**Минобрнауки России**

федеральное государственное бюджетное образовательное учреждение высшего образования

«Вологодский государственный университет»

(ВоГУ)

**Университетский колледж**

**ОТЧЁТ ПО ПРАКТИКЕ**

**Студента Плотникова Алексея Николаевича**

**Специальность 09.02.07 «Информационные системы и программирование»**

**Группа 8СПО09ИСП31-20**

**Преподаватель:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Оценка:\_\_\_\_\_\_\_\_\_\_\_\_\_ «\_\_\_»\_\_\_\_\_\_\_\_\_\_20\_\_г.**

**ВОЛОГДА**

СОДЕРЖАНИЕ

[1 БЕЗОПАСНОСТЬ ТРУДА. ЗНАКОМСТВО С ПРЕДПРИЯТИЕМ. ЗАКРЕПЛЕНИЕ РАБОЧЕГО МЕСТА. 3](#_Toc137924532)

[1.1 Инструктаж по охране труда 3](#_Toc137924533)

[1.2 Описание предприятия и рабочего места 3](#_Toc137924534)

[2 OS WINDOWS: ЗАГРУЗКА, НАСТРОЙКА, УПРАВЛЕНИЕ, ОБСЛУЖИВАНИЕ. 5](#_Toc137924535)

[3 РАЗРАБОТКА СПЕЦИФИКАЦИЙ ОТДЕЛЬНЫХ КОМПОНЕНТ 6](#_Toc137924536)

[3.1 Техническое задание на разработку 6](#_Toc137924537)

[3.2 Описание средств разработки 6](#_Toc137924538)

[2 РАЗРАБОТКА КОДА ПРОГРАММНОГО ПРОДУКТА НА ОСНОВЕ ГОТОВЫХ СПЕЦИФИКАЦИЙ НА УРОВНЕ МОДУЛЯ 7](#_Toc137924539)

[2.1 Разработка графического интерфейса 7](#_Toc137924540)

[2.2 Разработка программного функционала 16](#_Toc137924541)

[3 ОТЛАДКА, ТЕСТИРОВАНИЕ И ОПТИМИЗАЦИЯ ПРОГРАММНЫХ МОДУЛЕЙ 48](#_Toc137924542)

[3.1 Тестирование логики программы 48](#_Toc137924543)

[3.2 Тестирование интерфейса пользователя 48](#_Toc137924544)

[ВЫВОД 49](#_Toc137924545)

[ПРИЛОЖЕНИЕ 1 50](#_Toc137924546)

[ПРИЛОЖЕНИЕ 2 53](#_Toc137924547)

[ПРИЛОЖЕНИЕ 3 55](#_Toc137924548)

1 БЕЗОПАСНОСТЬ ТРУДА. ЗНАКОМСТВО С ПРЕДПРИЯТИЕМ. ЗАКРЕПЛЕНИЕ РАБОЧЕГО МЕСТА.

1.1 Инструктаж по охране труда

В первую очередь в ходе прохождения производственной практики был изучен инструктаж по охране труда и технике безопасности. В нём содержались правила, описывающие что необходимо и запрещено делать на рабочем месте. Основные и самые важные правила:

* Необходимо соблюдать порядок на рабочем месте, не держать на столе лишних предметов;
* Необходимо перед включением оборудования в электросеть, убедиться в исправности электроприборов;
* Запрещено приступать к работе в состоянии алкогольного или наркотического опьянения;
* Запрещено курить на рабочем месте;
* Экран должен быть ниже уровня глаз примерно на 5 градусов, и располагаться в прямой плоскости;
* Расстояние от монитора до глаз должно быть в пределах от 60 до 80 см.
* Рекомендуется после часа работы делать перерыв в целях снижения утомляемости;
* Необходимо блокировать профиль пользователя операционной системы всякий раз, когда необходимо покинуть рабочее место;
* Запрещено использовать чужое оборудование без разрешения;
* Запрещено производить самостоятельное вскрытие оборудования;
* Запрещено допускать попадание влаги на электроприборы.

1.2 Описание предприятия и рабочего места

ООО «Макси инновации» занимается разработкой программного обеспечения для розничной сети магазинов «Макси». Розничная сеть «Макси» насчитывает более 60 магазинов в 7 городах России. Сеть магазинов «Макси» занимается продажей товаров, закупаемых у разных поставщиков, а также продукции собственного производства. Также компания «Макси» занимается строительством коммерческой и жилой недвижимости и оптовой торговлей.

ООО «Макси инновации» разрабатывает множество программных продуктов для ведения бизнеса. Среди них есть торговая система для контроля бизнес-процессов; корпоративная социальная сеть, в которой есть полезная информация для сотрудников; мобильное приложение и сайт для покупателей, в которых можно смотреть и заказывать товары, использовать персональные скидки; мобильная торговая система для терминалов сбора данных, используемых сотрудниками.

При прохождении практики я был направлен в отдел программного обеспечения в команду, занимающуюся разработкой приложений для терминалов сбора данных и кассового оборудования. Моё рабочее место находилось в бизнес-центре на улице Молодёжной 25, города Вологды, в 225 кабинете. Мне был выделен персональный компьютер с операционной системой Windows 10, с гостевым доступом к виртуальной машине с такой же операционной системой. Также мне была предоставлена гостевая учетная запись на GitLab и доступ к корпоративному Wi-Fi.

2 OS WINDOWS: ЗАГРУЗКА, НАСТРОЙКА, УПРАВЛЕНИЕ, ОБСЛУЖИВАНИЕ.

В «Макси» в целях безопасности у программистов нет прав администратора, из-за чего множество программ нельзя установить самостоятельно. Чтобы установить нужное ПО, необходимо делать запрос системным администраторам. Таким образом, для полноценной работы мне потребовалось запросить установку клиента для Git – Sourcetree, так как для его установки требуется .Net Framework 4.8, установка которого требует подтверждение от администратора системы. Также были установлены Git и IDE Android Studio.

