы [.NET Framework](https://ru.wikipedia.org/wiki/Microsoft_.NET" \o "Microsoft .NET) и является развитием более старой технологии [Microsoft ASP](https://ru.wikipedia.org/wiki/Active_Server_Pages" \o "Active Server Pages). Добавить андройд студию и джаву

Android Studio – интегрированная среда разработки для работы с платформой Android. Основана на программном обеспечении IntelliJ IDEA от JetBrains, доступна для Windows, macOS, Linux.

Java – строго типизированный объектно-ориентированный язык программирования, приложения на Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре.

Microsoft SQL Server — [система управления реляционными базами данных (РСУБД)](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%A1%D0%A3%D0%91%D0%94" \o "Реляционная СУБД), разработанная корпорацией [Microsoft](https://ru.wikipedia.org/wiki/Microsoft" \o "Microsoft). Основной используемый язык запросов — [Transact-SQL](https://ru.wikipedia.org/wiki/Transact-SQL" \o "Transact-SQL), создан совместно Microsoft и [Sybase](https://ru.wikipedia.org/wiki/Sybase" \o "Sybase). Transact-SQL является реализацией стандарта [ANSI](https://ru.wikipedia.org/wiki/%D0%90%D0%BC%D0%B5%D1%80%D0%B8%D0%BA%D0%B0%D0%BD%D1%81%D0%BA%D0%B8%D0%B9_%D0%BD%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%B8%D0%BD%D1%81%D1%82%D0%B8%D1%82%D1%83%D1%82_%D1%81%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BE%D0%B2)/[ISO](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B6%D0%B4%D1%83%D0%BD%D0%B0%D1%80%D0%BE%D0%B4%D0%BD%D0%B0%D1%8F_%D0%BE%D1%80%D0%B3%D0%B0%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D0%BF%D0%BE_%D1%81%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%B8) по структурированному языку запросов ([SQL](https://ru.wikipedia.org/wiki/SQL)) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

SQL Server Management Studio (SSMS) — утилита из [Microsoft SQL Server](https://ru.wikipedia.org/wiki/Microsoft_SQL_Server" \o "Microsoft SQL Server) 2005 и более поздних версий для конфигурирования, управления и [администрирования](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%BD%D0%BE%D0%B5_%D0%B0%D0%B4%D0%BC%D0%B8%D0%BD%D0%B8%D1%81%D1%82%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) всех компонентов Microsoft SQL Server. Утилита включает [скриптовый](https://ru.wikipedia.org/wiki/%D0%A1%D0%BA%D1%80%D0%B8%D0%BF%D1%82) редактор и графическую программу, которая работает с объектами и настройками сервера.

Главным инструментом SQL Server Management Studio является Object Explorer, который позволяет пользователю просматривать, извлекать объекты сервера, а также полностью ими управлять.

ADO.NET Entity Framework (EF) — объектно-ориентированная технология доступа к данным, является [object-relational mapping](https://ru.wikipedia.org/wiki/ORM" \o "ORM) (ORM) решением для [.NET Framework](https://ru.wikipedia.org/wiki/.NET_Framework" \o ".NET Framework) от [Microsoft](https://ru.wikipedia.org/wiki/Microsoft). Предоставляет возможность взаимодействия с объектами как посредством [LINQ](https://ru.wikipedia.org/wiki/LINQ) в виде LINQ to Entities, так и с использованием Entity SQL.

При разработке сервера следует реализовать следующие API:

* REST – это стиль архитектуры программного обеспечения для распределенный систем, таких как WorldWideWeb, который, как правило используется для построения веб-служб.
* SOAP – протокол обмена структурированными сообщениями в распределенной вычислительной среде. Первоначально предназначался для реализации удаленного вызова процедур, сейчас используется для обмена сообщениями в формате XML или JSON.

Клиентом будет выступать мобильное приложение на устройстве с операционной системой Android 4.0.3 и выше.

# Архитектура проекта

Распределенное приложение состоит из 2-х частей:

Мобильное приложение. Эта часть состоит из Java-классов, с помощью которых реализована основная логика мобильного приложения и осуществляется обращение к серверу.

Сервер. Данная часть предоставляет ASP.NET приложение, которое обрабатывает запросы с клиента с помощью контроллеров и взаимодействует с базой данных посредством Entity Framework.

Графическое представление архитектуры распределенного приложения изображено на рисунке 2.1.

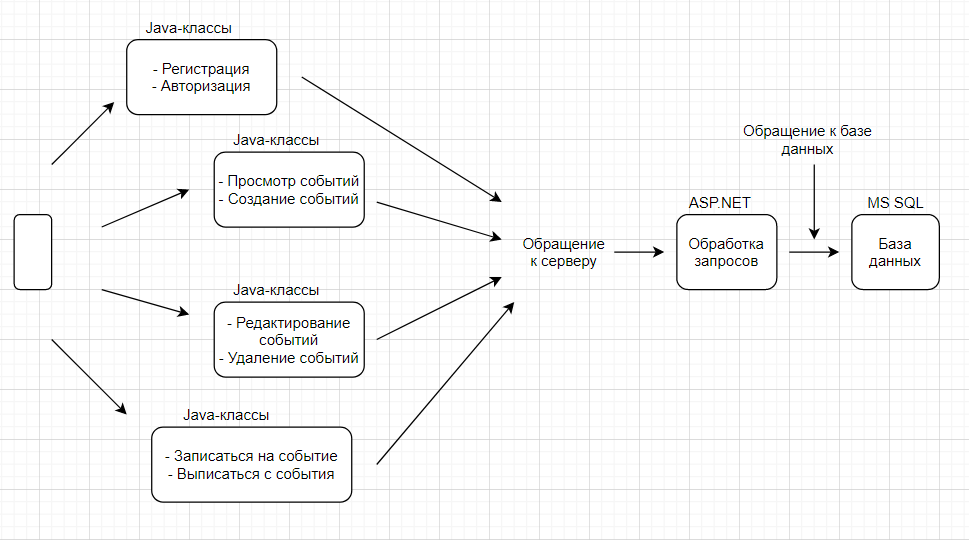


Рисунок. 2.1 – архитектура распределенного приложения

# Схема базы данных

В ходе написания приложения была разработана удаленная база данных «CourseProject»,схема которой представлена на рисунке 3.1.

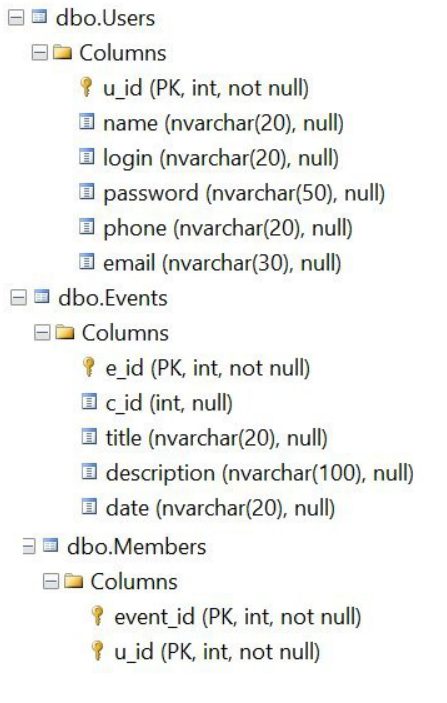


Рисунок 3.1 – структура базы данных

Таблица Users предназначена для хранения пользователей, зарегистрированных в приложении. Events – для хранении информации о созданных событиях, о том, кто их создал, и тд. Таблица Members такую важную информацию, как: какой пользователь записан на какой событие.

