МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра Математической кибернетики и компьютерных наук

РАЗРАБОТКА МОБИЛЬНОГО ФИТНЕС-ПРИЛОЖЕНИЯ ДЛЯ ОПЕРАЦИОННОЙ СИСТЕМЫ ANDROID

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 411 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Авраменко Алексея Владиславовича

Научный руководитель
доцент, к. ф.-м. н. _____ А. С. Иванов

Заведующий кафедрой
к. ф.-м. н. _____ А. С. Иванов

Саратов 2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время мобильные гаджеты стали очень тесно связаны с повседневной жизнью людей. Сложно представить современного человека, не имеющего смартфон с доступом в интернет. С каждым днём всё больше и больше людей становятся пользователями различных приложений и сервисов. Самой распространённой мобильной операционной системой является Android. На момент написания данной работы, количество активных устройств во всём мире, работающих на данной операционной системе, составляет более 2,5 миллиардов.

Целью настоящей работы является создание мобильного Android приложения в помощь организации занятия спортом, с возможностью хранения личных данных пользователя и синхронизацией с облачной базой данных. Для данной цели были определены следующие задачи:

- Рассмотрение аналогов фитнес-приложений для операционной системы Android,
- Изучение структуры Android приложений и их архитектурных компонентов,
- Разработка графического пользовательского интерфейса,
- Изучение работы облачной базы данных и её использование,
- Изучение работы встраиваемой локальной базы данных,
- Реализация мобильного приложения для операционной системы Android.

Работа состоит из введения, трёх глав, заключения, списка из 20 источников и трёх приложений. В первой главе под названием «Рассмотрение аналогов приложения» рассматриваются аналоги мобильных фитнес-приложений для анализа логики работы интерфейсов, на основе которого, будет разрабатываться бизнес логика приложения из практической части работы. Во второй главе «Компоненты Android приложений» описаны архитектурные компоненты Android приложений, при помощи которых была реализована структура работы приложения, а также визуальные компоненты, которые представляют собой элементы пользовательского интерфейса. В третьей главе «Реализация приложения» описаны инструменты разработки практической части, а также логика работы реализованного приложения.

1 Основное содержание работы

1.1 Рассмотрение аналогов приложения

В работе было рассмотрено три аналога фитнес-приложений:

— *Приложение «Библиотека упражнений»*. Данное приложение представляет из себя справочник упражнений и программ тренировок. У пользователя есть возможность использовать данное приложение как справочник упражнений, а также руководствоваться составленными программами тренировок.

— *Приложение «Фитнес тренер»*. Данное приложение представляет из себя справочник программ тренировок и даёт возможность пользователю самостоятельно добавлять тренировки и хранить их в памяти устройства.

— *Приложение «GymLog — онлайн дневник тренировок»* представляет собой пользовательский сервис для помощи занятий спортом в тренжёрном зале. Пользователь приложения имеет возможность изучать упражнения, просматривать и использовать программы тренировок, а также, авторизовавшись, составлять свои собственные тренировки и хранить их.

1.2 Компоненты Android приложений

В данной главе описаны основные архитектурные компоненты Android приложений.

Activity (Активность):

Основной компонент Android приложений. Именно в нём происходят действия, происходящие на экране. Они представляют из себя окна, заполняющие собой всё пространство экрана, что видит пользователь. В определённый момент времени, на экране отображается одно *Activity*, а, в процессе работы, приложение переключается между ними.

Жизненный цикл приложения:

В силу того, что мобильное приложение представляет собой структуру действий, которые имеют свойство влиять на другие действия, с течением времени, имеет место такое понятие как жизненный цикл. Так как компонент *Activity* является полным отражением работы приложения в момент времени, то имеет место говорить о жизненном цикле компонента *Activity*. В жизненном цикле *Activity* 6 основных методов переходов состояний:

— Метод `onCreate()` срабатывает, когда система впервые создаёт *Activity*.

Далее, после создания *Activity*, оно перезодит в состояние *«created»*, а после, в состояние *«started»*. В этом методе вы выполняете базовую логику запуска приложения, которая должна происходить только один раз за весь период действия.