После этого я настроил SSH-ключи для Git и установил их в настройках Sourcetree, чтобы работать с удалённым репозиторием на GitLab. Благодаря этому я смог скачать проект с мобильной торговой системой на предоставленный мне компьютер.

3 РАЗРАБОТКА СПЕЦИФИКАЦИЙ ОТДЕЛЬНЫХ КОМПОНЕНТ

3.1 Техническое задание на разработку

Мне предстояло разработать модуль в мобильной торговой системе для учёта подключённого оборудования. Требования к разрабатываемому программному обеспечению находятся в приложении 1.

3.2 Описание средств разработки

Для разработки мобильных приложений под Android чаще всего используются языки Java и Kotlin. Kotlin является более перспективным языком в Android разработке. В компании «Макси» для разработки мобильных приложений с основном используют язык Java 1.8 в связи с поддержкой имеющейся архитектуры.

В качестве IDE для разработки модуля взята Android Studio, так как она рекомендуется для использования в официальной документации Android. В ней есть множество необходимых инструментов, облегчающих разработку.

Для работы с локальными данными на устройстве используется система управления базами данных (СУБД) SqLite, так как она очень распространена в разработке под Android. Также для удобной работы с данными как с объектами используется ORMLite.

Чтобы выполнять некоторые действия асинхронно применяется распространённая библиотека реактивного программирования RxJava.

Мобильная торговая система написана с применением принципов чистой архитектуры, заключающейся в разбиении приложения на следующие слои: представление, бизнес-логика, слой данных и внешние сервисы. Компоненты системы должны работать с абстракциями вместо конкретных реализаций. Распространёнными инструментами являются библиотеки Dagger и Hilt. Hilt основан на Dagger и является более удобным и простым в применении, но на данный момент в «Макси» используется Dagger.

2 РАЗРАБОТКА КОДА ПРОГРАММНОГО ПРОДУКТА НА ОСНОВЕ ГОТОВЫХ СПЕЦИФИКАЦИЙ НА УРОВНЕ МОДУЛЯ

2.1 Разработка графического интерфейса

В первую очередь была разработана активность списка записей о подключённом оборудовании. Активность в соответствие с рисунком 1. Код активности:

<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

xmlns:app="http://schemas.android.com/apk/res-auto"

tools:context=".connectedEquipment.presentation.view.ConnectedEquipmentListActivity">

<androidx.recyclerview.widget.RecyclerView

android:id="@+id/rv\_connected\_equipment"

android:layout\_width="match\_parent"

android:layout\_height="0dp"

app:layout\_constraintBottom\_toTopOf="@+id/btn\_add"

app:layout\_constraintTop\_toTopOf="parent"/>

<Button

android:id="@+id/btn\_add"

android:layout\_height="wrap\_content"

android:layout\_width="wrap\_content"

android:paddingHorizontal="16dp"

android:paddingVertical="4dp"

android:text="@string/add"

android:onClick="btnAddClick"

style="@style/btn\_primary\_style"

app:layout\_constraintLeft\_toLeftOf="parent"

app:layout\_constraintRight\_toRightOf="parent"

app:layout\_constraintTop\_toBottomOf="@+id/rv\_connected\_equipment"

app:layout\_constraintBottom\_toBottomOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 1 – Активность списка записей о подключённом оборудовании.

Затем был разработан интерфейс активности работы с записью. Интерфейс в соответствие с рисунком 2. Код интерфейса активности:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

tools:context=".connectedEquipment.presentation.view.ConnectedEquipmentItemActivity">

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="vertical"

android:gravity="top"

android:padding="8dp">

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textColor="@color/gray"

android:text="@string/building\_address"/>

<Spinner

android:id="@+id/sp\_address"

android:layout\_width="match\_parent"

android:layout\_height="48dp"

style="@style/sp\_primary\_style">

</Spinner>

</LinearLayout>

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="vertical"

android:padding="8dp">

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textColor="@color/gray"

android:text="@string/switchboard"/>

<Spinner

android:id="@+id/sp\_switchboard"

android:layout\_width="match\_parent"

android:layout\_height="48dp"

style="@style/sp\_primary\_style">

</Spinner>

</LinearLayout>

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="vertical"

android:padding="8dp">

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textColor="@color/gray"

android:text="@string/switchboard\_port"/>

<EditText

android:id="@+id/et\_port"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:height="48dp"

android:inputType="number"

style="@style/et\_primary\_style">

</EditText>

</LinearLayout>

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="vertical"

android:padding="8dp">

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textColor="@color/gray"

android:text="@string/cable"/>

<Spinner

android:id="@+id/sp\_cables"

android:layout\_width="match\_parent"

android:layout\_height="48dp">

</Spinner>

</LinearLayout>

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="vertical"

android:padding="8dp">

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textColor="@color/gray"

android:text="@string/cable\_length"/>

<EditText

android:id="@+id/et\_cable\_length"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

style="@style/et\_primary\_style"

android:digits="0123456789."

android:height="48dp">

</EditText>

</LinearLayout>

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="vertical"

android:padding="8dp">

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textColor="@color/gray"

android:text="@string/connected\_equipment\_ip"/>

<EditText

android:id="@+id/et\_connected\_equipment\_ip"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:digits="0123456789."

android:height="48dp"

style="@style/et\_primary\_style">

</EditText>

</LinearLayout>

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="vertical"

android:padding="8dp">

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textColor="@color/gray"

android:text="@string/connected\_equipment\_mac"/>

<EditText

android:id="@+id/et\_connected\_equipment\_mac"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:digits="0123456789abcdefABCDEF-:"

android:height="48dp"

style="@style/et\_primary\_style">

</EditText>

</LinearLayout>

<Button

android:id="@+id/btn\_save"

style="@style/btn\_primary\_style"

android:onClick="btnSaveClick"

android:layout\_width="match\_parent"

android:layout\_height="48dp"

app:layout\_constraintBottom\_toBottomOf="parent"

android:layout\_marginHorizontal="64dp"

android:layout\_marginVertical="24dp"

android:text="@string/save" />

</LinearLayout>

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рисунок 2 – Интерфейс активности работы с записью.

