Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра дискретной математики и
информационных технологий

Разработка мобильного приложения для отображения состояния трамвайной сети

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 421 группы
направления 09.03.01 – Информатика и вычислительная техника
факультета компьютерных наук и информационных технологий
Церулика Ивана Владимировича

Научный руководитель
к. ф.-м.н., доцент _____ Л.Б. Тяпаев
подпись, дата

Зав. кафедрой
к. ф.-м.н., доцент _____ Л.Б. Тяпаев
подпись, дата

Саратов 2017

ВВЕДЕНИЕ

В последнее время стала популярна разработка приложений для мониторинга пробок на дороге, для мониторинга движения общественного транспорта. Пользуясь такими приложениями, каждый может грамотно спланировать свой путь, экономя уйму времени. Правда пользоваться такими приложениями пока можно только в крупных городах России, такими как Москва, Санкт-Петербург, Екатеринбург. Но как обстоит дело с остальными городами, и почему им нельзя так просто повторить опыт мегаполисов? Дело в том, что главной деталью в системах мониторинга являются gps-приемники, установленные на машинах, трамваях, троллейбусах. Оснастить такими приемниками всю сеть общественного транспорта могут не все города. Что же им остается?

Было предложено мобильное приложение, используя которое каждый может отслеживать состояние электротранспорта. Однако принцип его работы основан на другой системе. В этом приложении пользователи сами будут сообщать о различных ситуациях на путях и дорогах, а привилегированные пользователи, модераторы, будут обновлять статусы направлений электротранспорта. Соответственно каждый сможет узнать, к примеру, стоит ли трамвай., ходит, ходит только дежурный или есть задержки в расписании.

Таким образом целью дипломной работы является разработка «с нуля» мобильного приложения для отображения состояния трамвайной сети.

Для достижения данной цели необходимо решить следующие задачи:

1. Определение архитектуры приложения;
2. Предварительное определение внешнего вида компонентов View, как части модели MVC;
3. Определение объектной модели, представленной в виде бизнес-сущностей;
4. Спецификация API, на основе которого будет осуществляться взаимодействие серверного приложения и Android-клиента;

5. Определение основных компонентов View, с учетом API;
6. Программная реализация каркаса Android приложения;
7. Реализация бизнес-объектов;
8. Определение интерфейсов, для связи с серверов через API, разработка интерфейсов с учетом особенностей Android API.

Структурно работа состоит из введения, четырех глав, заключения, списка использованных источников и четырех приложений. Названия глав:

1 Краткое описание архитектуры приложения;

1.1. VIEW;

1.2. MODEL;

1.3. CONTROLLER;

2 Разработка RESTful API;

3 Разработка каркаса Android приложения;

4 Особенности получения бизнес-сущностей по http-запросу.

Научная новизна работы: разработано программное обеспечение для отслеживания состояний электротранспорта, с возможностью просматривать состояния, модерировать, отсылать репорты об изменениях. Приложение было реализовано с помощью инструментов разработки Android SDK, преимущественно на Java.

Разработанное приложение предназначено для всех, кто пользуется электротранспортом.

Основное содержание работы

1 Краткое описание архитектуры приложения

Построим архитектуру приложения на основе фундаментального паттерна MVC. В свою очередь для реализации MVC используется достаточно большое число шаблонов проектирования (в зависимости от сложности архитектурного решения), основные из которых — «наблюдатель», «стратегия», «компоновщик». Некоторые из этих паттернов уже реализованы в Android out-of-box для тех или иных компонент.

Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») - схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

Паттерн MVC отлично подходит для мобильного приложения, которое содержит огромное количество графических элементов и представлений, а также бизнес-сущностей, которые не обладают очень сложной логикой. Также MVC позволит легко масштабировать приложение, из-за разделения логики поведения в двух остальных компонентах, Model и Controller.