# Программная реализация приложения

## 4.1 Реализация сервера

Поскольку ASP.NET основывается на [Common Language Runtime](https://ru.wikipedia.org/wiki/Common_Language_Runtime" \o "Common Language Runtime) (CLR), которая является основой всех приложений Microsoft .NET, разработчики могут писать код для ASP.NET, используя языки программирования, входящие в комплект [.NET Framework](https://ru.wikipedia.org/wiki/.NET_Framework" \o ".NET Framework) ([C#](https://ru.wikipedia.org/wiki/C_Sharp), [Visual Basic.NET](https://ru.wikipedia.org/wiki/VB.NET), J# и [JScript .NET](https://ru.wikipedia.org/wiki/JScript_.NET" \o "JScript .NET)). ASP.NET имеет преимущество в скорости по сравнению со скриптовыми технологиями, так как при первом обращении код компилируется и помещается в специальный [кэш](https://ru.wikipedia.org/wiki/%D0%9A%D1%8D%D1%88), и впоследствии только исполняется, не требуя затрат времени на синтаксический анализ, [оптимизацию](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D1%82%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D0%BA%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80%D0%B0) и т. д.

Программная модель ASP.NET основывается на протоколе [HTTP](https://ru.wikipedia.org/wiki/HTTP) и использует его правила взаимодействия между сервером и браузером. При формировании страницы заложена абстрактная программной модель Web Forms, и на ней основана основная часть реализации программного кода.

Графическая структура классов представлена на рисунке 4.1.1.

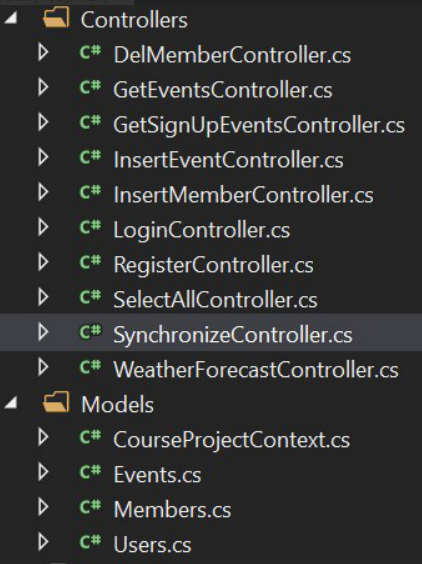


Рисунок 4.1.1 – структура сервера

Сервер имеет следующие пути для обработки HTTP-запросов (Приложение А)

* Get – получение курсовых работ или пользователей;
* Delete – для удаления пользователя или работы;
* Post – добавление информации курсовой работе или пользователе;
* Put – обновление информации.

## 4.2 Реализация мобильного приложения

Мобильное приложение реализовано для операционной системы Android 4.0.3 и выше. Данное приложение было разработано в специальной среде разработки –Android Studio.

Графическая структура классов представлена на рисунке 4.2.1.

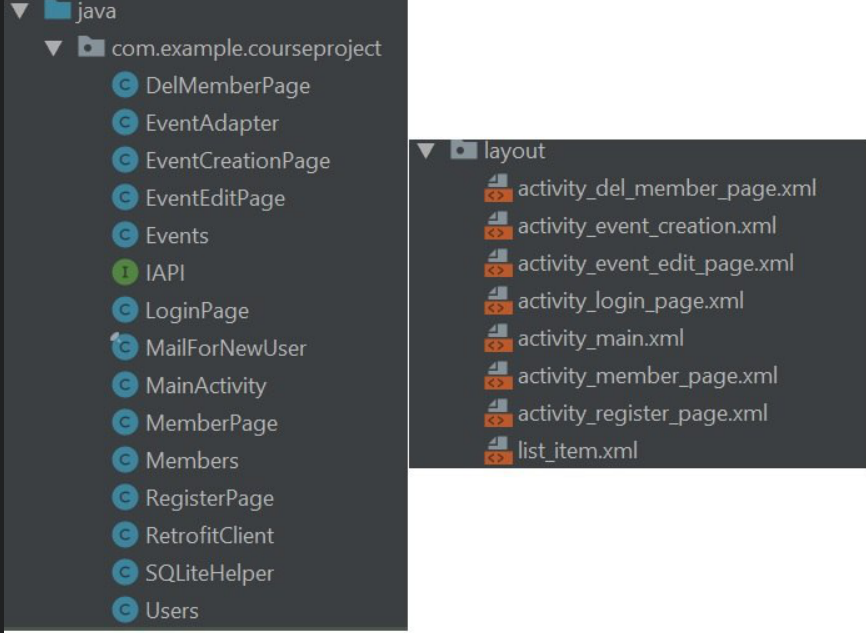


Рисунок 4.2.1 – структура проекта

### Проект в себе содержит папку java, в которой хранятся все классы, необходимые для работы приложения.

1. DelMemberPage – содержит реализацию методов для выписки пользователя с события, на которое он был записан.
2. EventAdapter – класс, представляющий собой адаптер для списка пользователей.
3. EventCreationPage – класс, содержащий реализацию для создания события и автоматической записи туда пользователя, создавшего событие.
4. EventEditPage – реализация методов для изменения или удаления созданного события, также содержит реализацию для отображения списка пользователей, записавшихся на событие.
5. Events – модель данных для событий.
6. IAPI – интерфейс, предоставляющий API методы для взаимодействия с сервером.
7. LoginPage – класс, представляющий собой набор функциональности, для аутентификации и авторизации пользователя в приложении.
8. MailForNewUser – класс, позволяющий при регистрации пользователя отправить ему приветственное письмо.
9. MainActivity – основной класс в приложении, при помощи него отображаются списки событий на которые пользователь зарегистрирован или нет.
10. MemberPage – содержит реализацию методов для записи пользователя на событие.
11. Members – модель данных для записанных на события пользователей.
12. RegisterPage – класс, содержащий набор методов, необходимых для регистрации пользователя в приложении.
13. RetrofitClient – содержит в себе параметры и различные настройки для соединения с сервером.
14. SQLiteHelper – предназначен для упрощения взаимодействия с локальной базой данных, предоставляет готовые методы для манипуляций с базой данных.
15. Users – модель данных для пользователей.

В папке layout находятся активности, которые представляют собой xml файлы, содержащие разметку. С помощью этой разметки в свою очередь отрисовываются макеты экранов, которые мы будем видеть при запуске приложения.

Все активности похожи своим наполнением, при этом отрисовываются по-разному, то есть используя похожую xml-разметку удается воспроизводить различные элементы на экране и обеспечить взаимодействие пользователя с программой.

### **4.2.1 Реализация REST**

Для взаимодействия мобильного приложения с сервером применяется архитектурный стиль REST. Основная задача сервера в этой архитектуре предоставить клиентам доступ к ресурсам по их идентификаторам. Под доступом подразумевается как получение информации, так и ее изменение или удаление. В зависимости от реализации сервиса, в ответ на наш запрос придёт ответ в формате JSON. На рисунке 4.2.1.1 представлен пример одного из get-запросов.