Для эмуляции входа в приложение и открытия модуля были разработаны соответствующие активности. Активность входа в приложение в соответствие с рисунком 3. Код активности в хода в приложение:

<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context=".main.presentation.MainActivity">

<TextView

android:layout\_gravity="center\_horizontal"

app:layout\_constraintTop\_toTopOf="parent"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintEnd\_toEndOf="parent"

android:layout\_height="wrap\_content"

android:layout\_width="wrap\_content"

android:layout\_marginVertical="48dp"

android:gravity="center\_horizontal"

android:text="@string/header\_activity\_main"

style="@style/tv\_header\_style" />

<Button

android:id="@+id/btn\_login"

style="@style/btn\_primary\_style"

android:onClick="btnLoginClick"

android:layout\_width="wrap\_content"

android:layout\_height="48dp"

app:layout\_constraintLeft\_toLeftOf="parent"

app:layout\_constraintRight\_toRightOf="parent"

app:layout\_constraintBottom\_toBottomOf="parent"

android:paddingHorizontal="16dp"

android:layout\_marginVertical="24dp"

android:text="@string/login" />

</androidx.constraintlayout.widget.ConstraintLayout>

Активность выбора модуля мобильной торговой системы в соответствие с рисунком 4. Код активности выбора модуля:

<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context=".modules.presentation.view.ModulesActivity">

<LinearLayout

android:layout\_height="wrap\_content"

android:layout\_width="match\_parent"

tools:ignore="MissingConstraints">

<androidx.recyclerview.widget.RecyclerView

android:id="@+id/rv\_module"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent" />

</LinearLayout>

</androidx.constraintlayout.widget.ConstraintLayout>

Изображение выглядит как текст, снимок экрана, Операционная система, программное обеспечение

Автоматически созданное описание

Рисунок 3 – Активность входа в приложение.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Автоматически созданное описание

Рисунок 4 – Активность выбора модуля.

2.2 Разработка программного функционала

После того как был разработан интерфейс приложения необходимо реализовать его функционал. В первую очередь был разработан слой данных приложения и написаны классы для сущностей предметной области:

* Building – описывает сооружения компании «Макси»;
* Cable – описывает модели кабелей, которыми производится подключение оборудования;
* SwitchboardModel – описывает модели распределительных щитов;
* Switchboard – описывает распределительные щиты, находящиеся в сооружениях;
* ConnectedEquipment – описывает оборудование, подключённое к распределительным щитам.

Сущность Building состоит из идентификатора и адреса сооружения. Код класса Building:

package com.example.mts.connectedEquipment.domain.entity;

import com.j256.ormlite.field.DatabaseField;

import com.j256.ormlite.field.ForeignCollectionField;

import com.j256.ormlite.table.DatabaseTable;

import java.io.Serializable;

import java.util.Collection;

import java.util.List;

/\*\*

\* Сущность, описывающая здание.

\*/

@DatabaseTable(tableName = "BUILDING")

public class Building implements Serializable {

public static final String UID = "UID";

public static final String ADDRESS = "ADDRESS";

@DatabaseField(columnName = UID, generatedId = true)

private int id;

@DatabaseField(columnName = ADDRESS)

private String address;

@ForeignCollectionField(eager = true)

private Collection<Switchboard> switchboardList;

public Building() {}

public Building(int id, String address, List<Switchboard> switchboardList) {

this.id = id;

this.address = address;

this.switchboardList = switchboardList;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Building that = (Building) o;

return id == that.id;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public Collection<Switchboard> getSwitchboardList() {

return switchboardList;

}

public void setSwitchboardList(List<Switchboard> switchboardList) {

this.switchboardList = switchboardList;

}

}

Сущность Cable состоит из идентификатора и названия модели. Код сущности:

package com.example.mts.connectedEquipment.domain.entity;

import com.j256.ormlite.field.DatabaseField;

import com.j256.ormlite.field.ForeignCollectionField;

import com.j256.ormlite.table.DatabaseTable;

import java.io.Serializable;

import java.util.Collection;

import java.util.List;

/\*\*

\* Сущность, описывающая модель кабеля.

\*/

@DatabaseTable(tableName = "CABLE")

public class Cable implements Serializable {

public static final String UID = "UID";

public static final String NAME = "NAME";

@DatabaseField(columnName = UID, generatedId = true)

private int id;

@DatabaseField(columnName = NAME)

private String name;

@ForeignCollectionField()

private Collection<ConnectedEquipment> connectedEquipmentList;

public Cable() {}

public Cable(int id, String name, List<ConnectedEquipment> connectedEquipmentList) {

this.id = id;

this.name = name;

this.connectedEquipmentList = connectedEquipmentList;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Collection<ConnectedEquipment> getConnectedEquipmentList() {

return connectedEquipmentList;

}

public void setConnectedEquipmentList(List<ConnectedEquipment> connectedEquipmentList) {

this.connectedEquipmentList = connectedEquipmentList;

}

}

Сущность SwitchboardModel состоит из идентификатора, названия модели и максимального количества портов в щите. Код сущности:

package com.example.mts.connectedEquipment.domain.entity;

import com.j256.ormlite.field.DatabaseField;

import com.j256.ormlite.field.ForeignCollectionField;

import com.j256.ormlite.table.DatabaseTable;

import java.io.Serializable;

import java.util.Collection;

import java.util.List;

/\*\*

\* Сущность, описывающая модель распределительного щита.