Рассмотрим особенности компонент, исходя из платформы Android. Controller и View зачастую могут объединяться в каких-то элементах приложения, но это отнюдь не значит, что мы отходим от паттерна. Это означает, что элемент имеет два различных сценария поведения. Классическим примером может служить диалоговое окно, логика поведения при появлении которого никак не соприкасается с действием, совершаемым в нем.

Компонент View имеет много элементов внешне похожих друг с другом, но имеющих отличную логику. Это мешает структурировать отображаемые элементы, строить отношения между ними. Однако уникальным решением было бы использовать локальные и анонимные

классы, которые присутствуют в языке Java, на котором в свою очередь базируется классическая Android разработка.

Локальные классы декларируются внутри методов основного класса и могут быть использованы только внутри этих методов. Локальные классы имеют доступ к членам внешнего класса. Также могут иметь доступ и к локальным переменным, если на них стоит модификатор final (константа). Анонимные классы подобны локальным, за исключением того, что не декларируется ссылочная переменная для инстанса.

1.1 VIEW

Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели.

Опишем основные элементы для представления в Android приложении, без которых нельзя будет обойтись.

1. Все статусы.

View «Все статусы» дает пользователю узнать состояние всех направлений трамваев или троллейбусов. Для того, чтобы узнать более подробную информацию существует следующее View.

2. Подробная информация о статусе.

View «Подробная информация о статусе» дает пользователю подробную информацию о статусе, такую как:

- Актуальность данных (дата)
- Подробное описание
- Картинка (опционально)

3. Вход для модераторов.

View «Вход для модераторов» дает пользователю возможность войти в раздел модерации. Представляет собой поля ввода логина и пароля.

4. Отправление репорта пользователем.

Репорт представляет собой краткую информацию, на основе которой модераторы будут менять статусы состояний электротранспорта. Он

обязательно включает в себя текст, и опционально. View «Отправление репорта пользователем» представляет собой окно ввода необходимой информации.

5. Просмотр репортов от юзеров.

Это View доступно только авторизованным пользователям - модераторам. В нем можно просматривать репорты от пользователей.

6. Обновление статусов модератором.

View «Обновление статусов модератором» представляет собой элемент выбора типа, направления электротранспорта, элемент выбора текущего статуса и элемент ввода дополнительной информации. Впоследствии отправленная информация отобразится в 1 и 2 View.

1.2 MODEL

Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя свое состояние. Хорошей практикой считается строить модель на основе простых Java классах (POJO – пер. Простой Джава Объект), так как объекты этих классов легко сериализовать. В модели было определено несколько ключевых элементов:

1. SimpleStatus - представляет основную информацию о состоянии определенного направления электротранспорта.
2. StatusesMap - представляет собой Map из SimpleStatus с ключом «направление». В Android проекте этот объект будет представлять собой реализацию паттерна Singleton, так как основная информация о статусах направлений может быть нужна для разных сегментов приложения
3. Report – представляет собой всю информацию, собранную из репорта юзера. Используется как объект для отображения репортов юзеров, так и для отправления на сервер в виде JSON.
4. DetailedStatusInfo - представляет собой подробную информацию о статусе. Включает в себя два объекта: SimpleStatus и Report.

5. PhotoReport - это объект-обертка для изображения, который включен в объект Report. Пользователи имеют возможность отправлять фото, которое будет переводиться в формат Base64, который можно передавать строкой. В последствии фото будет отображаться на экране. Объект PhotoReport обеспечивает как кодирование картинки для отправки, так и декодирование для отображения.
6. Login – представляет собой пару логин/пароль. Используется для отправки в формате JSON в попытке авторизации модератора.
7. ReportList – представляет собой массив объектов Report.

1.3 CONTROLLER

Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений. В Android предусмотрено несколько способов перехвата событий взаимодействия пользователя с приложением.

Первый способ, приемник событий, - это интерфейс в классе View, который содержит один метод обратного вызова. Эти методы будут вызываться платформой Android, когда в результате взаимодействия пользователя с объектом пользовательского интерфейса срабатывает отображаемый объект View, в котором зарегистрирован приемник.

В данном случае Listener реагирует на выбранный пользователем вид транспорта для отображения направлений.

