**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ**

**Кафедра Веб-технологий и компьютерного моделирования**

Кречко Екатерина Сергеевна

**РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ С ИСПОЛЬЗОВАНИЕМ ШАБЛОНА MVP**

Дипломная работа

Научный руководитель:

кандидат физ.-мат. наук

Блинов Игорь Николаевич

Допущен к защите

«\_\_\_» \_\_\_\_\_\_\_\_ 2017 г.

Зав. кафедрой Веб-технологий и компьютерного моделирования

кандидат физико-математических наук, доцент В.С. Романчик

Минск, 2017

**РЕФЕРАТ**

Дипломный проект состоит из пояснительной записки общим объемом 34 страниц, содержит 23 рисунка, 1 приложение и 6 использованных источников.

Целью дипломного проекта является сравнительный анализ существующих инструментов разработки мобильных приложений под различные платформы и применение одного из них на реальном проекте.

Первая глава содержит в себе теоретические сведения рассматриваемого вопроса и условно поделена на три раздела.

В первом разделе рассматривается рынок мобильных приложений в общем, анализируется доля основных мобильных платформ на этом рынке.

Второй раздел представляет собой краткий обзор существующих технологий для разработки мобильных приложений, их основные свойства и отличия друг от друга

В третьем разделе представлена информация по шаблонам проектирования мобильных приложений

Вторая глава включает в себя постановку задачи и описывает этапы разработки практической части. В ней рассказывается об архитектуре разработанного приложения, описывается его основной функционал, приводятся примеры работы приложения на телефоне.

Третья глава рассказывает о дальнейших перспективах в развитии написанного приложения.

**РЭФЕРАТ**

Дыпломны праект складаецца з тлумачальнай запіскі агульным аб'ёмам 34 старонкi , утрымлівае 23 малюнка , 1 прыкладанне і 6 выкарыстаных крыніц .

Мэтай дыпломнага праекта з'яўляецца параўнальны аналіз існуючых інструментаў распрацоўкі мабільных прыкладанняў пад розныя платформы і прымяненне аднаго з іх на рэальным праекце.

Першы раздзел ўтрымлівае ў сабе тэарэтычныя звесткі разгляданага пытання і ўмоўна падзелены на тры часткі .

У першай частке разглядаецца рынак мабільных прыкладанняў у агульным , аналізуецца доля асноўных мабільных платформаў на гэтым рынку.

Другая частка уяўляе сабой кароткі агляд існуючых тэхналогій для распрацоўкі мабільных прыкладанняў , іх асноўныя ўласцівасці і адрозненні адзін ад аднаго

У трэцяй частке прадстаўлена больш падрабязная інфармацыя аб шаблонах праэктыравання мабiльных прыкладанняў.

Другi раздзел ўключае ў сябе пастаноўку задачы і апісвае этапы распрацоўкі практычнай часткі. У iм распавядаецца пра архітэктуру распрацаванага прыкладання , апісваецца яго асноўны функцыянал , прыводзяцца прыклады працы прыкладання і на тэлефоне .

Трэцi раздзел распавядае пра далейшыя перспектывы ў развіцці напісанага прыкладання.

**ABSTRACT**

Graduation project contains an explanatory note, consisting of 34 pages, 23 images, 1 appendix and 6 sources.

The purpose of the graduation project is a comparative analysis of existing tools mobile application development for various platforms and using one of them on a real project.

The first chapter contains theoretical information about subject and it conventionally divided into three sections.

The first section examines the market for mobile applications in general, it analyzes part of all most popular mobile platforms on the market.

The second section presents an overview of existing technologies for mobile application development, their basic properties and differences of each other.

The third section provides more detailed information about patterns for mobile applications.

The second chapter includes a statement of the problem and describes the stages of development of the practical part. It describes the architecture of the developed application, describes its basic functionality,has examples of the application on the tablet and the phone .

The third chapter discusses the future prospects of the development of the application.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 6](#_Toc482559463)

[Глава 1. НЕОБХОДИМЫЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ 7](#_Toc482559464)

[1.1 Исследование предметной области 7](#_Toc482559465)

[1.2 Обзор существующих инструментов для достижения цели 7](#_Toc482559466)

[1.3 Основные шаблоны проектирования для мобильных устройств 9](#_Toc482559467)

[1.3.1 Обзор семейства шаблонов Model-View 10](#_Toc482559468)

[1.3.2 Шаблон Model-View-Controller 11](#_Toc482559469)

[1.3.3 Шаблон Presentation Model 12](#_Toc482559470)

[1.3.4 Шаблон Model-View-ViewModel 13](#_Toc482559471)

[1.3.5 Шаблон Model-View-Presenter 14](#_Toc482559472)

[Глава 2. РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ 16](#_Toc482559473)

[2.1 Постановка задачи 16](#_Toc482559474)

[2.2 Разработка архитектуры приложения 16](#_Toc482559475)

[2.3 Хранение данных 17](#_Toc482559476)

[2.4 Разработка базового функционала приложения 18](#_Toc482559477)

[2.4.1 Страница авторизации 18](#_Toc482559478)

[2.4.2 Расписание студента и преподавателя 19](#_Toc482559479)

[2.4.3 Детали занятия 22](#_Toc482559480)

[2.4.4 Список материалов 25](#_Toc482559481)

[2.4.5 Детали материалов 28](#_Toc482559482)

[2.4.6 Изменение деталей занятия 29](#_Toc482559483)

[Глава 3. ВОЗМОЖНЫЕ ПУТИ РАЗВИТИЯ ПРИЛОЖЕНИЯ 31](#_Toc482559484)

[3.1 Дальнейшие перспективы развития приложения 31](#_Toc482559485)

[ЗАКЛЮЧЕНИЕ 32](#_Toc482559486)

[СПИСОК ЛИТЕРАТУРЫ 33](#_Toc482559487)

[ПРИЛОЖЕНИЕ А 34](#_Toc482559488)

# ВВЕДЕНИЕ

В 2016 компания Google провела исследование, которое называлось «Особенности повеления интернет-пользователей». В опросах для данного исследования было опрошено около 60 тысяч человек из 61 страны, среди которых была и Беларусь. По результатам исследования 98% пользователей до 35 лет заходят в интернет ежедневно, 89% молодых пользователей используют для этого смартфоны.

Наиболее распространёнными являются смартфоны с операционными системами iOS и Android. По сведениям из различных источников, доля рынка смартфонов на Android занимает до 90% рынка Беларуси.

Целью данной работы является создание мобильного приложения с пользовательским интерфейсом с использованием шаблона проектирования MVP.

Для достижения поставленной цели необходимо решить ряд задач:

1. рассмотреть основные требования к мобильным приложениям;
2. определить области применения языков программирования для разработки мобильных приложений;
3. проанализировать функциональные возможности сред по созданию мобильных приложений;
4. изучить шаблоны проектирования мобильных приложений.

# НЕОБХОДИМЫЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

## Исследование предметной области

Изначально операционная система Android разрабатывалась компанией Android Inc., но в 2005 году компания Google купила компанию, занимавшуюся разработкой операционной системы. Уже 23 сентября 2008 года официально вышла первая версия операционной системы и полноценный пакет разработчика Software Development Kit 1.0 (SDK 1.0). С тех пор компания Google продолжает непрерывную работу над разработкой системы и регулярно выпускает новый версии продукта.

Android – операционная система, разработанная на базе ядра Linux. Android позволяет создавать Java-приложения, управляющие устройством через разработанные Google библиотеки. Android – открытая операционная система, которая предоставляет возможности интеграции с огромным и постоянно растущим числом сервисов от компании Google. Открытость операционной системы дает возможность настройки смартфона под определенного пользователя и его нужды.

## Обзор существующих инструментов для достижения цели

Android Studio – интегрированная среда разработки (IDE) для работы с платформой Android. Android Studio была разработана компанией Google на основе программного обеспечения Intellij IDEA от компании JetBrains. Первая версия 1.0 была выпущена в 2014 году, хотя среда разработки была доступна с версии 0.1, опубликованной в мае 2013 года. Android Studio является официальным средством разработки Android приложений и, так же как и операционная система, постоянно поддерживается и улучшается компанией Google.

Android Studio предоставляет множество встроенных функций, позволяющих максимизировать производительность разработчика и автоматизировать его рутинную работу, позволяя разрабатывать приложения максимально эффективно.

На рисунке ниже представлен главный экран Android Studio:

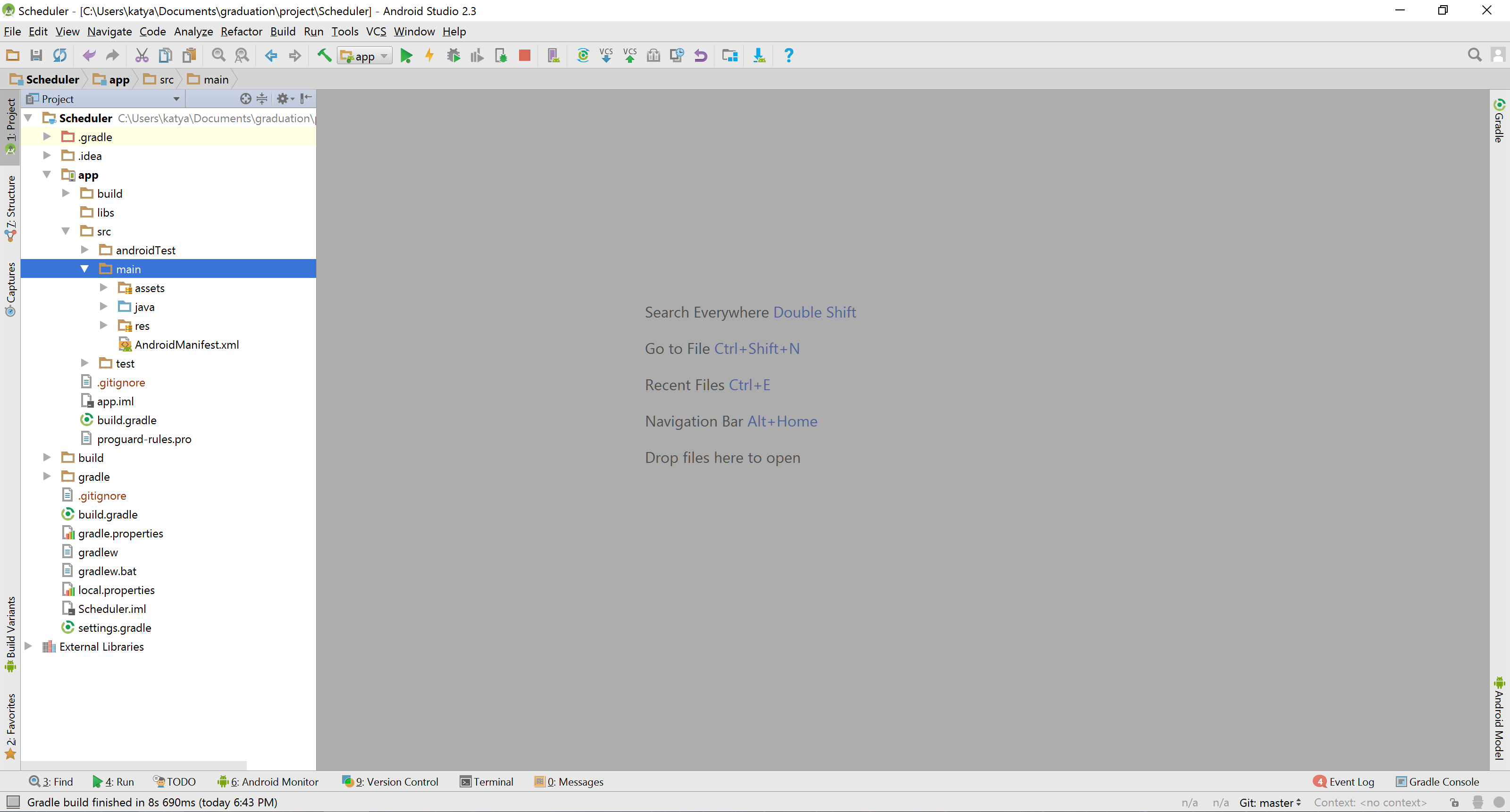


Рисунок ‎1.1 Android Studio

При установке Android Studio устанавливается ряд дополнительных инструментов, таких как Android SDK и Android Virtual Device (AVD). Android Virtual Device – эмулятор реального девайса на Android. В зависимости от назначения разрабатываемого приложения можно создавать различный виртуальные устройства, от обычных смартфонов до телевизоров и умных часов. При создании устройства можно задать собственные характеристики, такие как диагональ экрана, его ширина и высота, объем памяти и версию операционной систем. Этот инструмент позволяет отлаживать и тестировать приложение, поддерживаемое на любом устройстве, вне зависимости от его отдельных характеристик.

AVD позволяет не только запускать приложения, но и полноценно работать с ними. Инструмент эмулирует все основные системы устройства: работа с виртуальной клавиатурой, работу с мобильной сетью и сетями Wi-Fi, работу с GPS навигацией, физическими датчиками и кнопками устройства.



Рисунок ‎1.2 Эмулятор Android

## Основные шаблоны проектирования для мобильных устройств

Шаблон проектирования – некая архитектурная конструкция, представляющая собой решение проблемы, часто возникающей при разработке программного обеспечения.

Рассмотрим очевидный подход к разработке приложения с пользовательским интерфейсом. Разработчик построит форму, добавит на нее интерфейс, добавит туда код для отображения элементов управления, логику обработки событий интерфейса, данные и методы их обработки.