\*/

@DatabaseTable(tableName = "SWITCHBOARD\_MODEL")

public class SwitchboardModel implements Serializable {

public static final String UID = "UID";

public static final String NAME = "NAME";

public static final String MAX\_PORT\_AMOUNT = "MAX\_PORT\_AMOUNT";

@DatabaseField(columnName = UID, generatedId = true)

private int id;

@DatabaseField(columnName = NAME)

private String name;

@DatabaseField(columnName = MAX\_PORT\_AMOUNT)

private int maxPortAmount;

@ForeignCollectionField()

private Collection<Switchboard> switchboardList;

public SwitchboardModel() {}

public SwitchboardModel(int id, String name, int maxPortAmount, List<Switchboard> switchboardList) {

this.id = id;

this.name = name;

this.maxPortAmount = maxPortAmount;

this.switchboardList = switchboardList;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getMaxPortAmount() {

return maxPortAmount;

}

public void setMaxPortAmount(int maxPortAmount) {

this.maxPortAmount = maxPortAmount;

}

public Collection<Switchboard> getSwitchboardList() {

return switchboardList;

}

public void setSwitchboardList(List<Switchboard> switchboardList) {

this.switchboardList = switchboardList;

}

}

Сущность Switchboard состоит из идентификатора, инвентарного номера, модели и ссылки на сооружение, в котором текущий щит располагается. Код сущности:

package com.example.mts.connectedEquipment.domain.entity;

import com.j256.ormlite.field.DatabaseField;

import com.j256.ormlite.field.ForeignCollectionField;

import com.j256.ormlite.table.DatabaseTable;

import java.io.Serializable;

import java.util.Collection;

import java.util.List;

/\*\*

\* Сущность, описывающая распределительный щит.

\*/

@DatabaseTable(tableName = "SWITCHBOARD")

public class Switchboard implements Serializable {

public static final String UID = "UID";

public static final String INVENTORY\_NUMBER = "INVENTORY\_NUMBER";

@DatabaseField(columnName = UID, generatedId = true)

private int id;

@DatabaseField(columnName = INVENTORY\_NUMBER)

private String inventoryNumber;

@DatabaseField(foreign = true, foreignAutoRefresh = true)

private Building building;

@DatabaseField(foreign = true, foreignAutoRefresh = true)

private SwitchboardModel model;

@ForeignCollectionField()

private Collection<ConnectedEquipment> connectedEquipmentList;

public Switchboard() {}

public Switchboard(int id, Building building, SwitchboardModel model, List<ConnectedEquipment> connectedEquipmentList) {

this.id = id;

this.building = building;

this.model = model;

this.connectedEquipmentList = connectedEquipmentList;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Switchboard that = (Switchboard) o;

return id == that.id;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public Building getBuilding() {

return building;

}

public void setBuilding(Building building) {

this.building = building;

}

public SwitchboardModel getModel() {

return model;

}

public void setModel(SwitchboardModel model) {

this.model = model;

}

public Collection<ConnectedEquipment> getConnectedEquipmentList() {

return connectedEquipmentList;

}

public void setConnectedEquipmentList(List<ConnectedEquipment> connectedEquipmentList) {

this.connectedEquipmentList = connectedEquipmentList;

}

public String getInventoryNumber() {

return inventoryNumber;

}

public void setInventoryNumber(String inventoryNumber) {

this.inventoryNumber = inventoryNumber;

}

}

Сущность ConnectedEquipment состоит из идентификатора, номера занятого порта, модели используемого кабеля, его длины, ссылки на щит к которому подключено оборудование, IP и MAC адресов оборудования. Код сущности:

package com.example.mts.connectedEquipment.domain.entity;

import com.j256.ormlite.field.DatabaseField;

import com.j256.ormlite.table.DatabaseTable;

import java.io.Serializable;

/\*\*

\* Сущность, описывающая подключённое оборудование.

\*/

@DatabaseTable(tableName = "CONNECTED\_EQUIPMENT")

public class ConnectedEquipment implements Serializable {

public static final String UID = "UID";

public static final String PORT\_NUMBER = "PORT\_NUMBER";

public static final String CABLE\_LENGTH = "CABLE\_LENGTH";

public static final String IP = "IP";

public static final String MAC = "MAC";

public static final String U\_IDX\_SWITCHBOARD\_PORT = "U\_IDX\_SWITCHBOARD\_PORT";

@DatabaseField(columnName = UID, generatedId = true)

private int id;

@DatabaseField(foreign = true, foreignAutoRefresh = true, uniqueIndexName = U\_IDX\_SWITCHBOARD\_PORT)

private Switchboard switchboard;

@DatabaseField(columnName = PORT\_NUMBER, uniqueIndexName = U\_IDX\_SWITCHBOARD\_PORT)

private int portNumber;

@DatabaseField(foreign = true, foreignAutoRefresh = true)

private Cable cable;

@DatabaseField(columnName = CABLE\_LENGTH)

private int cableLength;

@DatabaseField(columnName = IP)

private String ip;

@DatabaseField(columnName = MAC)

private String mac;

public ConnectedEquipment() {}

public ConnectedEquipment(int id, Switchboard switchboard, int portNumber, Cable cable, int cableLength, String ip, String mac) {

this.id = id;

this.switchboard = switchboard;

this.portNumber = portNumber;

this.cable = cable;

this.cableLength = cableLength;

this.ip = ip;

this.mac = mac;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public Switchboard getSwitchboard() {

return switchboard;

}

public void setSwitchboard(Switchboard switchboard) {

this.switchboard = switchboard;

}

public int getPortNumber() {

return portNumber;

}

public void setPortNumber(int portNumber) {

this.portNumber = portNumber;

}

public Cable getCable() {

return cable;

}

public void setCable(Cable cable) {

this.cable = cable;

}

public int getCableLength() {

return cableLength;

}

public void setCableLength(int cableLength) {

this.cableLength = cableLength;

}

public String getIp() {

return ip;

}

public void setIp(String ip) {

this.ip = ip;

}

public String getMac() {

return mac;

}

public void setMac(String mac) {

this.mac = mac;

}

}

Далее для этих сущностей были разработаны DAO (data access object) классы, через которые происходила работа с данными в базе. Далее был разработан источник данных ConnectedEquipmentDataSource, который предоставляет доступ слою бизнес-логики выполнять операции над данными. Код интерфейса ConnectedEquipmentDataSource:

package com.example.mts.connectedEquipment.data.database;

import com.example.mts.connectedEquipment.domain.entity.Building;

import com.example.mts.connectedEquipment.domain.entity.Cable;

import com.example.mts.connectedEquipment.domain.entity.ConnectedEquipment;

import com.example.mts.connectedEquipment.domain.entity.Switchboard;

import java.sql.SQLException;

import java.util.List;

/\*\*

\* Источник данных подключённого оборудования.