Второй способ перехвата событий - это обработчик событий. Стандартное API предоставляет возможность реагирования на нажатие клавиш, касание по экрану.

2 Разработка RESTful API

Прежде чем приступать к написанию клиентского приложения, сперва нужно определить внешние интерфейсы взаимодействия сервера и клиента.

В качестве такого интерфейса был выбран архитектурный стиль RESTful API.

В отличие от веб-сервисов (веб-служб) на основе SOAP, который использует XML, не существует «официального» стандарта передающихся данных для RESTful веб-API. Дело в том, что REST является архитектурным стилем, в то время как SOAP является протоколом. Несмотря на то, что REST не является стандартом сам по себе, большинство RESTful-реализаций используют стандарты, такие как HTTP, URL, JSON и XML. Мы будем использовать JSON, так как этот стандарт удобен одновременно и для машины, и для человека.

Данные будут передаваться по протоколу HTTP. HTTP (англ. HyperText Transfer Protocol — «протокол передачи гипертекста») - протокол прикладного уровня передачи данных (изначально — в виде гипертекстовых документов в формате «HTML», в настоящий момент используется для передачи произвольных данных). Основой HTTP является технология «клиент-сервер».

Основным объектом манипуляции в HTTP является ресурс, на который указывает URI (Uniform Resource Identifier) в запросе клиента. Обычно такими ресурсами являются хранящиеся на сервере файлы, но ими могут быть логические объекты или что-то абстрактное. Особенностью протокола HTTP является возможность указать в запросе и ответе способ представления одного и того же ресурса по различным параметрам: формату, кодировке, языку и т. д. (в частности, для этого используется HTTP-заголовок). Именно благодаря возможности указания способа кодирования сообщения, клиент и сервер могут обмениваться двоичными данными, хотя данный протокол является текстовым.

HTTP - протокол прикладного уровня; аналогичными ему являются FTP и SMTP. Обмен сообщениями идёт по обыкновенной схеме «запрос-ответ». В отличие от многих других протоколов, HTTP не сохраняет своего состояния. Это означает отсутствие сохранения промежуточного состояния

между парами «запрос-ответ». Компоненты, использующие HTTP, могут самостоятельно осуществлять сохранение информации о состоянии, связанной с последними запросами и ответами (например, «куки» на стороне клиента, «сессии» на стороне сервера). Браузер, посылающий запросы, может отслеживать задержки ответов. Сервер может хранить IP-адреса и заголовки запросов последних клиентов. Однако сам протокол не осведомлён о предыдущих запросах и ответах, в нём не предусмотрена внутренняя поддержка состояния, к нему не предъявляются такие требования.

Рассмотрим типы http-запросов. GET - это клиентский запрос данных. Веб браузер посылает сообщение GET, чтобы извлекать страницы с веб-сервера. Как только сервер получает GET запрос, он отвечает строкой статуса, например, HTTP/1.1 200 OK, и собственно самим сообщением, основной частью которого может быть запрашиваемый файл, сообщение об ошибке, либо какая-либо другая информация. POST и PUT используются, чтобы посылать сообщения, которые загружают данные на веб-сервер. Например, когда пользователь вводит данные в форму, встроенную в веб страницу, POST включает данные в сообщение, посылаемое на сервер. PUT загружает ресурсы или контент на веб-сервер.

Следует понимать, что программное обеспечение при взаимодействии с Интернетом представляет пользователя, и программе следует информировать пользователя о любых действиях, которые он может произвести, но которые могут иметь непредсказуемое значение для него или других лиц. Поэтому запросы http запросы на сервер будут двух типов: GET(безопасный) и POST.

В нашем RESTful API будут встречаться запросы, которые должны быть доступны только привилегированным пользователям, модераторам. Поэтому необходимо добавить компонент безопасности на сервер, с которым будет взаимодействовать Android-приложение. Есть несколько способов добавить security к RESTful API.