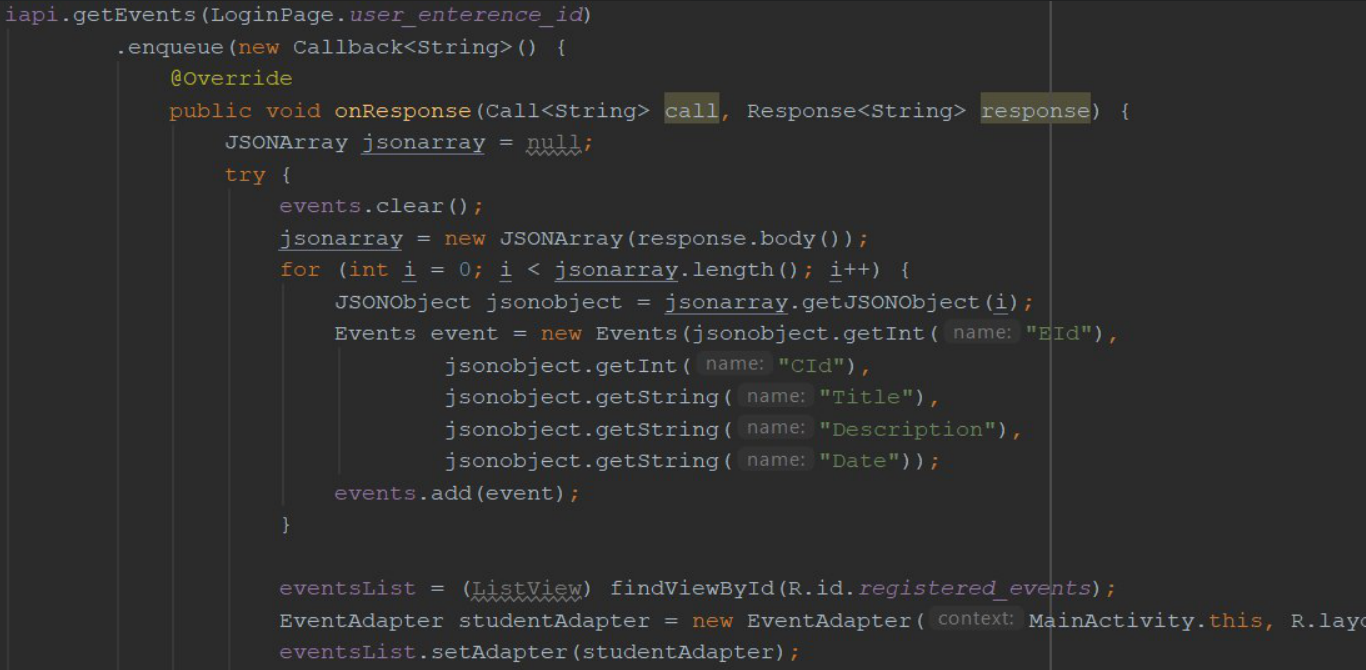


Рисунок 4.2.1.1 – get-запрос

В данном методе можно увидеть, что в него передаётся ID, по которому затем будет производить get-запрос. Внутри метода есть объект response, который будет содержать ответ от сервера, в нашем случае JSON-файл, содержащий объекты типа Events.

На рисунке 4.2.1.2 приведен пример post-запроса.

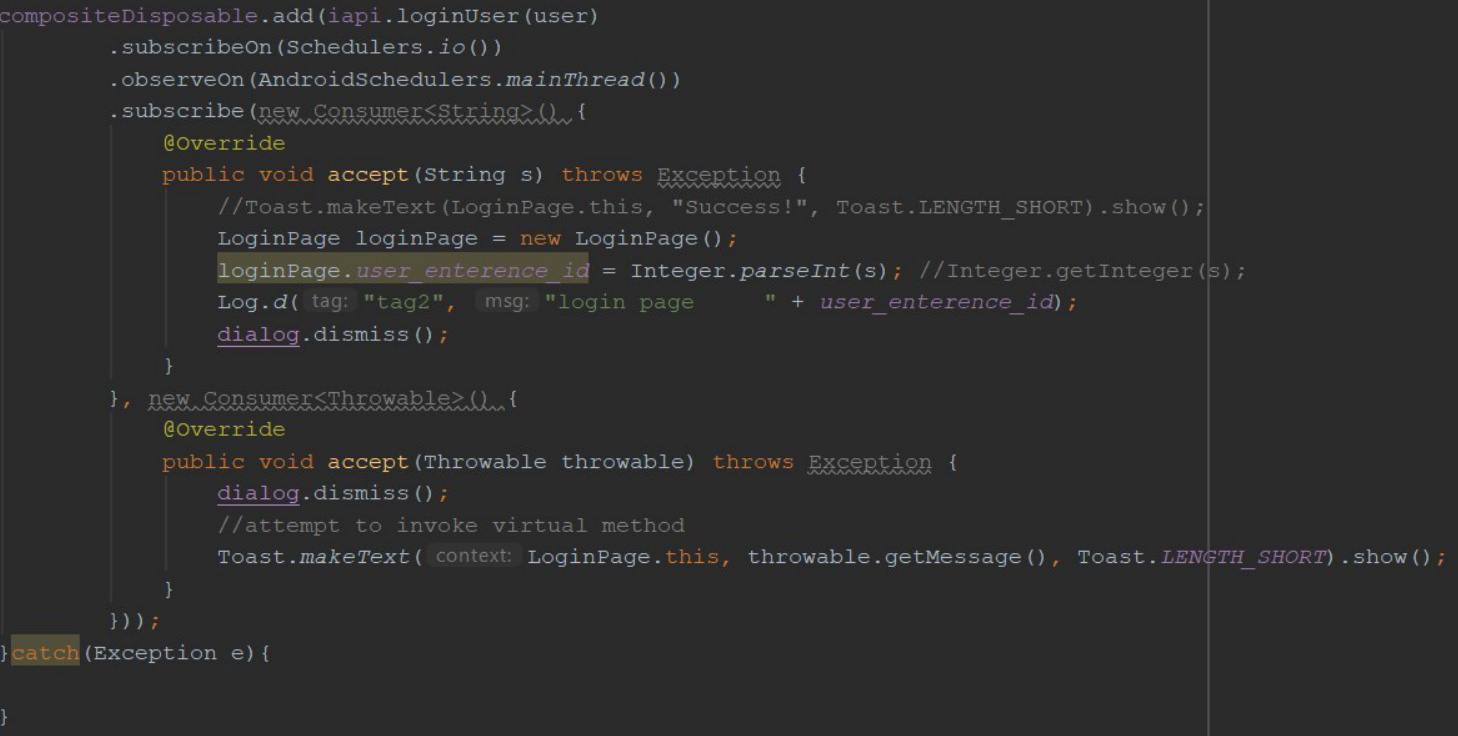


Рисунок 4.2.1.2 – post-запроса

В данном примере в метод передаётся объект User, содержащий логин и пароль пользователя.

Затем в случае успеха на клиент придет ID пользователя, которое в дальнейшем будет использоваться для различных манипуляций с приложением.

GET – позволяет получить данные, а также конкретный ответ по заданным параметрам. POST – позволяет передать данные на сервер.

Таким образом можно сказать, что REST – это взаимодействие «клиент-серверной» архитектуры с помощью HTTP-запросов и HTTP-ответов по протоколу, не подтверждающему состояние.

# Руководство пользователя

## 5.1 Android приложение

Для начала необходимо установить приложение на мобильный устройство с API не меньше 4.0.3 (Android 6.0). Запустим наше приложение. Сразу после запуска мы увидим окно авторизации (Рисунок 5.1.1).

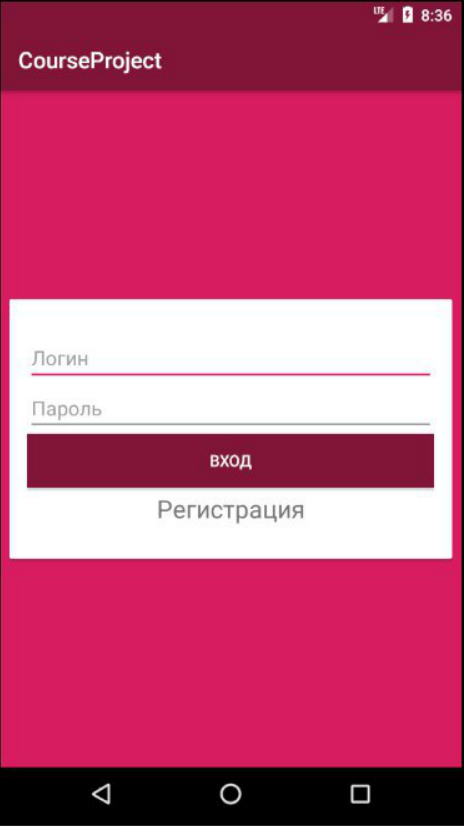


Рисунок 5.1.1 – окно авторизации

Если пользователь еще не зарегистрирован в системе, то необходимо нажать на поле «Регистрация» и произойдет переход на нужную активность (Рисунок 5.1.2).