\*/

public interface ConnectedEquipmentDataSource {

/\*\*

\* Возвращает список подключённого оборудования.

\* @return список подключённого оборудования.

\* @throws SQLException

\*/

List<ConnectedEquipment> getConnectedEquipment() throws SQLException;

/\*\*

\* Удаляет указанное подключённое оборудование.

\* @param id идентификатор записи подключённого оборудования.

\* @throws SQLException

\*/

void deleteConnectedEquipment(int id) throws SQLException;

/\*\*

\* Добавляет запись о подключённом оборудовании в БД.

\* @param connectedEquipment запись подключённого оборудования, которую нужно добавить.

\* @throws SQLException

\*/

void createConnectedEquipment(ConnectedEquipment connectedEquipment) throws SQLException;

/\*\*

\* Обновляет запись о подключённом оборудовании в БД.

\* @param connectedEquipment запись подключённого оборудования, которую нужно обновить.

\* @throws SQLException

\*/

void updateConnectedEquipment(ConnectedEquipment connectedEquipment) throws SQLException;

/\*\*

\* Возвращает список моделей кабелей.

\* @return список моделей кабелей.

\* @throws SQLException

\*/

List<Cable> getCables() throws SQLException;

/\*\*

\* Возвращает список зданий.

\* @return список зданий.

\* @throws SQLException

\*/

List<Building> getBuildings() throws SQLException;

}

Реализация источника данных заключена в классе ConnectedEquipmentDataSourceImpl, который для выполнения операций использует DAO. Код класса ConnectedEquipmentDataSourceImpl:

package com.example.mts.connectedEquipment.data.database;

import com.example.mts.connectedEquipment.data.dao.BuildingDAO;

import com.example.mts.connectedEquipment.data.dao.CableDAO;

import com.example.mts.connectedEquipment.data.dao.ConnectedEquipmentDAO;

import com.example.mts.connectedEquipment.data.dao.SwitchboardDAO;

import com.example.mts.connectedEquipment.data.dao.SwitchboardModelsDAO;

import com.example.mts.connectedEquipment.domain.entity.Building;

import com.example.mts.connectedEquipment.domain.entity.Cable;

import com.example.mts.connectedEquipment.domain.entity.ConnectedEquipment;

import com.example.mts.connectedEquipment.domain.entity.Switchboard;

import com.example.mts.connectedEquipment.domain.entity.SwitchboardModel;

import java.sql.SQLException;

import java.util.List;

import javax.inject.Inject;

/\*\*

\* Реализация источника данных подключённого оборудования.

\*/

public class ConnectedEquipmentDataSourceImpl implements ConnectedEquipmentDataSource {

/\*\*

\* DAO для работы с подключённым оборудованием.

\*/

private ConnectedEquipmentDAO connectedEquipmentDAO;

/\*\*

\* DAO для доступа к моделям кабелей.

\*/

private CableDAO cableDAO;

/\*\*

\* DAO для доступа к зданиям.

\*/

private BuildingDAO buildingDAO;

/\*\*

\* DAO для доступа к моделям электрических щитков.

\*/

private SwitchboardModelsDAO switchboardModelsDAO;

/\*\*

\* DAO для доступа к электрическим щиткам.

\*/

private SwitchboardDAO switchboardDAO;

/\*\*

\* Конструктор класса ConnectedEquipmentDataSourceImpl.

\* @param connectedEquipmentDAO DAO для доступа к подключённому оборудованию.

\* @param cableDAO DAO для доступа к моделям кабелей.

\* @param buildingDAO DAO для доступа к зданиям.

\* @param switchboardModelsDAO DAO для доступа к моделям электрических щитков.

\* @param switchboardDAO DAO для доступа к электрическим щиткам.

\*/

@Inject

public ConnectedEquipmentDataSourceImpl(ConnectedEquipmentDAO connectedEquipmentDAO,

CableDAO cableDAO,

BuildingDAO buildingDAO,

SwitchboardModelsDAO switchboardModelsDAO,

SwitchboardDAO switchboardDAO) {

this.connectedEquipmentDAO = connectedEquipmentDAO;

this.cableDAO = cableDAO;

this.buildingDAO = buildingDAO;

this.switchboardModelsDAO = switchboardModelsDAO;

this.switchboardDAO = switchboardDAO;

}

@Override

public List<ConnectedEquipment> getConnectedEquipment() throws SQLException {

return connectedEquipmentDAO.queryForAll();

}

@Override

public void deleteConnectedEquipment(int id) throws SQLException {

connectedEquipmentDAO.deleteById(id);

}

@Override

public void createConnectedEquipment(ConnectedEquipment connectedEquipment) throws SQLException {

connectedEquipmentDAO.create(connectedEquipment);

}

@Override

public void updateConnectedEquipment(ConnectedEquipment connectedEquipment) throws SQLException {

connectedEquipmentDAO.update(connectedEquipment);

switchboardDAO.update(connectedEquipment.getSwitchboard());

}

@Override

public List<Cable> getCables() throws SQLException {

return cableDAO.queryForAll();

}

@Override

public List<Building> getBuildings() throws SQLException {

List<Building> buildings = buildingDAO.queryForAll();

for (Building building: buildings) {

for (Switchboard switchboard: building.getSwitchboardList()) {

SwitchboardModel model = switchboardModelsDAO.queryForId(switchboard.getModel().getId());

switchboard.setModel(model);

}

}

return buildings;

}

}

Далее был разработан слой бизнес-логики. Чаще всего UseCase классы возвращают либо какие-то данные, либо сообщение о том, что действие успешно выполнено, поэтому было разработано два базовых класса: BaseSingleUseCase и BaseCompletableUseCase, которые по выполнении действия возвращают объекты Single и Completable из библиотеки RxJava соответственно. Код класса BaseSingleUseCase:

package com.example.mts.base;

import io.reactivex.Scheduler;

import io.reactivex.Single;

/\*\*

\* Базовый UseCase для действий, возвращающих в качестве результата Single.