Такой прямолинейный подход хорош, но только для очень простых интерфейсов без сложной логики. При разработке приложений с более сложной логики у такого подхода сразу раскрывается ряд недостатков:

1. Невозможность повторного использования кода. Код данных, написанный в реализации интерфейса, можно использовать только в связке с этим интерфейсом. Логика, реализованная в интерфейсе, может быть использована только с этим интерфейсом.
2. Высокая сложность поддержки кода. Небольшое изменение в пользовательском интерфейсе повлечет за собой неоправданно большие изменения в остальных частях кода. С ростом приложения, числа его функций и элементов интерфейса сложность поддержки только растет.
3. Ухудшение тестируемости. Код не поддается модульному тестированию. В таком случае используется тестирование пользовательского тестирования. Для ручного тестирования – нужен конечный пользователь (тестировщик), для автоматизированного – большие временные затраты на разработку тестов и, возможно, специалиста по автоматизации тестов.

Основная причина появления всех проблем заключается в нарушении одного из принципов объектно-ориентированного программирования – принципа единственной ответственности. В соответствии с этим принципом, любой объект должен иметь лишь одну обязанность; у класса должна быть только одна причина для его изменения. При описанном подходе к разработке класс интерфейса отвечает и за изменения в представлении, и за логику, и за обработку данных.

### Обзор семейства шаблонов Model-View

MV\* шаблоны предназначены для разделения кода обработки данных (Model), кода пользовательского интерфейса (View) и кода логики (Presenter, Controller, ViewModel и др.). Такой подход имеет ряд преимуществ перед описанным прямолинейным подходом. Код становится более поддерживаемым, появляется возможность повторного использования и тестирования отдельных модулей. Такой код не нарушает принципов объектно-ориентированного программирования и, в частности, принципа единственной ответственности.

Основными представителями семейства MV\* шаблонов являются Model-View-Controller, Model-View-ViewModel, Model-View-Presenter. Рассмотрим подробнее общие черты этих шаблонов.

**Model.** Модель представляет собой данные для работы приложения. Модель данных должна быть полностью независима от всех остальных частей продукта. Одна модель может иметь множество различных представлений и контроллеров. Независимость модели дает возможность использовать ее повторно.

**View.** Представление отвечает за отображение данных, полученных от модели. Представление – пользовательский интерфейс, методы, отвечающие за его изменение. Классы представления содержат только логику, отвечающую за работу с интерфейсом пользователя.

### Шаблон Model-View-Controller

Одним из первых шаблонов разделения представления от логики и модели стал шаблон Model-View-Controller. Концепция MVC была описана в конце 1970-х годов Трюгве Реенскаугом, работавшем над языком программирования «Smalltalk». Этот шаблон в корне изменил взгляд на проектирование приложений с пользовательским интерфейсом.

**Controller.** Контроллер представляет собой управляющую логику и отвечает за взаимодействие модели и представления. Контроллер направляет данные от пользователя к системе и обновляет модель или представление. Когда пользователь взаимодействует с приложением, контроллер получает уведомление о действиях пользователя и решает, что с ними делать.

Поскольку шаблон был разработан во времена, когда программирование кардинально отличалось от того, что мы знаем сейчас, нет строгих правил реализации шаблона MVC. Запросы к базам данных, валидация данных, реализация бизнес-логики может размещаться как в модели, так и в контроллере. В зависимости от размещения этих функций, модель может трактоваться по-разному. Различают *пассивную* и *активную* модели.

Пассивная модель не имеет никаких средств воздействия ни на представление, ни на контроллер. В таком случаем модель используется исключительно в качестве источника данных. Все изменения в модели отслеживаются контроллером, который обновляет представление.

Активная модель содержит не только данные, но и оповещает представление об изменениях. Модель при такой реализации содержит бизнес-логику приложения. Представления, в такой реализации, подписываются на изменения в модели. Такая реализация позволяет сохранить независимость модели как от контроллера, так и от представления.

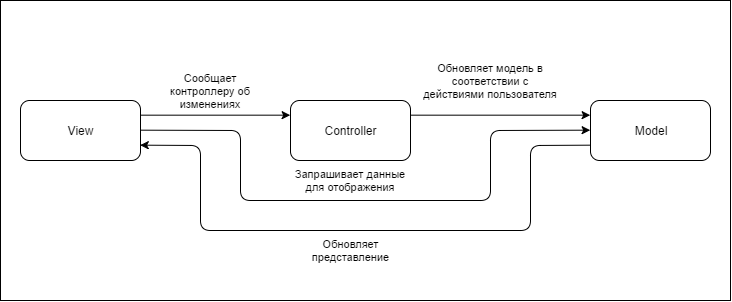


Рисунок ‎1.3 Схема модели MVC

### Шаблон Presentation Model

**PresentationModel.** Модель представления содержит логику пользовательского интерфейса.

Часто информация, содержащаяся в модели не может быть непосредственно использована на представлении. Модель представления может преобразовывать данные из модели, дополнять их или собирать данные из нескольких источников. Это наиболее вероятно, когда у вас нет полного контроля над моделью. Например, если вы получаете данные от сторонних веб-сервисов или же из базы данных существующего приложения.

Модель представления хранит состояние пользовательского интерфейса. В свойствах могут храниться данные, никак не связанные с моделью. Это может быть элемент, выбранный в данный момент; ошибки валидации данных.

Представление может извлекать данные из модели представления и получать всю необходимую информацию для отображения. Одно из преимуществ такого подхода заключается в том, что вы можете создать логическое и полностью тестируемое представление вашего UI, не полагаясь на тестирование визуальных элементов.

### Шаблон Model-View-ViewModel

Шаблон MVVM впервые был представлен Джоном Госсманом в 2005 году как модификация шаблона Presentation Model.

**ViewModel.** ViewModel не может общаться с Представлением напрямую. Вместо этого она представляет легко связываемые свойства и методы в виде команд. View может привязываться к этим свойствам, чтобы получать информацию из ViewModel и вызывать на ней команды (методы). Это не требует того, чтобы View знала о ViewModel. Таким образом, вы можете использовать любую ViewModel для View, которая предоставляет нужные свойства.

Поскольку ViewModel предоставляет View всю необходимую информацию в удобном виде, то само представление может быть довольно простым.

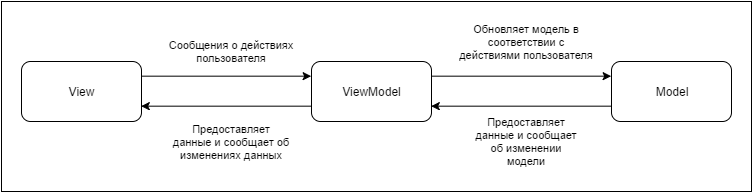


Рисунок ‎1.4 Схема модели MVVM

### Шаблон Model-View-Presenter

**Presenter.** Презентер содержит всю логику пользовательского интерфейса и отвечает за синхронизацию модели и представления. Когда представление уведомляет презентер, что пользователь что-то сделал (например, нажал кнопку), презентер принимает решение об обновлении модели и синхронизирует все изменения между моделью и представлением.