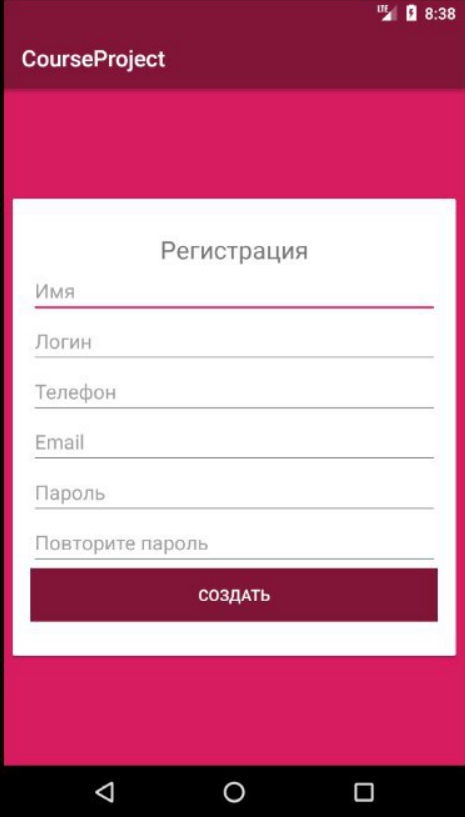
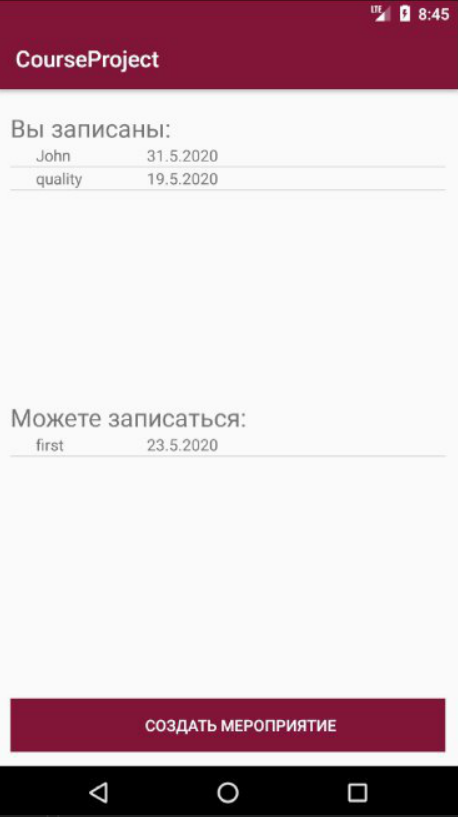
******

Рисунок 5.1.2 – окно регистрации

Если успешно пройти авторизацию пользователя на первой странице, то произойдет переход на главную страницу, содержащую два списка: список событий, на которые пользователь уже зарегистрирован и список событий, на которые пользователь может записаться. Также можно обнаружить кнопку «создать мероприятие», при нажатии на которую произойдет переход на страницу создания события. (Рисунок 5.1.3).



Риснок 5.1.3 – главная страница

При нажатии на кнопку «создать мероприятие» происходит переход на страницу создания события, где пользователь может ввести такие данные как: название события, дата (выбирается нажатием на слово «дата», при нажатии открывается календарь и предоставляется возможность выбрать дату) и описание события. (Рисунок 5.1.4).

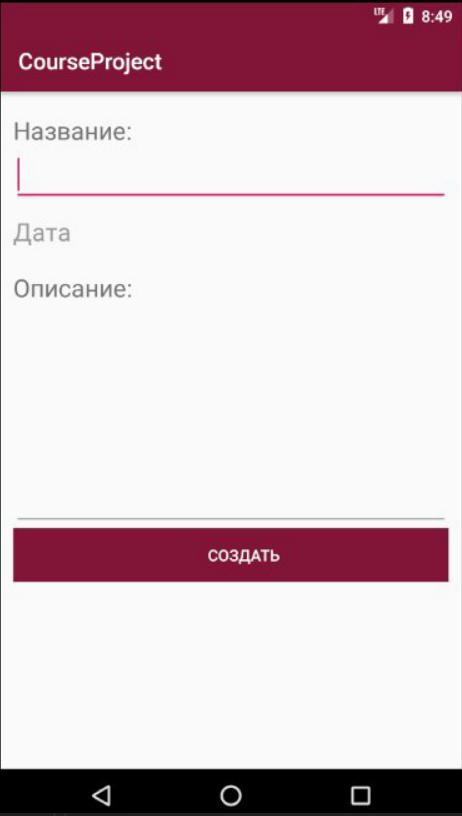


Рисунок 5.1.4 – создание события

После ввода данных и нажатии кнопки «создать» произойдет создание события и переход на главный экран приложения.

Если нажать на элемент списка (имя события) в верхнем списке, то приложение определит является ли пользователь, вошедший в приложение, создателем выбранного события или нет. В случае если пользователь является создателем, его переадресует на форму редактирования события, где также можно будет увидеть список записавшихся пользователей. (Рисунок 5.1.5).

Если не является, то пользователя перекинет на страницу, где он сможет выписаться с события. (Рисунок 5.1.6).

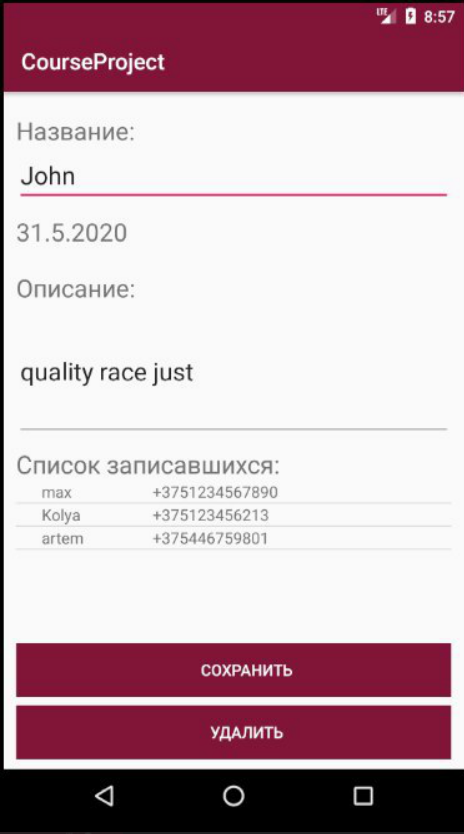


Рисунок 5.1.5 – редактирование события

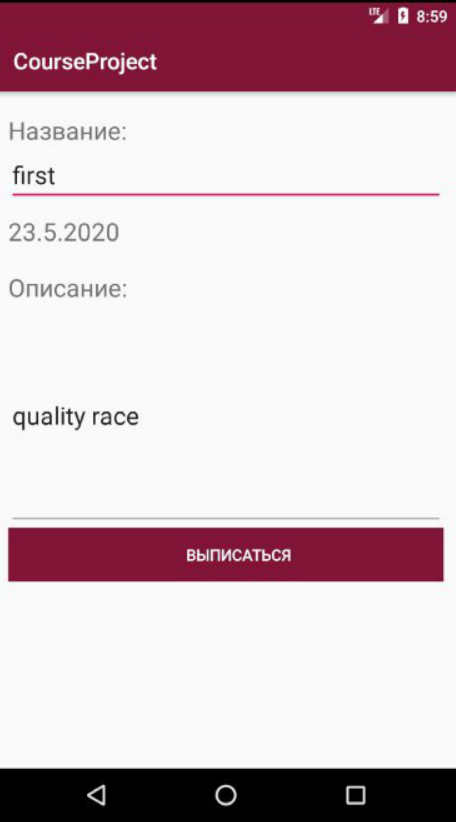


Рисунок 5.1.6 – выписывание с события

Если же произвести нажатие на элемент нижнего списка (список событий, на которые пользователь не записан), то произойдет переход на экран, где пользователю будет предоставлена более полная информация о событии и также пользователю представится возможность записаться на событие. (Рисунок 5.1.7).