— Метод `onStart()` срабатывает, когда перехожит в состояние *«started»*. Этот метод делает *Activity* видимым для пользователя, так как готовиться к деятельности, перед тем как стать интерактивным. После завершения этого вызова, действие переходит в состояние *«resumed»*.
— Метод `onResume()` срабатывает после того, как Activity переходит в состояние *«resumed»*. Это состояние, в котором пользователь взаимодействует с приложением. Приложение остаётся в этом состоянии пока не произойдёт так называемое, прирывающее событие.
— Метод `onPause()` вызывается, когда Activity переходит в состояние *«paused»*. При выполнении этого метода, приложение находится в состоянии паузы, и будет находиться в нём, пока снова не перейдёт в состояние *«resumed»*.
— Метод `onStop()` вызывается, когда Activity переходит в состояние *«stopped»*. Когда Activity переходит в остановленное состояние, любой компонент, связанный с жизненным циклом Activity, получит «ON_STOP» событие.
— Метод `onDstroy()` вызывается до уничтожения Activity.

Fragment:

Фрагменты впервые появились в Android версии 3.0. Их основной целью является обеспечение большей динамичностью и гибкостью пользовательских интерфейсов на больших экранах, например, у планшетов. Поскольку экраны планшетов гораздо больше, чем у смартфонов, они предоставляют больше возможностей для объединения и перестановки компонентов пользовательского интерфейса. Фрагменты избавляют разработчика от необходимости управлять сложными изменениями в иерархии представлений. Фрагмент всегда должен быть встроен в Activity, и на его жизненный цикл напрямую влияет жизненный цикл Activity. В настоящий момент частой практикой является создание единого Activity, с множеством фрагментов внутри него.

LiveData:

Важнейшим архитектурным компонентом Android приложения является *LiveData*. *LiveData* — это класс хранения данных, работающий по принципу паттерна проектирования *Observer* (наблюдатель). Принципиальным отличием

является то, что *LiveData* учитывает жизненный цикл других компонентов приложения, таких как Activity, фрагменты или службы. Это гарантирует то, что *LiveData* будет обновлять только те компоненты приложения, которые находятся в состоянии активного жизненного цикла.

View Model:

ViewModel — это класс, отвечающий за логику работы приложения, за связь между данными и пользовательским интерфейсом. Компоненты архитектуры предоставляют *ViewModel* вспомогательный класс для контроллера пользовательского интерфейса, который отвечает за подготовку данных для пользовательского интерфейса. *ViewModel* объекты автоматически сохраняются во время изменений конфигурации, так что содержащиеся в них данные сразу же становятся доступны для следующего Activity или экземпляра фрагмента. Например, если нужно отобразить список пользователей в приложении, то необходимо возложить ответственность за получение и сохранение списка пользователей классу *ViewModel* вместо Activity или фрагмента.

Таким образом, с помощью компонента *ViewModel*, можно отделить логику работы приложения от логики обработки данных, а также использовать его в качестве посредника для обмена данными между фрагментами или Activity.

Элемент ConstraintLayout:

С появлением версии «Android Studio» 2.2, разработка пользовательского интерфейса значительно упростилась. Дело в том, что с появлением этого элемента, появилось больше возможностей позиционировать элементы *View*, находящиеся внутри *ConstraintLayout*, с помощью, так называемых, привязок. У любого элемента с каждой из четырёх сторон есть возможность «привязаться» к окружающим его элементам. При привязывании элементов устанавливается фиксированная дистанция до тех элементов, к которым привязан данный элемент. Также, при помощи *ConstraintLayout* можно задать область, которую будет занимать элемент, в процентном соотношении от экрана. Таким образом, вёрстка Activity и фрагментов приложения стала гибче.

Элемент BottomNavigationView:

Элемент *BottomNavigationView* представляет собой стандартную нижнюю панель навигации для приложения. Нижние панели навигации позволяют пользователям легко просматривать и переключаться между представлениями

верхнего уровня одним нажатием. Их следует использовать, когда приложение имеет от трех до пяти пунктов назначения верхнего уровня.

Элемент TabLayout:

Элемент *TabLayout* предоставляет горизонтальный макет для отображения вкладок. Макет обрабатывает взаимодействия для группы вкладок, включая:

— прокрутка поведения
— жесты
— выбор вкладки
— анимации
— выравнивание

Элемент ListView:

Элемент *ListView* отображает коллекцию представлений с вертикальной прокруткой, где каждое представление располагается сразу под предыдущим представлением в списке.

Элемент Spinner:

Элемент *Spinner* предоставляет быстрый способ выбрать одно значение из набора. В состоянии по умолчанию счетчик показывает свое текущее выбранное значение. Если дотронуться до счетчика, отобразится раскрывающееся меню со всеми другими доступными значениями, из которых пользователь может выбрать новое.