Существует два основных подхода к реализации шаблона MVP: Passive View и Supervising Controller.

**Passive View.** В этом варианте Представление ничего не знает о модели, но вместо этого предоставляет простые свойства для всей информации, которую необходимо отобразить на экране. Презентер будет считывать информацию из модели и обновлять свойства во View. В данной реализации получается довольно много кода как в Представлении, так и в Презентере. Тем не менее, это сделает взаимодействие между ними более тестируемым.

**Supervising Controller.** В этом варианте MVP представление знает о модели и отвечает за связывание данных с отображением. Это делает общение между Презентером и Представлением более лаконичным, но в ущерб тестируемости взаимодействия Представление-Презентер При такой реализации Презентер ередает больше ответственности на Представление.

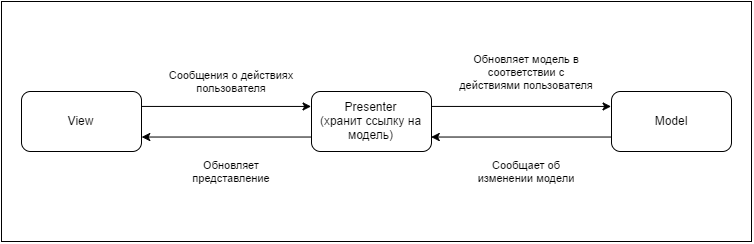


Рисунок ‎1.5 Схема модели MVP в общем виде

# РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ

## Постановка задачи

В общих чертах задача – разработать приложение, простое и удобное в использовании, предназначенное для студентов и преподавателей. Приложение позволит пользователям быстро просматривать их расписание, информацию по курсам, информацию по преподавателям, дополнительные материалы по предметам.

Приложение поддерживает 2 роли: студента и преподавателя. От роли пользователя зависит список отображаемых занятий и доступность возможности редактировать их. Студент может только просматривать свое расписание и материалы к курсам, в то же время преподаватель имеет возможность не только просматривать расписание и материалы, но также может редактировать их.

## Разработка архитектуры приложения

Три основных класса модели, которые используются в приложении – это Lesson, Material и User, представляющие соответственно занятие, дополнительный материал и пользователя. Основные поля этих классов показаны ниже (Рисунок ‎2.1).

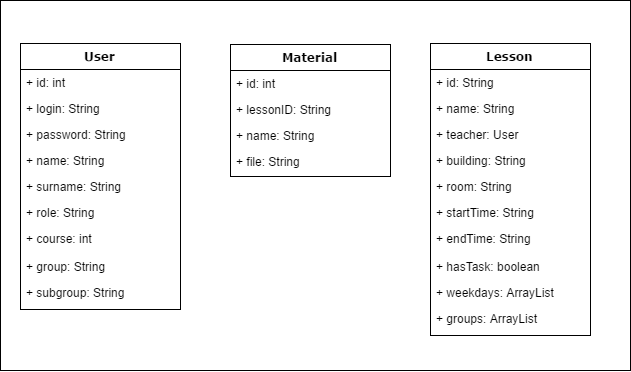


Рисунок ‎2.1 Поля классов User, Material, Lesson

## Хранение данных

Для хранения данных приложения используется база данных SQLite. SQLite— компактная встраиваемая реляционная база данных. Исходный код библиотеки передан в общественное достояние. В 2005 году проект получил награду Google-O’Reilly Open Source Awards.

Библиотека Android содержит абстрактный класс SQLiteOpenHelper, с помощью которого можно создавать, открывать и обновлять базы данных. Это основной класс, с которым приходится работать в своих проектах. При реализации этого вспомогательного класса от разработчика скрывается логика, на основе которой принимается решение о создании или обновлении базы данных перед ее открытием. Класс SQLiteOpenHelper содержит два обязательных абстрактных метода:

• onCreate() — вызывается при первом создании базы данныхЖ

@Override  
**public void** onCreate(SQLiteDatabase db) {  
 db.execSQL(***SQL\_CREATE\_LESSONS\_TABLE***);  
 db.execSQL(***SQL\_CREATE\_MATERIALS\_TABLE***);  
}

• onUpgrade() — вызывается при модификации базы данных:

@Override  
**public void** onUpgrade(SQLiteDatabase db, **int** oldVersion, **int** newVersion) {  
 db.execSQL(***SQL\_DELETE\_LESSONS\_TABLE***);  
 db.execSQL(***SQL\_DELETE\_MATERIALS\_TABLE***);  
 onCreate(db);  
}

Для управления базой данных SQLite существует класс SQLiteDatabase. В классе SQLiteDatabase определены методы query(), insert(), delete() и update() для чтения, добавления, удаления, изменения данных. Кроме того, метод execSQL() позволяет выполнять любой допустимый код на языке SQL применимо к таблицам базы данных, если вы хотите провести эти (или любые другие) операции вручную.

## Разработка базового функционала приложения

### Страница авторизации

После открытия приложения первое окно, которое видит пользователь – окно авторизации. Здесь пользователь вводит свои логин и пароль. При обработке этих данных определяется роль пользователя.