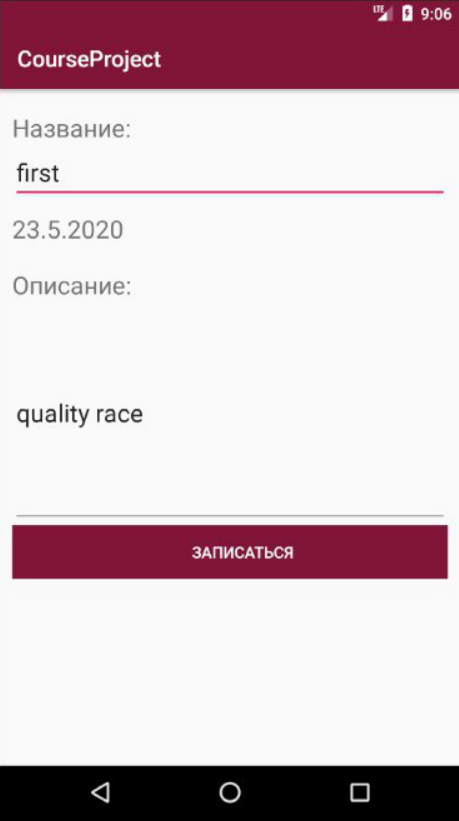


Рисунок 5.1.7 – записывание на событие

Если обобщить все вышесказанное и показанное, то на данных скриншотах приведена работа приложение и с помощью текста изложены все нюансы при работе с ним.

# Заключение

Решая поставленную задачу, я пришел к удовлетворительному результату. Таким образом, была достигнута цель, и был создан проект «Event-manager», который включает в себя android–клиент и сервер.

Программная система имеет простой, интуитивно понятный пользователю интерфейс.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанный продукт работает корректно, а требования технического задания выполнены в полном объеме.

В результате приложение обладает следующими особенностями:

* получение данных с сервера;
* реализована отправка данных на удалённый сервер по технологии REST API;
* реализован полностью работоспособный сервер и удалённая база данных.

# Список использованных источников

1. Stackoverflow - https://stackoverflow.com;

2. Cyberforum - https://cyberforum.ru;

3. Github - https://github.com;

4. Android documentation - <https://developer.android.com/docs>.

5. Java documentation - <https://docs.oracle.com/en/java/>

6. ASP.NET documentation - <https://docs.microsoft.com/en-us/aspnet/core/>

7. SQLServer documentation - <https://docs.microsoft.com/en-us/sql/sql-server/>

# Приложение А

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Newtonsoft.Json;

using Server.Models;

// For more information on enabling Web API for empty projects, visit https://go.microsoft.com/fwlink/?LinkID=397860

namespace Server.Controllers

{

[Route("api/[controller]/{id:int}/{id2:int}")]

//[Route("api/[controller]")]

public class DelMemberController : Controller

{

public void Delete(int id, int id2)

{

using (CourseProjectContext dbContext = new CourseProjectContext())

{

try

{

Microsoft.Data.SqlClient.SqlParameter param = new Microsoft.Data.SqlClient.SqlParameter("@eid", id);

Microsoft.Data.SqlClient.SqlParameter param2 = new Microsoft.Data.SqlClient.SqlParameter("@uid", id2);

dbContext.Events.FromSqlRaw("delmember @eid, @uid", param, param2).ToList();

}

catch(Exception e)

{

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Server.Models;

using Microsoft.EntityFrameworkCore;

using Newtonsoft.Json;

// For more information on enabling Web API for empty projects, visit https://go.microsoft.com/fwlink/?LinkID=397860

namespace Server.Controllers

{

[Route("api/[controller]")]

public class GetEventsController : Controller

{

CourseProjectContext dbContext = new CourseProjectContext();

[HttpGet("{id}")]

public string Get(int id)

{

using (CourseProjectContext dbContext = new CourseProjectContext())

{

Microsoft.Data.SqlClient.SqlParameter param = new Microsoft.Data.SqlClient.SqlParameter("@id", id);

var events = dbContext.Events.FromSqlRaw("getevents @id", param).ToList();

return JsonConvert.SerializeObject(events);

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Newtonsoft.Json;

using Server.Models;

// For more information on enabling Web API for empty projects, visit https://go.microsoft.com/fwlink/?LinkID=397860

namespace Server.Controllers

{

[Route("api/[controller]")]

public class InsertEventController : Controller

{

CourseProjectContext dbContext = new CourseProjectContext();

// POST api/<controller>

[HttpPost]

public void Post([FromBody]Events value)

{

Events events = new Events();

events.CId = value.CId;

events.Title = value.Title;

events.Description = value.Description;

events.Date = value.Date;

try

{

dbContext.Add(events);

dbContext.SaveChanges();

}

catch (Exception e)

{

}

}

// PUT api/<controller>/5

//[HttpPut("{id}")]

public string Put([FromBody]Events value)

{

Events events = dbContext.Events.Where(u => u.EId.Equals(value.EId)).First();

//events.EId = value.EId;

//events.CId = value.CId;

events.Title = value.Title;

events.Description = value.Description;

events.Date = value.Date;

try

{

dbContext.Update(events);

dbContext.SaveChanges();

return JsonConvert.SerializeObject("User has been registered!");

}

catch (Exception ex)

{

return JsonConvert.SerializeObject(ex.Message);

}

}

// DELETE api/<controller>/5

[HttpDelete("{id}")]

public void Delete(int id)

{

try

{

Events events = dbContext.Events.FirstOrDefault(u => u.EId.Equals(id));

dbContext.Events.Remove(events);

dbContext.SaveChangesAsync();

}

catch (Exception e)

{

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Server.Models;

// For more information on enabling Web API for empty projects, visit https://go.microsoft.com/fwlink/?LinkID=397860

namespace Server.Controllers

{

[Route("api/[controller]")]

public class InsertMemberController : Controller

{

CourseProjectContext dbContext = new CourseProjectContext();

// POST api/<controller>

[HttpPost]

public void Post([FromBody]Members value)

{

Members member = new Members();

member.EventId = value.EventId;

member.UId = value.UId;

try

{

dbContext.Add(member);

dbContext.SaveChanges();

}

catch (Exception e)

{

}

}

// DELETE api/<controller>/5

[HttpDelete("{id}")]

public void Delete(int id)

{

try

{

Microsoft.Data.SqlClient.SqlParameter param = new Microsoft.Data.SqlClient.SqlParameter("@eid", id);

dbContext.Events.FromSqlRaw("delmembers @eid", param).ToList();

}

catch (Exception e)

{

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Newtonsoft.Json;

using Server.Models;