PickerDialog:

PickerDialog предоставляет виджет для выбора даты.

Adapter:

Adapter — это класс, который связывает наборы данных с *View* элементами. То есть, *Adapter* отвечает за передачу данных из бизнеслогики приложения в вид, то есть во *View*. К примеру, у нас есть элемент *ListView*, элементами которого являются *TextView*. Для того, чтобы ввести строковые данные в *ListView* мы должны определить массив строк, создать объект типа `ArrayAdapter`, передать массив строк объекту адаптера, а объект адаптера записать в поле объекта *ListView*. В методе `getView` будет происходить отрисовка элементов `ListView` с переданными строками коллекции.

1.3 Реализация практической части

Было реализовано мобильное приложение для операционной системы Android. Оно представляет из себя 3 вкладки нижнего меню, где крайняя левая вкладка является справочником упражнений, которые разделены на 3 категории по частям тела: "руки "туловище"и "ноги". Информация об упражнениях хранится в облачной базе и загружается из интернета. Средняя вкладка является личным журналом тренировок пользователя. Пользователь имеет возможность записывать свои собственные тренировки, которые сохраняются в облачной базе данных. Тренировка представляет из себя набор записей по упражнениям, то есть запись представляет из себя упражнение и запись информации, например о количестве подходов, повторений и используемом весе. Журнал тренировок доступен только зарегистрированным и авторизованным пользователям. Если пользователь не авторизован, то средняя вкладка выводит сообщение об этом. Третья вкладка отвечает за аутентификацию, то есть за авторизацию и регистрацию. Также, к приложению подключена локальная база данных. При запуске приложения, данные об упражнениях загружаются в локальную базу данных.

Средствами разработки приложения являются:

- Среда разработки «Android Studio»;
- Облачная СУБД от компании «Firebace» класса NoSQL;
- Встраиваемая кроссплатформенная СУБД *«SQLite»*.

Структура коллекций облачной СУБД:

Для хранения и синхронизации данных используется NoSQL база данных, предоставляемая компанией *«Firebase»*. Данные в этом сервисе хранятся в виде коллекций и документов, имеющих структуру JSON файлов. Были организованы две коллекции, это коллекция *«User»*, содержащая в себе документы, имеющие строковые ключи, и содержащие поля: *«email»*, *«name»*, *«surname»*, *«workouts»*. Поле *«workouts»* представляет собой `ArrayList`, в котором определены поля *«date»*, *«id»* и *«notes»*, который также представляет собой `ArrayList`. В полях элементов «notes» определены поля: *«d»*, *«id_exercise»*, *«id_workouts»*, *«record»*.

Подключение SQLite и создание локальной базы данных:

Для локального хранения данных приложения, была использована встраиваемая СУБД SQLite. Для создания таблиц, был реализован класс `DBHelper`,

наследованный от абстрактного класса для работы с СУБД SQLiteOpenHelper. В данном классе происходит создание самой базы данных и, содержащихся в ней, таблиц.

Классы сущностей:

Для работы с данными приложения, были реализованы классы сущностей для работы с базой данных:

- User, с полями email: String, name: String, surname: String, journal: List<Workout>
- Workout, с полями id: String, date: GregorianCalendar, idUser: String , notes: List<Note>
- Note, с полями id: String, record: String, idWorkout: String, idExersice : String
- Exercise, с полями id: String, name: String, partOfBody: String, description: String

Работа фрагмента авторизации:

Начало приложения начинается с выполнения кода класса MainActivity . После инициализации главного экрана, начинается выполнение кода класса фрагмента авторизации.

Разработка приложения была начата с вёрстки ConstrLayout фрагментов аутентификации. Начальным фрагментом во вкладке аутентификации является фрагмент авторизации в приложении. Был создан класс AuthorizationFragment отражающий жизненный цикл фрагмента авторизации LiveCycle.

В данном файле, в методе onCreateView инициализируем ViewModel данного фрагмента. Далее «раздуваем» и возвращаем view в MainActivity. Далее, срабатывает метод onViewCreated. В данном методе происходит инициализация всех элементов View. Это элементы ввода текста PlainText, в которые пользователь вводит email и пароль для авторизации, и 2 кнопки, отвечающие за авторизацию и регистрацию пользователя. Для авторизации и регистрации пользователя, был создан класс Authentication. Также, для проверки подключения к интернету, был реализован статический класс, Далее, мы выполняем проверку на подключение к интернету, для этого был создан статический класс InternetConnection с методом isConnect для проверки на соединение с интернетом.