1. Basic Authentication.

Basic Authentication - REST-клиент (пользователь) использует свои логин и пароль для получения доступа к REST-сервису через http-запрос. Как незашифрованный текст, передаются логин и пароль пользователя. Эти данные кодируются простым Base64 и могут быть легко декодированы любым пользователем. При использовании Basic Authentication, обязательно должен использоваться https протокол для передачи данных сервису.

2. Digest Authentication.

Digest Authentication - это почти тоже самое что первый метод, только логин и пароль передаются в зашифрованном виде, а не как обычный текст. Логин и пароль шифруются MD5 алгоритмом и его достаточно сложно расшифровать. При этом подходе можно использовать незащищенное http-соединение и не бояться, что пароль будет перехвачен злоумышленниками. Реализация серверной части остается такой же.

3. Token Authentication

Суть этого способа заключается в том, что пользователь, используя свои логин и пароль авторизуется в приложение и получает token для доступа к Rest сервису. Доступ к сервису, который выдает token должен обязательно быть осуществлен через https соединение, доступ к Rest сервису можно сделать через обычный http. Token должен содержать логин, пароль, так же может содержать expiration time и роли пользователя, а так же любую нужную для вашего приложения информацию. После того как token готов и, к примеру, все его параметры разделены двоеточием или другим символом, или сериализованы, как json или xml-объект, его необходимо зашифровать, прежде чем отдать пользователю. Нужно учесть, что только REST-сервис должен знать, как расшифровывать этот token. После того как token приходит на REST-сервис он его расшифровывает и получает все необходимые данные для аутентификации и, если надо, авторизации Rest клиента. Реализация такого security будет кардинально отличатся от предыдущих двух.

Мы будем использовать последний способ, авторизацию через token, так как в случае его использования мы будем иметь возможность передавать меньше данных и, используя паттерн Singleton, легко и безопасно осуществлять запросы, требующие авторизации. Такие запросы обязательно требуют передачи информации, но в случае с GET-запросом, по спецификации мы не должны иметь тела метода, поэтому сам token будет помещаться в headers, то есть как параметр http-запроса.

3 Разработка каркаса Android приложения

Разработка каркаса - обязательный шаг, перед созданием всей логики приложения, по причине того, что логика должна строиться исходя из нужд доступных представлений.

Существует два основных способа представить информацию в Android приложении. Посредством класса Activity и посредством класса Fragment.

Activity - это компонент приложения, который выдает экран, и с которым пользователи могут взаимодействовать для выполнения каких-либо действий. Activity, которая запускается первой, считается главной. Из нее можно запустить другую активность. Причем не только ту, которая относится к нашему приложению, но и другого приложения. Пользователю будет казаться, что все запускаемые им активности являются частями одного приложения, хотя на самом деле они могут быть определены в разных приложениях и работают в разных процессах. Activity имеет жизненный цикл - начало, когда Android создает экземпляр активности, промежуточное состояние, и конец, когда экземпляр уничтожается системой и освобождает ресурсы.

У Activity есть метод OnCreate, который вызывается при инициализации. В нем удобно сохранять ссылки на компоненты View, чтобы сразу описать их логику поведения или использовать в дальнейшем.

Фрагмент (класс Fragment) представляет поведение или часть пользовательского интерфейса в операции (класс Activity). Разработчик

может объединить несколько фрагментов в одну операцию для построения многопанельного пользовательского интерфейса и повторного использования фрагмента в нескольких операциях. Фрагмент можно рассматривать как модульную часть операции. Фрагменты впервые появились в Android версии 3.0 (API уровня 11), главным образом, для обеспечения большей динамичности и гибкости 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 позволяют делать это, избавляя разработчика от необходимости управлять сложными изменениями в иерархии представлений. Разбивая макет операции на фрагменты, разработчик получает возможность модифицировать внешний вид операции в ходе выполнения и сохранять эти изменения в стеке переходов назад, которым управляет операция.

Фрагмент подобен Activity и так же имеет жизненный цикл.