// For more information on enabling Web API for empty projects, visit https://go.microsoft.com/fwlink/?LinkID=397860

namespace Server.Controllers

{

[Route("api/[controller]")]

public class LoginController : Controller

{

CourseProjectContext dbContext = new CourseProjectContext();

// POST api/<controller>

[HttpPost]

public string Post([FromBody]Users value)

{

if (dbContext.Users.Any(user => user.Login.Equals(value.Login)))

{

Users user = dbContext.Users.Where(user => user.Login.Equals(value.Login)).First();

if (user.Password.Equals(value.Password))

{

return JsonConvert.SerializeObject(user.UId);

}

else

{

return JsonConvert.SerializeObject("Wrong password!");

}

}

else

{

return JsonConvert.SerializeObject("User not existing!");

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Newtonsoft.Json;

using Server.Models;

// For more information on enabling Web API for empty projects, visit https://go.microsoft.com/fwlink/?LinkID=397860

namespace Server.Controllers

{

[Route("api/[controller]")]

public class RegisterController : Controller

{

CourseProjectContext dbContext = new CourseProjectContext();

// POST api/<controller>

[HttpPost]

public string Post([FromBody]Users value)

{

if(!dbContext.Users.Any(user => user.Login.Equals(value.Login)))

{

Users user = new Users();

user.Name = value.Name;

user.Login = value.Login;

user.Password = value.Password;

user.Phone = value.Phone;

user.Email = value.Email;

try

{

dbContext.Add(user);

dbContext.SaveChanges();

return JsonConvert.SerializeObject("User has been registered!");

}

catch(Exception e)

{

return JsonConvert.SerializeObject(e.Message);

}

}

else

{

return JsonConvert.SerializeObject("User is existing!");

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Newtonsoft.Json;

using Server.Models;

namespace Server.Controllers

{

[Route("api/[controller]")]

public class SynchronizeController : Controller

{

// GET api/<controller>/5

public string Get( )

{

using (CourseProjectContext dbContext = new CourseProjectContext())

{

return JsonConvert.SerializeObject(dbContext.Members.Count());

}

}

}

}

# Приложение B

package com.example.courseproject;  
  
import androidx.appcompat.app.AppCompatActivity;  
  
import android.content.Intent;  
import android.database.sqlite.SQLiteDatabase;  
import android.os.Bundle;  
import android.util.Log;  
import android.view.View;  
import android.widget.EditText;  
import android.widget.TextView;  
import android.widget.Toast;  
  
import io.reactivex.android.schedulers.AndroidSchedulers;  
import io.reactivex.disposables.CompositeDisposable;  
import io.reactivex.functions.Consumer;  
import io.reactivex.schedulers.Schedulers;  
  
public class DelMemberPage extends AppCompatActivity {  
  
 String title;  
 String description;  
 String date;  
 TextView date\_del\_member\_page;  
 int id;  
  
 IAPI iapi;  
 CompositeDisposable compositeDisposable = new CompositeDisposable();  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_del\_member\_page*);  
  
 iapi = RetrofitClient.*getInstance*().create(IAPI.class);  
  
 Intent intent = getIntent();  
 id = Integer.*decode*(intent.getStringExtra(MainActivity.*EXTRA\_EV\_ID*));  
 title = intent.getStringExtra(MainActivity.*EXTRA\_EV\_TITLE*);  
 description = intent.getStringExtra(MainActivity.*EXTRA\_EV\_DESCR*);  
 date = intent.getStringExtra(MainActivity.*EXTRA\_EV\_DATE*);  
 EditText title\_text = (EditText)findViewById(R.id.*title\_del\_member*);  
 title\_text.setText(title);  
 EditText title\_descr = (EditText)findViewById(R.id.*description\_del\_member*);  
 title\_descr.setText(description);  
  
 date\_del\_member\_page = findViewById(R.id.*date\_del\_member\_page*);  
 date\_del\_member\_page.setText(date);  
 }  
  
 public void UnregisterMember(View view){  
 SQLiteDatabase db = getBaseContext().openOrCreateDatabase("app.db", *MODE\_PRIVATE*, null);  
  
 db.execSQL("delete from members where e\_id = "+id+" and u\_id = "+LoginPage.*user\_enterence\_id*+";");  
  
 compositeDisposable.add(iapi.delMember(id,LoginPage.*user\_enterence\_id*)  
 .subscribeOn(Schedulers.*io*())  
 .observeOn(AndroidSchedulers.*mainThread*())  
 .subscribe(new Consumer<String>() {  
 @Override  
 public void accept(String s) throws Exception {  
 //Toast.makeText(DelMemberPage.this, s, Toast.LENGTH\_SHORT).show();  
 Log.*d*("tag", s);  
 }  
 }, new Consumer<Throwable>() {  
 @Override  
 public void accept(Throwable throwable) throws Exception {  
 Log.*d*("tag", throwable.getMessage());  
 //Toast.makeText(DelMemberPage.this, throwable.getMessage(), Toast.LENGTH\_SHORT).show();  
 }  
 }));  
  
 Intent intent = new Intent(this, MainActivity.class);  
 startActivity(intent);  
 }  
}

package com.example.courseproject;  
import android.content.Context;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.ArrayAdapter;  
import android.widget.TextView;  
  
import androidx.annotation.NonNull;  
  
import java.util.List;  
  
public class EventAdapter extends ArrayAdapter<Events>{  
  
 private LayoutInflater inflater;  
 private int layout;  
 private List<Events> events;  
  
 public EventAdapter(Context context, int resource, List<Events> events){  
 super(context, resource, events );  
 this.events = events;  
 this.layout = resource;  
 this.inflater = LayoutInflater.*from*(context);  
 }  
  
 public EventAdapter(@NonNull Context context, int resource) {  
 super(context, resource);  
 }  
  
 public View getView(int position, View convertView, ViewGroup parent) {  
  
 View view=inflater.inflate(this.layout, parent, false);  
  
 TextView nameView = (TextView) view.findViewById(R.id.*title\_on\_list*);  
 TextView nameView2 = (TextView) view.findViewById(R.id.*date\_on\_list*);  
  
 Events event = events.get(position);  
  
 nameView.setText(event.getTitle());  
 nameView2.setText(event.getDate());  
 return view;  
 }  
}

package com.example.courseproject;  
  
import androidx.appcompat.app.AppCompatActivity;  
  
import android.app.DatePickerDialog;  
import android.app.Dialog;  
import android.content.Intent;  
import android.database.Cursor;  
import android.database.sqlite.SQLiteDatabase;  
import android.os.Bundle;  
import android.util.Log;  
import android.view.View;  
import android.widget.DatePicker;  
import android.widget.EditText;  
import android.widget.TextView;  
import android.widget.Toast;  
  
import java.text.SimpleDateFormat;  
import java.util.Calendar;  
import java.util.Locale;  
  
import io.reactivex.android.schedulers.AndroidSchedulers;  
import io.reactivex.disposables.CompositeDisposable;  
import io.reactivex.functions.Consumer;  
import io.reactivex.schedulers.Schedulers;  
  
public class EventCreationPage extends AppCompatActivity {  
  
 public static SQLiteHelper *sqLiteHelper*;  
 String id;  
  
 TextView tvDate;  
 int currentYear = Calendar.*getInstance*().get(Calendar.*YEAR*);  
 int currentMonth = Calendar.*getInstance*().get(Calendar.*MONTH*);  
 int currentDay = Calendar.*getInstance*().get(Calendar.*DAY\_OF\_MONTH*);  
  
 IAPI iapi;  
 CompositeDisposable compositeDisposable = new CompositeDisposable();  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_event\_creation*);  
  
 iapi = RetrofitClient.*getInstance*().create(IAPI.class);  
  