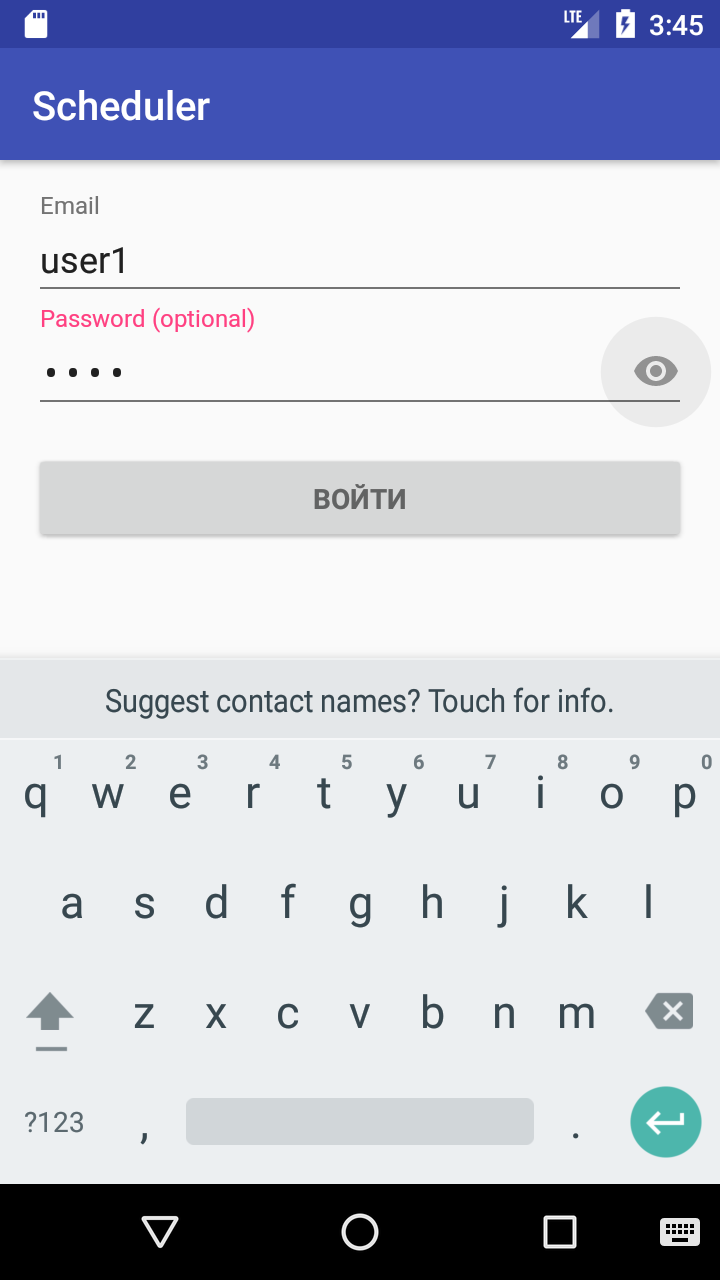


Рисунок ‎2.2 Страница авторизации пользователя

### Расписание студента и преподавателя

После успешной авторизации пользователь увидит страницу с тремя вкладками, первая из которых будет активна. На первой вкладке отображается расписание. Для студента отображаются те занятия, на которых должна присутствовать его группа.

Все занятия отображаются списком, отсортированным по времени начала того или иного занятия. Для того, чтобы реализовать сортировку объектов пользовательского типа был имплементирован класс Comparator, отвечающий за сравнение объектов. Также был переопределен метод compareTo в классе Lesson (метод интерфейса Comparable, который был имплементирован для класса Lesson).

**public class** LessonComparator **implements** Comparator<Lesson>{  
 @Override  
 **public int** compare(Lesson o1, Lesson o2) {  
 **return** o1.compareTo(o2);  
 }  
}

@Override  
**public int** compareTo(@NonNull Object o) {  
  
 **try**{  
 Lesson object = (Lesson) o;  
 SimpleDateFormat dateFormat = **new** SimpleDateFormat(**"hh.mm"**);  
 Date currentStartTime = dateFormat.parse(**this**.**startTime**);  
 Date objectStartTime = dateFormat.parse(object.getStartTime());  
  
 **if**(currentStartTime.equals(objectStartTime)){  
 **return** 0;  
 } **else if**(currentStartTime.after(objectStartTime)){  
 **return** 1;  
 } **else if** (currentStartTime.before(objectStartTime)){  
 **return** -1;  
 }  
  
 } **catch** (Exception e){  
 e.printStackTrace();  
 }  
 **return** 0;  
}

Для каждого занятия отображается краткая информация по нему: время начало и окончания, название предмета и место проведения. Для отображения занятий в виде списка использовался элемент ListView, а для заполнения списка данными использовался стандартный класс андроид BaseAdapter, методы которого были переопределены для работы с объектами типа Lesson:

**public class** ScheduleAdapter **extends** BaseAdapter {  
  
 Context **context**;  
 LayoutInflater **layoutInflater**;  
 ArrayList <Lesson> **lessons**;  
  
 **public** ScheduleAdapter(Context context, ArrayList <Lesson> less) {  
 **this**.**context** = context;  
 **lessons** = less;  
 **layoutInflater** = (LayoutInflater) context  
 .getSystemService(Context.***LAYOUT\_INFLATER\_SERVICE***);  
 }  
  
 @Override  
 **public int** getCount() {  
 **return lessons**.size();  
 }  
  
 @Override  
 **public** Object getItem(**int** position) {  
 **return lessons**.get(position);  
 }  
  
 @Override  
 **public long** getItemId(**int** position) {  
 **return** position;  
 }  
  
 @Override  
 **public** View getView(**int** position, View convertView, ViewGroup parent) {  
 View view = convertView;  
 **if** (view == **null**) {  
 view = **layoutInflater**.inflate(R.layout.***item\_list\_lesson***, parent, **false**);  
 }  
 **return** view;  
 }  
  
 Lesson getLesson(**int** position) {  
 **return** ((Lesson) getItem(position));  
 }  
}

Вид получившегося списка показан ниже (Рисунок ‎2.3).

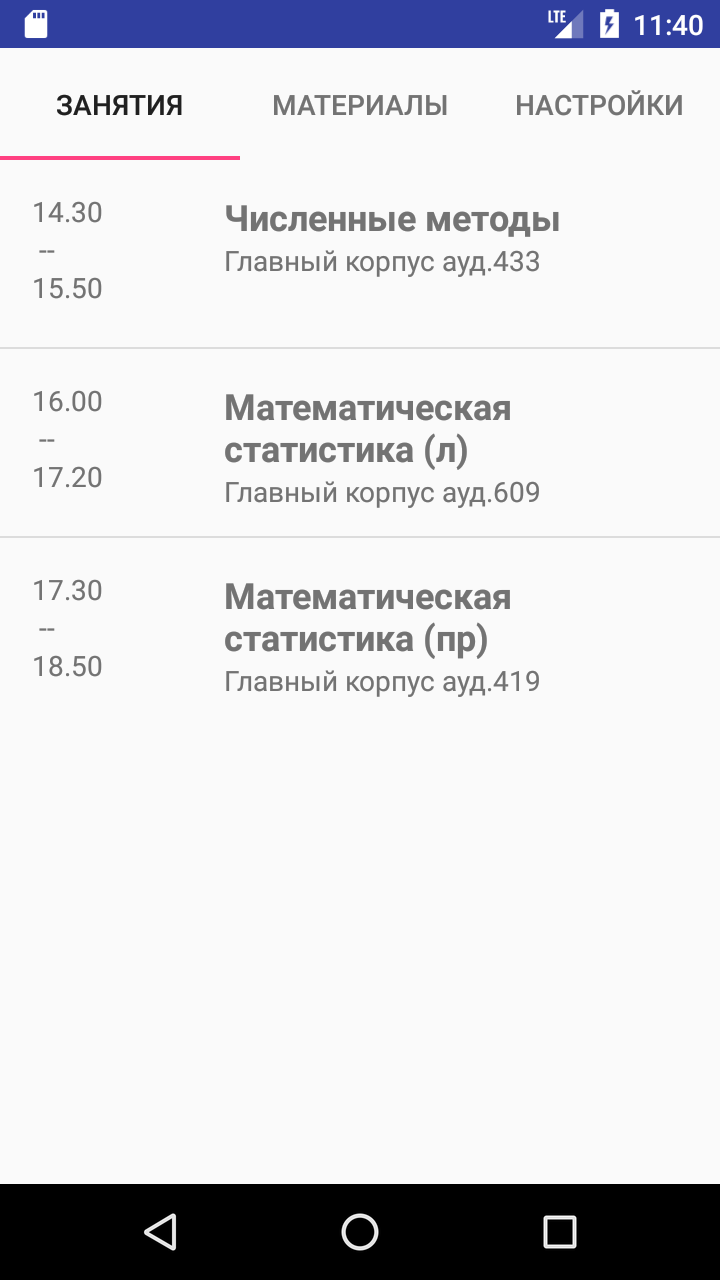


Рисунок ‎2.3 Расписание пользователя

Каждый элемент списка представляет собой объект класса Lesson. При нажатии на элемент списка пользователь будет направлен на окно с полной информацией о занятии (Рисунок ‎2.4)