\* @param <T> репозиторий для выполнения действия.

\* @param <V> тип данных, который оборачивается в Single.

\*/

public abstract class BaseSingleUseCase<T, V> {

/\*\*

\* Репозиторий для выполнения действия.

\*/

protected T repository;

/\*\*

\* Планировщик для выполнения действия.

\*/

protected Scheduler executorScheduler;

/\*\*

\* Конструктор класса BaseSingleUseCase.

\* @param repository репозиторий для выполения действия.

\* @param executorScheduler планировщик для выполнения действия.

\*/

public BaseSingleUseCase(T repository, Scheduler executorScheduler) {

this.repository = repository;

this.executorScheduler = executorScheduler;

}

/\*\*

\* Выполнение usecase.

\* @return результат выполнения действия.

\*/

public abstract Single<V> execute();

}

Код класса BaseCompletableUseCase:

package com.example.mts.base;

import io.reactivex.Completable;

import io.reactivex.Scheduler;

/\*\*

\* Базовый UseCase для действий, возвращающих в качестве результата Completable

\* @param <T> репозиторий для выполнения действия.

\* @param <V> параметры для выполения действия.

\*/

public abstract class BaseCompletableUseCase<T, V> {

/\*\*

\* Репозиторий для выполнения действия.

\*/

protected T repository;

/\*\*

\* Планировщик для выполнения действия.

\*/

protected Scheduler executorScheduler;

/\*\*

\* Конструктор класса BaseSingleUseCase.

\* @param repository репозиторий для выполения действия.

\* @param executorScheduler планировщик для выполнения действия.

\*/

public BaseCompletableUseCase(T repository, Scheduler executorScheduler) {

this.repository = repository;

this.executorScheduler = executorScheduler;

}

/\*\*

\* Выполнение usecase.

\* @param param параметр для выполнения действия.

\* @return результат выполнения действия.

\*/

public abstract Completable execute(V param);

}

В качестве примера класса унаследованного от BaseSingleUseCase можно привести GetCablesUseCase, который извлекает список кабелей. Код класса:

package com.example.mts.connectedEquipment.domain.interactor;

import com.example.mts.base.BaseSingleUseCase;

import com.example.mts.connectedEquipment.domain.entity.Cable;

import com.example.mts.connectedEquipment.domain.repository.ConnectedEquipmentRepository;

import java.util.List;

import javax.inject.Inject;

import io.reactivex.Scheduler;

import io.reactivex.Single;

import io.reactivex.android.schedulers.AndroidSchedulers;

/\*\*

\* UseСase-класс получения моделей кабелей.

\*/

public class GetCablesUseCase extends BaseSingleUseCase<ConnectedEquipmentRepository, List<Cable>> {

/\*\*

\* Конструктор класса GetCablesUseCase.

\* @param connectedEquipmentRepository репозиторий работы с оборудованием.

\* @param executorScheduler планировщик для извлечения записей.

\*/

@Inject

public GetCablesUseCase(ConnectedEquipmentRepository connectedEquipmentRepository, Scheduler executorScheduler) {

super(connectedEquipmentRepository, executorScheduler);

}

/\*\*

\* Выполнение извлечения моделей кабелей.

\* @return модели кабелей.

\*/

public Single<List<Cable>> execute() {

return repository

.getCables()

.subscribeOn(executorScheduler)

.observeOn(AndroidSchedulers.mainThread());

}

}

Данный класс использует репозиторий ConnectedEquipmentRepository, реализация которого – ConnectedEquipmentRepositoryImpl, использует созданный раннее источник данных ConnectedEquipmentDataSource. Код класса ConnectedEquipmentRepositoryImpl:

package com.example.mts.connectedEquipment.domain.repository;

import com.example.mts.connectedEquipment.data.database.ConnectedEquipmentDataSource;

import com.example.mts.connectedEquipment.domain.entity.Building;

import com.example.mts.connectedEquipment.domain.entity.Cable;

import com.example.mts.connectedEquipment.domain.entity.ConnectedEquipment;

import com.example.mts.connectedEquipment.domain.entity.Switchboard;

import java.sql.SQLException;

import java.util.List;

import javax.inject.Inject;

import io.reactivex.Completable;

import io.reactivex.Single;

/\*\*

\* Реализация репозитория работы с подключённым оборудованием.

\*/

public class ConnectedEquipmentRepositoryImpl implements ConnectedEquipmentRepository {

/\*\*

\* Источник данных репозитория.

\*/

private ConnectedEquipmentDataSource connectedEquipmentDataSource;

/\*\*

\* Конструктор класса ConnectedEquipmentRepositoryImpl по умолчанию.

\*/

public ConnectedEquipmentRepositoryImpl() {

}

/\*\*

\* Конструктор класса ConnectedEquipmentRepositoryImpl.

\*

\* @param connectedEquipmentDataSource Источник данных.

\*/

@Inject

public ConnectedEquipmentRepositoryImpl(ConnectedEquipmentDataSource connectedEquipmentDataSource) {

this.connectedEquipmentDataSource = connectedEquipmentDataSource;

}

@Override

public Single<List<ConnectedEquipment>> getConnectedEquipment() {

return Single.create(emitter -> {

try {

emitter.onSuccess(connectedEquipmentDataSource.getConnectedEquipment());

} catch (SQLException e) {

emitter.onError(e);

}

});

}

@Override

public Completable deleteConnectedEquipment(int id) {

return Completable.create(emitter -> {

try {

connectedEquipmentDataSource.deleteConnectedEquipment(id);

emitter.onComplete();

} catch (SQLException e) {

emitter.onError(e);

}

});

}

@Override

public Completable createConnectedEquipment(ConnectedEquipment connectedEquipment) {

return Completable.create(emitter -> {

try {

connectedEquipmentDataSource.createConnectedEquipment(connectedEquipment);

emitter.onComplete();