При авторизации, происходит проверка на наличие подключения к интернету, если интернет есть, то проверяем статус предыдущей сессии, был интернет, или нет и совпадает ли пользователь предыдущей сессии с тем пользователем, который авторизуется в данный момент. Для этих целей, в локальной базе данных, была создана таблица *settings* содержащая одну запись, со столбцами *status* для хранения факта о том, был интернет либо отсутствовал (значения «1» или «0» соответственно) и *email* для хранения идентификатора пользователя предыдущей сессии. Итак, если в предыдущей сессии интернет отсутствовал и значение идентификатора пользователя предыдущей сессии совпадает с польховователем, который авторизуется в данный момент, то данные пользователя, хранящиеся в глобальной базе данных в интернете, перезаписываются данными, хранящимися в локальной базе данных. Иначе, данные о пользователе перезаписываются из глобальной базы данных в локальную базу данных. Если же интернет отсутствует, то данные о пользователе загружаются из локальной базы данных.

При обращении к облачной базе, используется Android компонент `LiveData` . Глобальная база данных даёт ответ на вопрос о том, авторизован пользователь в интернете, или нет, если авторизован, то переходим на фрагмент аккаунта, иначе выполнение кода идёт дальше. Так как при попадании на фрагменты, приложение выполняет заново их код, и жизненный цикл начинается сначала инициализируется переменная типа `User`. В данную переменную заносится значение поля класса `AuthorizationViewModel` типа `User`. Дело в том, что `ViewModel` класс данного фрагмента хранит в себе текущего авторизованного пользователя `User` и доступ к этой переменной, из объекта класса `AuthorizationViewModel` есть во всём приложении. После возврата текущего пользователя, выполняется проверка, авторизован ли пользователь, если он авторизован, то выполняется переход на фрагмент аккаунта, иначе приложение ждёт дальнейших действий.

Работа фрагмента аккаунта:

После успешной авторизации, приложение, с фрагмента авторизации, переходит на фрагмент аккаунта. Во фрагменте аккаунта, жизненный цикл которого описывается классом `Account Fragment` выводится информация о текущем пользователе в соответствующих `View` элементах. Также на фрагменте существует кнопка выхода пользователя . При нажатии на кнопку, выполнение

кода переходит в слушатель кнопки. В слушателе вызывается метод *recording* для синхронизации с облачной базой данных из класса `AccountViewModel`. В данном методе, вызывается метод с аналогичным названием *recording* класса `RecordingToDB` пакета `Synchronization`. В данный метод передаётся текущий пользователь со своими данными, который завершает сессию. Логика данного метода построена следующим образом: в таблице *settings* запоминается идентификатор текущего пользователя, далее, данные о пользователе пересохраняются (а если такого пользователя в локальной базе данных не существует, то просто сохраняется) в локальной базе данных. Далее, происходит проверка на подключение к интернету, и статус подключения также, как и идентификатор пользователя, заносится в таблицу *settings*. Также, если соединение с интернетом есть, то данные о пользователе записываютс либо перезаписываются в облачную базу данных.

Далее, после синхронизации, выполняется выход пользователя из облачной базы и глобальному объекту текущего пользователя присваиванется значение.

Работа фрагмента регистрации:

При нажатии на кнопку регистрации, во фрагменте авторизации, приложение переходит во фрагмент регистрации. Во фрагменте регистрации инициализированы поля регистрации, а именно name, surname, email, password и повторение password. При нажатии на кнопку «зарегистрироваться» выполнение кода переходит в слушатель кнопки. В слушателе, вызывается метод объекта `ViewModel` данного класса, а параметрами, взятыми из полей ввода. Происходит регистрация и приложение переходит на фрагмент авторизации.