### Детали занятия

Данный экран содержит полную информацию по занятию, которое представлено объектом класса Lesson:

1. Название предмета;
2. Фамилию, имя и отчество преподавателя;
3. Место проведения занятия;
4. Время начала и окончания;
5. Список групп, которые должны присутствовать на занятии;
6. Кнопку «Задание», открывающую домашнее задание (Рисунок ‎2.5);
7. Кнопку «Редактировать», открывающую окно редактирования информации о занятии (Рисунок ‎2.7).

Кнопки отображаются только при выполнении определенных условий. Кнопка «Задание» появляется для всех типов пользователей, но только если у выбранного занятия есть дополнительные материалы (то есть поле hasTask объекта класса Lesson имеет значение true). Кнопка «Редактировать» появится только для пользователя в роли преподавателя.

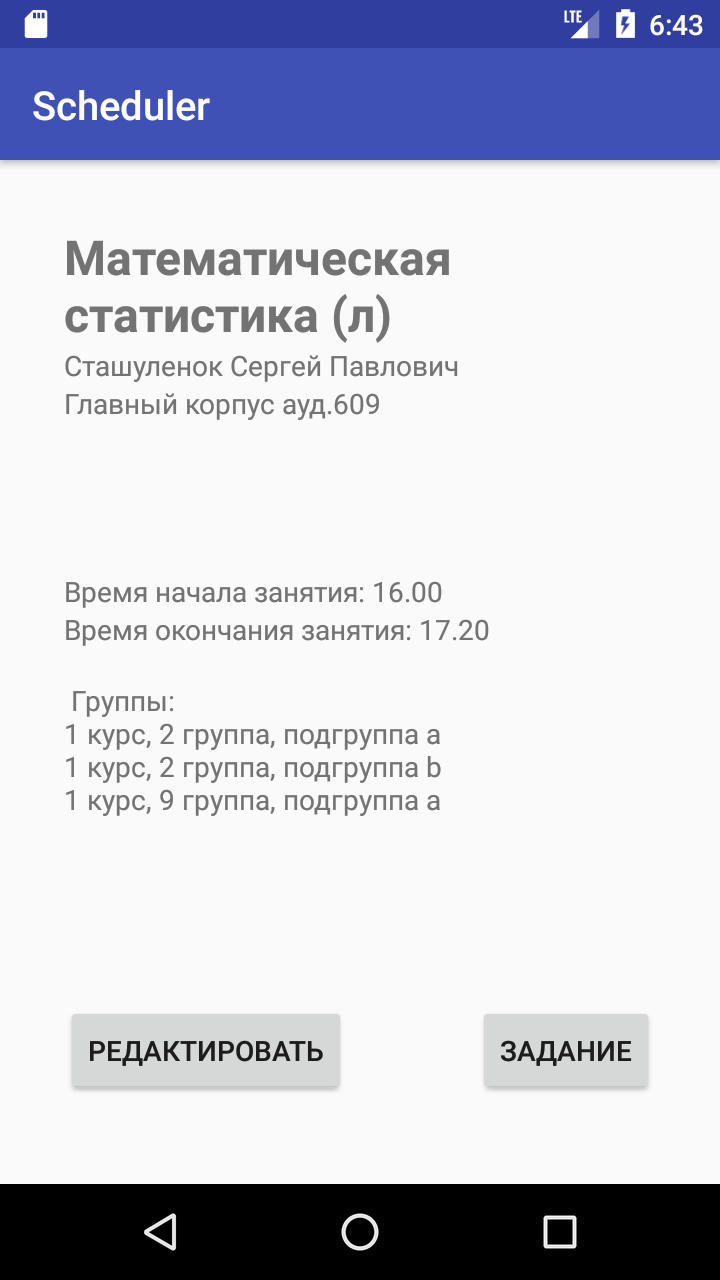


Рисунок ‎2.4 Детали занятия

Для того, чтобы открыть новое окно используется класс Intent, с помощью которого на новое активити передается объект Lesson:

**public void** openLessonDetailsActivity(Lesson lesson){  
 Intent intent = **new** Intent(**this**, LessonDetailsActivity.**class**);  
 intent.putExtra(**"Lesson"**, lesson);  
 startActivity(intent);  
}

Переданный объект считывается с использованием того же класса Intent. Для того чтобы таким образом передавать объекты, для класса Lesson был имплементирован интерфейс Serializable.

lesson = (Lesson) getIntent().getSerializableExtra(**"Lesson"**);

### Список материалов

На второй вкладке главного экрана отображается список материалов, сгруппированных по предметам.

Для отображения материалов использовался элемент ExpandableListView:

<**ExpandableListView  
 android:id="@+id/listMaterials"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"**/>

По аналогии с простым ListView для заполнения списка данными использовался адаптер, переопределяющий методы встроенного BaseExpandableListAdapter:

**public class** MaterialsAdapter **extends** BaseExpandableListAdapter {  
  
 **private** Context **context**;  
 **private** List<Lesson> **lessonNames**;  
 HashMap<Lesson, List<Material>> **materialsLesson**;  
  
 **public** MaterialsAdapter(Context context, List<Lesson> lessons, HashMap<Lesson, List<Material>> materialsLesson) {  
 **this**.**context** = context;  
 **this**.**lessonNames** = lessons;  
 **this**.**materialsLesson** = materialsLesson;  
 }  
  
 @Override  
 **public int** getGroupCount() {  
 **return this**.**lessonNames**.size();  
 }  
  
 @Override  
 **public int** getChildrenCount(**int** groupPosition) {  
 **return this**.**materialsLesson**.get(**this**.**lessonNames**.get(groupPosition)).size();  
 }  
  
 @Override  
 **public** Object getGroup(**int** groupPosition) {  
 **return this**.**lessonNames**.get(groupPosition);  
 }  
  
 @Override  
 **public** Object getChild(**int** groupPosition, **int** childPosition) {  
 **return this**.**materialsLesson**.get(**this**.**lessonNames**.get(groupPosition))  
 .get(childPosition);  
 }  
  
 @Override  
 **public long** getGroupId(**int** groupPosition) {  
 **return** groupPosition;  
 }  
  