} catch (SQLException e) {

emitter.onError(e);

}

});

}

@Override

public Completable updateConnectedEquipment(ConnectedEquipment connectedEquipment) {

return Completable.create(emitter -> {

try {

connectedEquipmentDataSource.updateConnectedEquipment(connectedEquipment);

emitter.onComplete();

} catch (SQLException e) {

emitter.onError(e);

}

});

}

@Override

public Single<List<Cable>> getCables() {

return Single.create(emitter -> {

try {

emitter.onSuccess(connectedEquipmentDataSource.getCables());

} catch (SQLException e) {

emitter.onError(e);

}

});

}

@Override

public Single<List<Building>> getBuildings() {

return Single.create(emitter -> {

try {

emitter.onSuccess(connectedEquipmentDataSource.getBuildings());

} catch (SQLException e) {

emitter.onError(e);

}

});

}

}

В итоге взаимодействие активности и UseCase классов происходит благодаря классам представителям. Класс представитель ConnectedEquipmentListPresenter для активности списка подключённого оборудования:

package com.example.mts.connectedEquipment.presentation.presenter;

import com.example.mts.connectedEquipment.domain.entity.ConnectedEquipment;

import com.example.mts.connectedEquipment.domain.interactor.DeleteConnectedEquipmentUseCase;

import com.example.mts.connectedEquipment.domain.interactor.GetConnectedEquipmentUseCase;

import com.example.mts.connectedEquipment.presentation.view.ConnectedEquipmentListView;

import java.util.ArrayList;

import java.util.List;

import javax.inject.Inject;

import io.reactivex.CompletableObserver;

import io.reactivex.SingleObserver;

import io.reactivex.disposables.Disposable;

/\*\*

\* Класс-представитель списка модуля подключённого оборудования.

\*/

public class ConnectedEquipmentListPresenter {

/\*\*

\* Представление активности.

\*/

ConnectedEquipmentListView view;

/\*\*

\* UseСase загрузки списка оборудования из БД.

\*/

GetConnectedEquipmentUseCase getConnectedEquipmentUseCase;

/\*\*

\* UseCase удаления подключённого оборудования.

\*/

DeleteConnectedEquipmentUseCase deleteConnectedEquipmentUseCase;

/\*\*

\* Список подключённого оборудования.

\*/

private List<ConnectedEquipment> connectedEquipments = new ArrayList<>();

/\*\*

\* Текущая запись.

\*/

private ConnectedEquipment currentConnectedEquipment;

/\*\*

\* Показывает нужно ли обновлять данные в представлении.

\*/

private boolean isViewNeedToReload = true;

/\*\*

\* Возвращает список подключённого оборудования.

\* @return список подключённого оборудования.

\*/

public List<ConnectedEquipment> getConnectedEquipment() {

return connectedEquipments;

}

/\*\*

\* Устанавливает список подключённого оборудования.

\* @param connectedEquipments новый список подключённого оборудования.

\*/

public void setConnectedEquipments(List<ConnectedEquipment> connectedEquipments) {

this.connectedEquipments = connectedEquipments;

}

/\*\*

\* Устанавливает свойство необходимости обновлять представление при возникновении события onResume().

\* @param viewNeedToReload свойство необходимости обновлять представление.

\*/

public void setViewNeedToReload(boolean viewNeedToReload) {

isViewNeedToReload = viewNeedToReload;

}

/\*\*

\* Конструктор по умолчанию.

\*/

public ConnectedEquipmentListPresenter() {}

/\*\*

\* Конструктор класса ConnectedEquipmentListPresenter.

\* @param view представление списка подключённого оборудования.

\* @param getConnectedEquipmentUseCase UseCase загрузки списка подключённого оборудования.

\* @param deleteConnectedEquipmentUseCase UseCase удаления подключённого оборудования.

\*/

@Inject

public ConnectedEquipmentListPresenter(ConnectedEquipmentListView view,

GetConnectedEquipmentUseCase getConnectedEquipmentUseCase,

DeleteConnectedEquipmentUseCase deleteConnectedEquipmentUseCase) {

this.view = view;

this.getConnectedEquipmentUseCase = getConnectedEquipmentUseCase;

this.deleteConnectedEquipmentUseCase = deleteConnectedEquipmentUseCase;

}

/\*\*

\* Заполняет список подключённого оборудования, если необходимо.

\*/

public void fillConnectedEquipment() {

if (isViewNeedToReload) {

loadConnectedEquipment();

}

}

/\*\*

\* Загружает список подключённого оборудования.

\*/

private void loadConnectedEquipment() {

getConnectedEquipmentUseCase.execute().subscribe(new ConnectedEquipmentListPresenter.LoadConnectedEquipmentObserver());

}

/\*\*

\* Класс-наблюдатель, обновляющий список оборудования в представлении после успешной выгрузки.

\*/

private class LoadConnectedEquipmentObserver implements SingleObserver<List<ConnectedEquipment>> {

@Override

public void onSubscribe(Disposable d) {}

@Override

public void onSuccess(List<ConnectedEquipment> connectedEquipments) {

ConnectedEquipmentListPresenter.this.connectedEquipments = connectedEquipments;

updateView();

}

@Override

public void onError(Throwable e) {

throw new RuntimeException("Возникла ошибка при получении списка подключённого оборудования.");

}

}

/\*\*

\* Обновляет список подключённого оборудования в представлении.

\*/

private void updateView() {

view.fillConnectedEquipmentList();

}

/\*\*

\* Возникает при нажатии на элемент списка подключённого оборудования.

\* @param connectedEquipment нажатый элемент.

\*/

public void onEquipmentClick(ConnectedEquipment connectedEquipment) {

isViewNeedToReload = false;

view.openConnectedEquipmentItem(connectedEquipment);

}

/\*\*

\* Возникает при долгом нажатии на элемент списка подключённого оборудования.

\* @param connectedEquipment нажатый элемент.