 *sqLiteHelper* = new SQLiteHelper(this, "app.db", null, 1);  
 *sqLiteHelper*.queryData("CREATE TABLE IF NOT EXISTS events " +  
 "(e\_id INTEGER PRIMARY KEY AUTOINCREMENT," +  
 " c\_id INTEGER," +  
 " title TEXT," +  
 " description TEXT," +  
 " date TEXT)");  
  
 *sqLiteHelper*.queryData("CREATE TABLE IF NOT EXISTS members " +  
 "(e\_id INTEGER," +  
 " u\_id INTEGER)" );  
  
 Intent intent = getIntent();  
 //Toast.makeText(getApplicationContext(), "id " + intent.getStringExtra(LoginPage.EXTRA\_ID) , Toast.LENGTH\_SHORT).show();  
 id = intent.getStringExtra(LoginPage.*EXTRA\_ID*);  
 //id = intent.getStringExtra(MainActivity.EXTRA\_EV\_ID);  
 //Log.d("tag", id);  
  
  
 tvDate = findViewById(R.id.*tvDate*);  
  
 }  
  
  
 public void CreateEvent(View view){  
 EditText title = (EditText)findViewById(R.id.*event\_title*);  
 EditText description = (EditText)findViewById(R.id.*description*);  
 //EditText date = (EditText)findViewById(R.id.event\_date);  
  
 //LoginPage loginPage = new LoginPage();  
 //int id2 = Integer.decode(MainActivity.user\_enterence\_id);  
 int id2 = LoginPage.*user\_enterence\_id*;  
  
 //int id2 = Integer.decode(id);  
  
 Log.*d*("tag", id2+" "+title.getText().toString().trim()+" "+  
 " "+description.getText().toString().trim()+  
 " "+tvDate);  
 try{  
 *sqLiteHelper*.CreateEvent(  
 //Integer.getInteger(id),  
 id2,  
 title.getText().toString().trim(),  
 description.getText().toString().trim(),  
 tvDate.getText().toString()  
 );  
  
 Events event = new Events(0,id2,title.getText().toString().trim(),description.getText().toString().trim(),tvDate.getText().toString());  
 compositeDisposable.add(iapi.insertEvent(event)  
 .subscribeOn(Schedulers.*io*())  
 .observeOn(AndroidSchedulers.*mainThread*())  
 .subscribe(new Consumer<String>() {  
 @Override  
 public void accept(String s) throws Exception {  
 //Toast.makeText(EventCreationPage.this, s, Toast.LENGTH\_SHORT).show();  
 }  
 }, new Consumer<Throwable>() {  
 @Override  
 public void accept(Throwable throwable) throws Exception {  
 //Toast.makeText(EventCreationPage.this, throwable.getMessage(), Toast.LENGTH\_SHORT).show();  
 }  
 }));  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
  
 //sqLiteHelper.queryData("insert into events (e\_id, c\_id, title, description) values (99,99,'title','description')");  
// String str = "";  
// SQLiteDatabase db2 = getBaseContext().openOrCreateDatabase("app.db", MODE\_PRIVATE, null);  
// Cursor query2 = db2.rawQuery("SELECT \* FROM events;", null);  
// if(query2.moveToFirst()){  
// do{  
// str += query2.getLong(0)+" "+query2.getLong(1) +" " +query2.getString(2)+" "+query2.getString(3);  
// }  
// while(query2.moveToNext());  
// }  
//  
// String str2 = "";  
// SQLiteDatabase db3 = getBaseContext().openOrCreateDatabase("app.db", MODE\_PRIVATE, null);  
// Cursor query3 = db3.rawQuery("SELECT \* FROM members;", null);  
// if(query3.moveToFirst()){  
// do{  
// str2 += query3.getLong(0)+" "+query3.getLong(1)+ "|";  
// }  
// while(query3.moveToNext());  
// }  
// Log.d("tag", str2);  
 //Toast.makeText(getApplicationContext(), "chto ne tak "+str, Toast.LENGTH\_SHORT).show();  
  
  
  
 String str = "";  
  
 int event\_id = 0;  
 SQLiteDatabase db = getBaseContext().openOrCreateDatabase("app.db", *MODE\_PRIVATE*, null);  
 Cursor query = db.rawQuery("SELECT e\_id FROM events where c\_id = '"+id2+"';", null);  
 if(query.moveToFirst()){  
 do{  
 event\_id = query.getInt(0);  
 str += query.getInt(0) + " ";  
 }  
 while(query.moveToNext());  
 }  
 Log.*d*("tag", "eventcreation "+str +" u\_id "+id2 +" e\_id "+event\_id);  
// try{  
// sqLiteHelper.CreateMember(  
// event\_id,  
// id2  
// //Integer.getInteger(id)  
//  
// );  
// }  
// catch (Exception e){  
// e.printStackTrace();  
// }  
  
  
 db.execSQL("insert into members values ("+event\_id+","+id2+")");  
  
 Members member = new Members(event\_id, id2);  
 compositeDisposable.add(iapi.insertMember(member)  
 .subscribeOn(Schedulers.*io*())  
 .observeOn(AndroidSchedulers.*mainThread*())  
 .subscribe(new Consumer<String>() {  
 @Override  
 public void accept(String s) throws Exception {  
 //Toast.makeText(EventCreationPage.this, s, Toast.LENGTH\_SHORT).show();  
 }  
 }, new Consumer<Throwable>() {  
 @Override  
 public void accept(Throwable throwable) throws Exception {  
 //Toast.makeText(EventCreationPage.this, throwable.getMessage(), Toast.LENGTH\_SHORT).show();  
 }  
 }));  
  
 //Toast.makeText(getApplicationContext(), "Событие успешно создано!", Toast.LENGTH\_SHORT).show();  
 Intent intent = new Intent(this, MainActivity.class);  
 startActivity(intent);  
 }  
  
  
  
  
 //test calendar  
 public void datePickerHandler(View view) {  
 showDialog(1);  
 }  
  
 protected Dialog onCreateDialog(int id) {  
 if (id == 1) {  
 DatePickerDialog tpd = new DatePickerDialog(this, myCallBack, currentYear, currentMonth, currentDay);  
 return tpd;  
 }  
 return super.onCreateDialog(id);  
 }  
  
 DatePickerDialog.OnDateSetListener myCallBack = new DatePickerDialog.OnDateSetListener() {  
 public void onDateSet(DatePicker view, int year, int monthOfYear,  
 int dayOfMonth) {  
 currentYear = year;  
 currentMonth = monthOfYear + 1;  
 currentDay = dayOfMonth;  
 tvDate.setText(currentDay + "." + currentMonth + "." + currentYear);  
 }  
 };  
}

package com.example.courseproject;  
  
import androidx.appcompat.app.AppCompatActivity;  
  
import android.app.DatePickerDialog;  
import android.app.Dialog;  
import android.content.Intent;  
import android.database.Cursor;  
import android.database.sqlite.SQLiteDatabase;  
import android.net.Uri;  
import android.os.Bundle;  
import android.util.Log;  
import android.view.View;  
import android.widget.AdapterView;  
import android.widget.DatePicker;  
import android.widget.EditText;  
import android.widget.ListView;  
import android.widget.TextView;  
import android.widget.Toast;  
  
import java.util.ArrayList;  
import java.util.Calendar;  
import java.util.List;  
  
import io.reactivex.disposables.CompositeDisposable;  
import retrofit2.Call;  
import retrofit2.Callback;  
import retrofit2.Response;  
  
  
public class EventEditPage extends AppCompatActivity {  
  
 String title;  
 String description;  
 String date;  
 String u\_name;  
 String u\_phone;  
 String u\_email;  
  