 @Override  
 **public long** getChildId(**int** groupPosition, **int** childPosition) {  
 **return** childPosition;  
 }  
  
 @Override  
 **public boolean** hasStableIds() {  
 **return false**;  
 }  
  
 @Override  
 **public** View getGroupView(**int** groupPosition, **boolean** isExpanded, View convertView, ViewGroup parent) {  
 Lesson lesson = (Lesson) getGroup(groupPosition);  
 **if** (convertView == **null**) {  
 LayoutInflater infalInflater = (LayoutInflater) **this**.**context**.getSystemService(Context.***LAYOUT\_INFLATER\_SERVICE***);  
 convertView = infalInflater.inflate(R.layout.***list\_group***, **null**);  
 }  
  
 TextView lblListHeader = (TextView) convertView.findViewById(R.id.***lblListHeader***);  
 lblListHeader.setTypeface(**null**, Typeface.***BOLD***);  
 lblListHeader.setText(lesson.getName());  
  
 **return** convertView;  
 }  
  
 @Override  
 **public** View getChildView(**int** groupPosition, **int** childPosition, **boolean** isLastChild, View convertView, ViewGroup parent) {  
 **final** Material material = (Material) getChild(groupPosition, childPosition);  
  
 **if** (convertView == **null**) {  
 LayoutInflater infalInflater = (LayoutInflater) **this**.**context**.getSystemService(Context.***LAYOUT\_INFLATER\_SERVICE***);  
 convertView = infalInflater.inflate(R.layout.***list\_item***, **null**);  
 }  
  
 TextView txtListChild = (TextView) convertView.findViewById(R.id.***lblListItem***);  
  
 txtListChild.setText(material.getName());  
 **return** convertView;  
 }  
  
 @Override  
 **public boolean** isChildSelectable(**int** groupPosition, **int** childPosition) {  
 **return true**;  
 }  
}

При нажатии на название предмета открывается группа, содержащая список доступных материалов. При нажатии на один из материалов, открывается окно, отображающее содержимое материала (Рисунок ‎2.5).

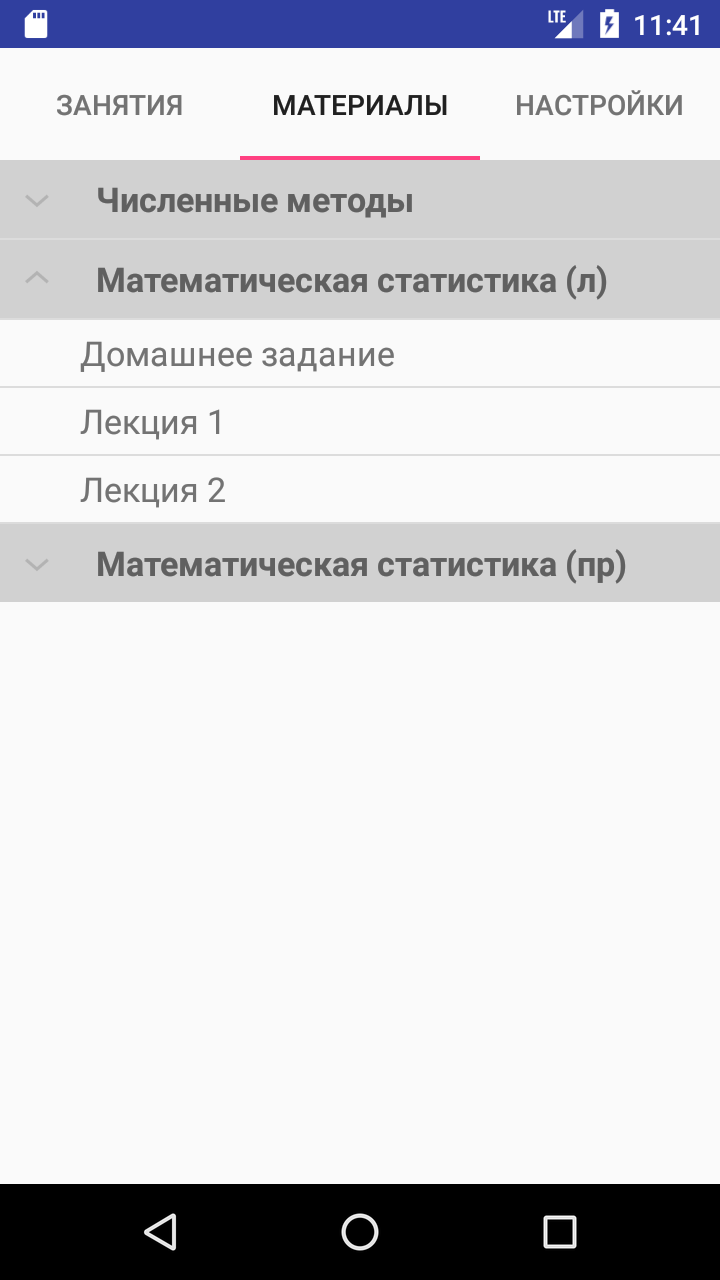


Рисунок ‎2.5 Список доступных материалов

### Детали материалов

На данном окне отображается название конкретного материала и показывается содержимое текстового файла, связанного с этим материалом.