В отличии от Activity в жизненном цикле Фрагмента можно встретить дополнительные методы, такие как OnCreateView. В нем удобнее получать ссылки на компоненты View, т.к. поиск ведется от главного макета (Layout) Фрагмента.

Оба класса, Activity и Fragment, используют макеты. С помощью XML-элементов, который имеется в Android, можно быстро и просто создавать макеты пользовательского интерфейса и содержащиеся в нем элементы, точно так же, как при создании веб-страниц в HTML - с помощью вложенных элементов.

4 Особенности получения бизнес-сущностей по http-запросу

Как уже известно, вся получаемая с сервера информация поступает на клиентское приложение с помощью http-запроса в виде JSON объекта, который впоследствии преобразуется в бизнес-сущность. Обычно для того, чтобы получить все нужные бизнес-сущности в клиентских приложениях

создается компонент по паттерну DAO, но в виду особенностей Android, этот интерфейс реализован иначе.

Два пакета реализуют доступ к информации с сервера - это пакет http и пакет json. А также вспомогательные классы.

Классы-геттеры http-запросов в пакете http могут выдать только экземпляр класса Result, который выдает результат-строку (если есть) и код результата http-запроса. По коду http-запроса осуществляется отлов ошибок.

Ниже представлен код класса Result:

```
public class Result {
    private int code;        //поле кода http,
    private String result;  //поле результата

    public Result(int code, String result) {
        this.code = code;    //конктруктор, если результат
        this.result = result;//ожидаем
    }

    public Result(int code){ //конструктор, если результат
        this.code = code;    //не ожидается
        result = "";
    }

    public int getCode() { // геттер
        return code;
    }

    public String getStringResult() { //геттер
        return result;
    }
}
```

Если бы был реализован паттерн DAO, то пришлось бы описывать много java-исключений, что сделало бы компонент громоздким, а error-handling неудобный.

Error-handling частично содержится в классах типа Fragment и Activity, но в основном вынесен в отдельные классы, в пакете handlers. Они наследуются от out-of-box обработчика Handler. Разберемся, зачем понадобился отдельный обработчик и почему важно его использовать.

По умолчанию все компоненты одного приложения работают в одном процессе, и большинство приложений не должно менять это поведение. В

каждом процессе запускаются потоки. Если внутри, в главном потоке (UI Thread), запустить, например, долгий сетевой вызов, как мы и собираемся сделать, или какую-то тяжелую инициализацию, то приложение зависнет, зависнет его интерфейса и, скорее всего, приложение предложит сделать Force close. Впрочем, работа службы в том же потоке, что и остальное приложение, имеет свои плюсы - есть доступ к элементам интерфейса. Если бы служба работала в другом потоке, доступ к UI бы отсутствовал.

Вся обработка события - это кейс-блок с условиями вывода сообщений об ошибках. Если понадобится поменять содержание сообщения, то в анонимном классе просто следует добавить перед вызовов метода super() отдельный блок отлова события с нужной обработкой.

Вывод ошибок стандартизирован с помощью класса Toast. Toast - это простое всплывающее сообщение, которое занимает минимальное место на экране. Toast обладает фиксированным местом появления и стандартным временем отображения на экране. С этим сообщением нельзя взаимодействовать.

ЗАКЛЮЧЕНИЕ

В ходе данной дипломной были сделаны следующие задачи:

1. Предварительное определение внешнего вида компонентов View, как части модели MVC;
2. Спецификация API, на основе которого будет осуществляться взаимодействие серверного приложения и Android клиента;
3. Определение основных сущностей бизнес-объектов с учетом API;
4. Определение основных компонентов View, с учетом API;
5. Программная реализация каркаса Android приложения;
6. Реализация бизнес-объектов;
7. Написание интерфейсов, для связи с серверов через API.

Было спроектировано и реализовано Android-приложение, позволяющее просматривать текущее состояние электротранспорта, осуществлять модерацию состояний, и отсылать сообщения об изменении состояний. Правильно выбранная архитектура позволит легко масштабировать приложение, позволит использовать его не только для города, для которого оно создавалось (Саратов), но и для других городов.