 TextView date\_edit\_page;  
 int currentYear = Calendar.*getInstance*().get(Calendar.*YEAR*);  
 int currentMonth = Calendar.*getInstance*().get(Calendar.*MONTH*);  
 int currentDay = Calendar.*getInstance*().get(Calendar.*DAY\_OF\_MONTH*);  
  
 IAPI iapi;  
 CompositeDisposable compositeDisposable = new CompositeDisposable();  
  
 int id;  
 private List<Events> events = new ArrayList();  
 ListView eventsList;  
 public static SQLiteHelper *sqLiteHelper*;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_event\_edit\_page*);  
  
 iapi = RetrofitClient.*getInstance*().create(IAPI.class);  
  
 date\_edit\_page = findViewById(R.id.*date\_edit\_page*);  
  
 Intent intent = getIntent();  
 //Toast.makeText(getApplicationContext(), intent.getStringExtra(MainActivity.EXTRA\_EV\_ID) , Toast.LENGTH\_SHORT).show();  
 id = Integer.*decode*(intent.getStringExtra(MainActivity.*EXTRA\_EV\_ID*));  
 title = intent.getStringExtra(MainActivity.*EXTRA\_EV\_TITLE*);  
 description = intent.getStringExtra(MainActivity.*EXTRA\_EV\_DESCR*);  
 date = intent.getStringExtra(MainActivity.*EXTRA\_EV\_DATE*);  
 EditText title\_text = (EditText)findViewById(R.id.*event\_title\_edit*);  
 title\_text.setText(title);  
 EditText title\_descr = (EditText)findViewById(R.id.*description\_edit*);  
 title\_descr.setText(description);  
// EditText date\_text = (EditText)findViewById(R.id.event\_date\_edit);  
// date\_text.setText(date);  
 date\_edit\_page.setText(date);  
  
  
 events.clear();  
  
 SQLiteDatabase db = getBaseContext().openOrCreateDatabase("app.db", *MODE\_PRIVATE*, null);  
 Cursor query = db.rawQuery("SELECT users.name, users.email, users.phone FROM users inner join members " +  
 "on users.u\_id = members.u\_id where members.e\_id = "+id+";", null);  
  
 String str = "";  
 if(query.moveToFirst()){  
 do{  
 Events event = new Events(0,  
 0,  
 query.getString(0),  
 query.getString(1),  
 query.getString(2));  
 events.add(event);  
 str += query.getString(0);  
 }  
 while(query.moveToNext());  
 }  
 query.close();  
 db.close();  
  
  
 Log.*d*("tag","eventeditpage" + str);  
  
 eventsList = (ListView) findViewById(R.id.*members\_list*);  
 EventAdapter studentAdapter = new EventAdapter(this, R.layout.*list\_item*, events);  
 eventsList.setAdapter(studentAdapter);  
  
 AdapterView.OnItemClickListener itemListener = new AdapterView.OnItemClickListener() {  
 @Override  
 public void onItemClick(AdapterView<?> parent, View v, int position, long id) {  
 Events selectedState = (Events) parent.getItemAtPosition(position);  
 u\_name = String.*valueOf*(selectedState.getTitle());  
 u\_phone = String.*valueOf*(selectedState.getDate());  
 u\_email = String.*valueOf*(selectedState.getDescription());  
  
 Intent intent = new Intent(Intent.*ACTION\_DIAL*);  
 intent.setData(Uri.*parse*("tel:"+u\_phone));  
 startActivity(intent);  
  
// View view = new View(getApplicationContext());  
// GoToDetailsMemberPage(view);  
 }  
 };  
 eventsList.setOnItemClickListener(itemListener);  
  
 }  
  
 public void EditEvent(View view){  
 SQLiteDatabase db = getBaseContext().openOrCreateDatabase("app.db", *MODE\_PRIVATE*, null);  
  
 EditText title = (EditText)findViewById(R.id.*event\_title\_edit*);  
 EditText descriptiont = (EditText)findViewById(R.id.*description\_edit*);  
 //EditText date = (EditText)findViewById(R.id.event\_date\_edit);  
 String desc = descriptiont.getText().toString();  
 Log.*d*("tag", desc);  
  
 if(title.getText().toString().trim().equals("") || desc.trim().equals("")){  
 Toast.*makeText*(this, "Какое-то из полей пустое", Toast.*LENGTH\_SHORT*).show();  
 }  
 else{  
 db.execSQL("UPDATE events SET title = '"+title.getText()+"'," +  
 "date = '"+date\_edit\_page.getText()+"'," +  
 "description = '"+desc+"' WHERE e\_id = "+id+";");  
  
 Events event = new Events(id,0,title.getText().toString(),desc,date\_edit\_page.getText().toString());  
 //Update in remote db  
 Call<Events> call= iapi.updateEvent(event);  
 call.enqueue(new Callback<Events>() {  
 @Override  
 public void onResponse(Call<Events> call, Response<Events> response) {  
 Log.*d*("tag", response.message()+"");  
 }  
  
 @Override  
 public void onFailure(Call<Events> call, Throwable throwable) {  
 Log.*d*("tag", throwable.getMessage());  
 }  
 });  
  
 Intent intent = new Intent(this, MainActivity.class);  
 startActivity(intent);  
 }  
 }  
  
 public void DeleteEvent(View view){  
 *sqLiteHelper* = new SQLiteHelper(this, "app.db", null, 1);  
 *sqLiteHelper*.deleteEvent(id);  
 *sqLiteHelper*.deleteMembers(id);  
  
 Call<Void> call = iapi.deleteEvent(id);  
 call.enqueue(new Callback<Void>() {  
 @Override  
 public void onResponse(Call<Void> call, Response<Void> response) {  
 Log.*d*("tag2", response.message()+"");  
 }  
  
 @Override  
 public void onFailure(Call<Void> call, Throwable throwable) {  
 Log.*d*("tag2", throwable.getMessage());  
 }  
 });  
  
 Call<Void> call2 = iapi.deleteMembers(id);  
 call2.enqueue(new Callback<Void>() {  
 @Override  
 public void onResponse(Call<Void> call, Response<Void> response) {  
 Log.*d*("tag2", response.message()+"");  
 }  
  
 @Override  
 public void onFailure(Call<Void> call, Throwable throwable) {  
 Log.*d*("tag2", throwable.getMessage());  
 }  
 });  
  
 Intent intent = new Intent(this, MainActivity.class);  
 startActivity(intent);  
 }  
  
// public void GoToDetailsMemberPage(View view){  
// Intent intent = new Intent(this, DetailsMemberPage.class);  
// intent.putExtra(MainActivity.EXTRA\_EV\_TITLE, u\_name);  
// intent.putExtra(MainActivity.EXTRA\_EV\_DESCR, u\_phone);  
// intent.putExtra(MainActivity.EXTRA\_EV\_DATE, u\_email);  
// startActivity(intent);  
// }  
  
 //test calendar  
 public void datePickerHandler(View view) {  
 showDialog(1);  
 }  
  
 protected Dialog onCreateDialog(int id) {  
 if (id == 1) {  
 DatePickerDialog tpd = new DatePickerDialog(this, myCallBack, currentYear, currentMonth, currentDay);  
 return tpd;  
 }  
 return super.onCreateDialog(id);  
 }  
  
 DatePickerDialog.OnDateSetListener myCallBack = new DatePickerDialog.OnDateSetListener() {  
 public void onDateSet(DatePicker view, int year, int monthOfYear,  
 int dayOfMonth) {  
 currentYear = year;  
 currentMonth = monthOfYear + 1;  
 currentDay = dayOfMonth;  
 date\_edit\_page.setText(currentDay + "." + currentMonth + "." + currentYear);  
 }  
 };  
}