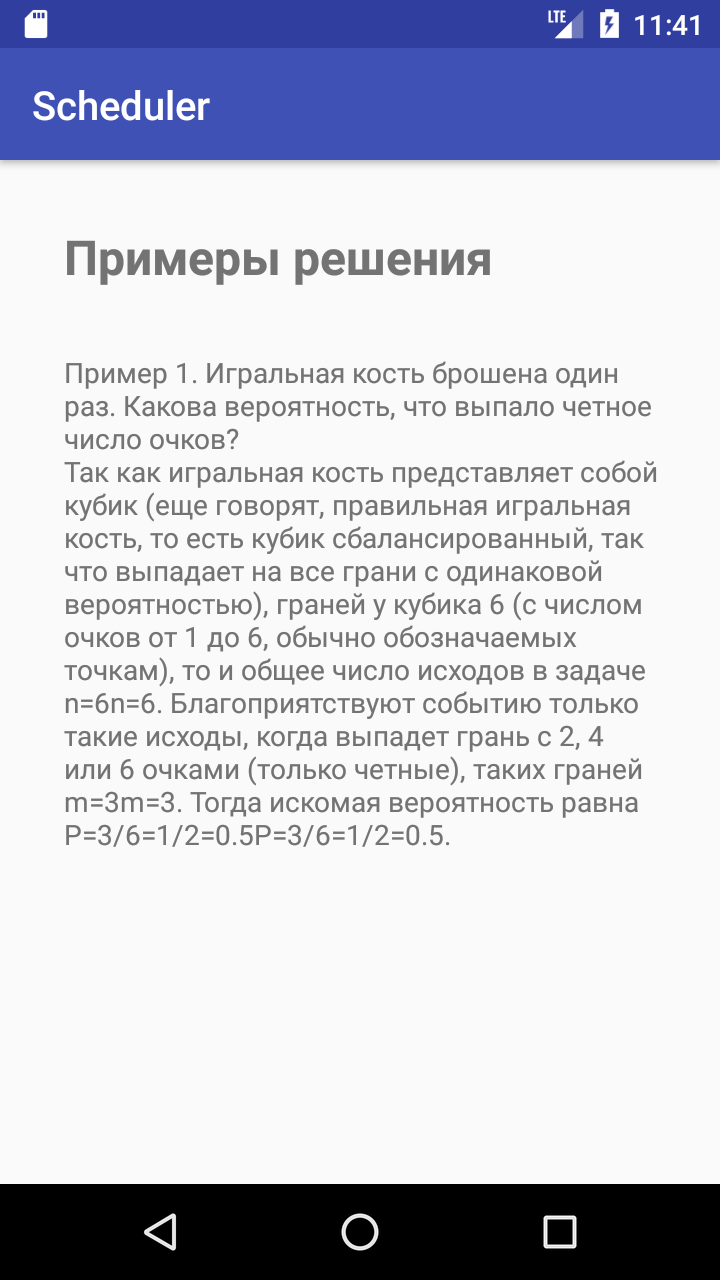


Рисунок ‎2.6 Содержимое материалов

### Изменение деталей занятия

На это окно может попасть только пользователь, авторизовавшийся как «преподаватель». Изменить можно место проведения занятия и время его начала и окончания. После нажатия кнопки «Сохранить изменения» все изменения, внесенные пользователем заносятся в базу данных.

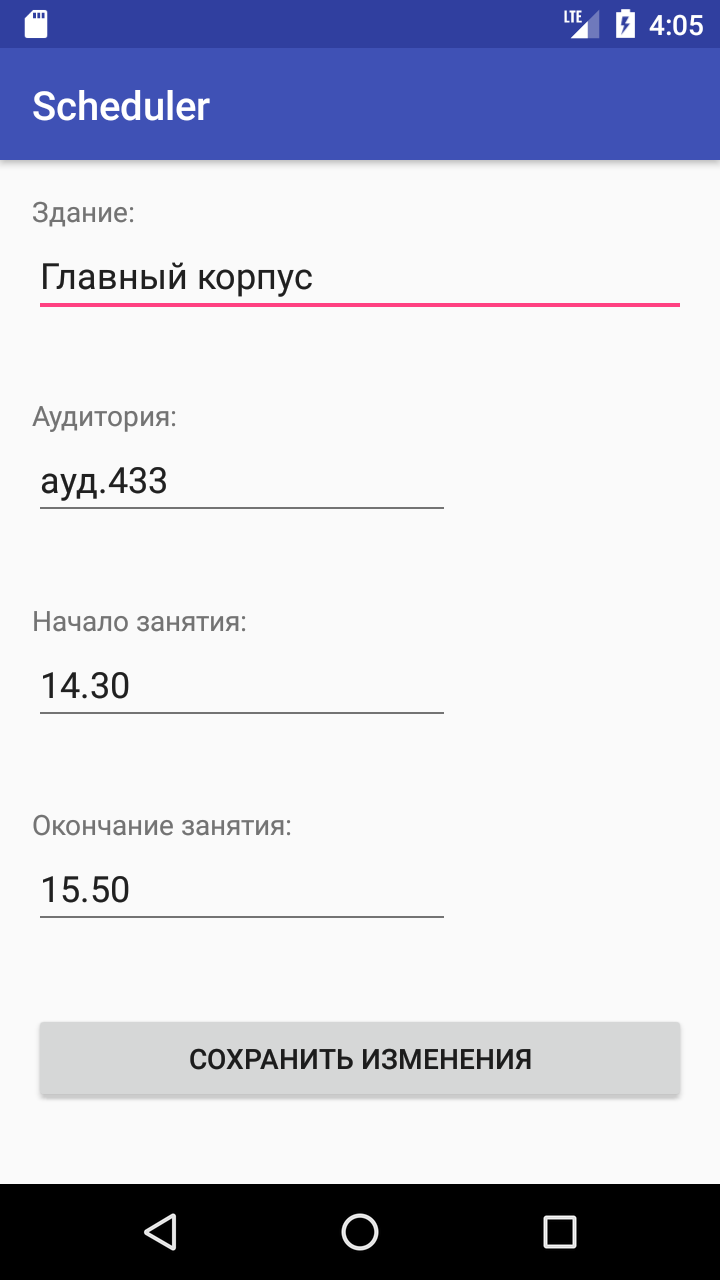


Рисунок ‎2.7 Изменение деталей занятия

# ВОЗМОЖНЫЕ ПУТИ РАЗВИТИЯ ПРИЛОЖЕНИЯ

## Дальнейшие перспективы развития приложения

Целевая аудитория данного приложения – студенты и преподаватели механико-математического факультета, поэтому в ближайшей перспективе – внедрение приложения на факультете, в университете, привязка к общеуниверситетской базе данных пользователей и расширение функционала, поддержка различных форматов файлов.

Также в дальнейшей перспективе – поддержка других операционных систем, таких как iOS и Windows Mobile.

# ЗАКЛЮЧЕНИЕ

В ходе данной дипломной работы были изучены мобильные платформы и различные способы написания приложений на них. Проведен сравнительный анализ существующего инструментария для разработки, по результатам которого выделен шаблон проектирования, позволяющий наилучшим образом написать приложение, с пользовательским интерфейсом. Так же такой подход позволяет разделить реализацию приложения на отдельные компоненты и значительно упростить расширяемость функционала.

На основании полученных знаний и по результатам исследований различных путей написания мобильных приложений, было написано приложение, полностью обладающее всем необходимым функционалом.

Исходя из прогнозов развития мобильных технологий в обозримом будущем, можно сделать вывод, что написанное приложение и дальше будет развиваться и прогрессировать.

# СПИСОК ЛИТЕРАТУРЫ

1. <https://dev.by/lenta/main/google-91-belorusov-ezhednevno-prisutstvuyut-onlayn>
2. <https://habrahabr.ru/post/215605/>
3. <https://code.tutsplus.com/tutorials/how-to-adopt-model-view-presenter-on-android--cms-26206>
4. Блинов, И.Н., Романчик, В. С. Java. Методы программирования: уч.-мет. пособие / И.Н. Блинов, В.С. Романчик. - Минск: издательство «Четыре четверти», 2013. - 896 с.
5. Голощапов, А.Л. Google Android. Программирование для мобильных устройств.
6. Варакин, М.В. Разработка мобильных приложений под Android: учебное пособие / М.В. Варакин; Центр компьютерного обучение "Специалист" при МГТУ им. Н.Э. Баумана – Москва, 2012 – 128с.

# ПРИЛОЖЕНИЕ А

Исходный код данного проекта находиться на диске прилагаемом к данной документации.