Работа фрагмента списка упражнений:

При переходе на левую вкладку нижнего меню приложения, действие переходит на фрагмент упражнений, который описан классом `ExercisesFragment` . В данном фрагменте, для перехода вкладок между частями тел, соответствующим упражнениям, используются элементы `TabLayout` и `ViewPager`. Элемент `ViewPager` нужен элементу `TabLayout` для логики перехода между фрагментами. Чтобы задать логику переходов, был реализован класс `TabLayoutPagerAdapter` , наследник `FragmentStatePagerAdapter`. В данном адаптере, метод `getItem`, возвращает объект класса `TabExercisesFragment`, описывающий сразу 3 фрагмента с упражнениями, в зависимости от части тела. В конструктор класса

приходит соответствующий индекс, зависящий от выбранной пользователем вкладки элемента TabLayout. При выполнении кода класса, описывающего вкладки, в зависимости от переданного индекса, «раздувается» тот или иной фрагмент, т.е. отправляется в MainActivity. Для каждого фрагмента определён элемент ListView. Также создан общий вид для элементов ListView. Для ListView упражнений реализуем адаптер, который будет принимать список упражнений одной части тела.

Для нажатия на элемент ViewList с упражнениями, объявлен слушатель. Внутри слушателя отправляется позиция элемента, на который нажали. После этого мы переходим на новый фрагмент для описания упражнения, для этого в новый фрагмент передаётся позиция для списка упражнений. Переход на другой фрагмент осуществляется с помощью компонента LiveData. В методе onResume общего класса для вкладки с упражнениями, подписываемся на LiveData, которая изменяется в слушателе на клик на упражнение.

После того, как мы перешли на фрагмент описания конкретного упражнения, по пришедшей позиции в списке упражнений, достаётся конкретное упражнение, поля которого вставляются в элементы фрагмента. На экране появляется название, изображение и описание упражнения, а в левом верхнем углу, появляется кнопка возврата на предыдущий фрагмент.

Работа фрагмента журнала тренировок:

При нажатии на среднюю вкладку нижнего меню, приложение переходит во фрагмент журнала тренировок. В методе onCreateView класса этого фрагмента, производится проверка на пользователя.

Если в объекте AuthorisationViewModel поле, типа User равно null то «раздуваем» view, отвечающий за фрагмент сообщения о том, что пользователь не авторизован, иначе, если он не равен null, то «раздуваем» view, отвечающий за управление тренировками.

Данный фрагмент представляет собой ListView, с элементами, представляющими собой тренировки пользователя. Элемент имеет TextView для вывода даты и кнопку «удалить», для удаления тренировки.

При нажатии на кнопку «добавить тренировку» на экране появляется виджет PickerDialog, представляющий из себя окошко выбора даты.

После нажатия кнопки «ок» в элементе ListView появляется тренировка с заданной датой.

Работа фрагмента списка записей об упражнениях в тренировке:

Нажимая на имеющуюся тренировку, в списке тренировок, приложение переходит на фрагмент списка заметок данной тренировки.

Во фрагменте списка записей, также, как и во фрагменте списка тренировок, существует кнопка добавления новых элементов. Также, в элементе `ListView`, установлен слушатель на нажатие на его элементы. После нажатия на элемент, описывающий запись об упражнении, приложение переходит на фрагмент изменения записи о тренировке.

Работа фрагмента изменения записи об упражнении:

После нажатия на элемент, описывающий запись о тренировке, приложение переходит на фрагмент изменения тренировки. Он представляет из себя фрагмент, на котором расположены выпадающий список `Spinner` с выбором уже имеющимися в приложении упражнениями, текстовое поле для ввода и кнопка сохранения изменений о заметке.

ЗАКЛЮЧЕНИЕ

По итогам выполненной работы, было создано мобильное Android приложение в помощь организации занятия спортом, с возможностью хранения личных данных пользователя и синхронизацией с облачной базой данных.

На основе рассмотренных аналогов, одной из целей разработки стало внесение простоты и гибкости в работу с личными записями пользователя. По сравнению с приведёнными аналогами, у пользователя есть возможность записи каждой тренировки, без привязки к программе, и в том формате, в котором ему удобно, так как ему предоставляется поле ввода, не привязанное к конкретному формату записи.

В процессе разработки, возникали различные трудности, а, в частности, с организацией бизнеслогики приложения, в силу использования облачной базы данных. Тем не менее, удалось добиться корректной работы приложения и цель данной работы была достигнута.

В данной работе были решены следующие задачи:

- Были рассмотрены аналоги фитнес-приложений для операционной системы Android.
- Была изучена структура Android приложений и их архитектурных компонетнов.
- Был разработан графический пользовательский интерфейс приложения.
- Была изучена и использована облачная база данных.
- Была изучена и использована встраиваемая локальная база данных.
- Было реализовано мобильное приложение для операционной системы Android.