\*/

public void onEquipmentLongClick(ConnectedEquipment connectedEquipment) {

currentConnectedEquipment = connectedEquipment;

view.openDeleteConnectedEquipmentDialog();

}

/\*\*

\* Возникает при нажатии на кнопку добавления.

\*/

public void onbtnAddClick() {

view.createConnectedEquipmentItem();

}

/\*\*

\* Удаляет текущую запись о подключённом оборудовании.

\*/

public void deleteConnectedEquipment() {

deleteConnectedEquipmentUseCase.execute(new DeleteConnectedEquipmentUseCase.Param(currentConnectedEquipment.getId()))

.subscribe(new DeleteConnectedEquipmentObserver());

}

/\*\*

\* Класс-наблюдатель, обновляющий список оборудования в представлении после удаления записи.

\*/

private class DeleteConnectedEquipmentObserver implements CompletableObserver {

@Override

public void onSubscribe(Disposable d) {}

@Override

public void onComplete() {

loadConnectedEquipment();

}

@Override

public void onError(Throwable e) {

throw new RuntimeException("Возникла ошибка при удалении записи.");

}

}

}

Также следует отметить, что для заполнения списков данными в активностях следует создавать классы адаптеры. Например, был создан класс адаптер для списка подключённого оборудования ConnectedEquipmentAdapter. Код класса:

package com.example.mts.connectedEquipment.presentation.view.adapters;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.TextView;

import androidx.annotation.NonNull;

import androidx.cardview.widget.CardView;

import androidx.recyclerview.widget.RecyclerView;

import com.example.mts.R;

import com.example.mts.connectedEquipment.domain.entity.ConnectedEquipment;

import java.util.List;

/\*\*

\* Адаптер для списка подключённого оборудования.

\*/

public class ConnectedEquipmentAdapter extends RecyclerView.Adapter<ConnectedEquipmentViewHolder> {

/\*\*

\* Интерфейс слушателя события нажатия на элемент списка.

\*/

public interface OnClickListener {

/\*\*

\* Возникает при нажатии на элемент списка.

\* @param connectedEquipment элемент, на который произошло нажатие.

\* @param position позиция нажатого элемента.

\*/

void onClick(ConnectedEquipment connectedEquipment, int position);

/\*\*

\* Возникает при долгом нажатии на элемент списка.

\* @param connectedEquipment элемент, на который произошло нажатие.

\* @param position позиция нажатого элемента.

\* @return

\*/

void onLongClick(ConnectedEquipment connectedEquipment, int position);

}

/\*\*

\* Слушатель события нажатия на элемент списка.

\*/

private final OnClickListener onClickListener;

/\*\*

\* Коллекция, для которой создан адаптер.

\*/

private List<ConnectedEquipment> connectedEquipments;

/\*\*

\* Конструктор класса ConnectedEquipmentAdapter.

\* @param connectedEquipments список подключённого оборудования.

\* @param onClickListener слушатель события onClick.

\*/

public ConnectedEquipmentAdapter(List<ConnectedEquipment> connectedEquipments, OnClickListener onClickListener) {

this.connectedEquipments = connectedEquipments;

this.onClickListener = onClickListener;

}

@NonNull

@Override

public ConnectedEquipmentViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {

CardView cardView = (CardView) LayoutInflater.from(parent.getContext())

.inflate(R.layout.card\_view\_connected\_equipment, parent, false);

return new ConnectedEquipmentViewHolder(cardView);

}

@Override

public void onBindViewHolder(@NonNull ConnectedEquipmentViewHolder holder, int position) {

ConnectedEquipment connectedEquipment = connectedEquipments.get(position);

CardView cardView = holder.getCardView();

setCardTitle(cardView, connectedEquipment);

setCardSubtitle(cardView, connectedEquipment);

holder.itemView.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

onClickListener.onClick(connectedEquipment, position);

}

});

holder.itemView.setOnLongClickListener(new View.OnLongClickListener() {

@Override

public boolean onLongClick(View v) {

onClickListener.onLongClick(connectedEquipment, position);

return true;

}

});

}

/\*\*

\* Устанавливает заголовок в строке о подключённом оборудовании.

\* @param cardView представление строки.

\* @param connectedEquipment исходная запись о подключённом оборудовании.

\*/

private void setCardTitle(CardView cardView, ConnectedEquipment connectedEquipment) {

String title = connectedEquipment.getSwitchboard().getBuilding().getAddress()

+ ".\nИнв. номер: " + connectedEquipment.getSwitchboard().getInventoryNumber()

+ ". Модель: " + connectedEquipment.getSwitchboard().getModel().getName()

+ ".\nПорт: " + connectedEquipment.getPortNumber();

TextView textView = (TextView) cardView.findViewById(R.id.title);

textView.setText(title);

}

/\*\*

\* Устанавливает подзаголовок в строке о подключённом оборудовании.

\* @param cardView представление строки.

\* @param connectedEquipment исходная запись о подключённом оборудовании.

\*/

private void setCardSubtitle(CardView cardView, ConnectedEquipment connectedEquipment) {

String subtitle = "IP: " + connectedEquipment.getIp() + "\nMAC: " + connectedEquipment.getMac();

TextView textView = (TextView) cardView.findViewById(R.id.subtitle);

textView.setText(subtitle);

}

@Override

public int getItemCount() {

return connectedEquipments.size();

}

}

Для проверки корректности IP и MAC адресов был создан класс Validator, также для проверки его работы были написаны модульные тесты, все из которых успешно прошли. Код класса Validator:

package com.example.mts.utils;

/\*\*

\* Класс для валидации даннных.

\*/

public class Validator {

/\*\*

\* Возвращает истину, если переданная строка является ip-адресом.

\* @param ip проверяемая строка.

\* @return истина, если переданная строка является ip-адресом.

\*/

public static boolean isIpValid(String ip) {

return ip != null && ip.matches("^((25[0-5]|(2[0-4]|1\\d|[1-9]|)\\d)\\.?\\b){4}$");

}

/\*\*

\* Возвращает истину, если переданная строка является MAC-адресом.

\* @param mac проверяемая строка.

\* @return истина, если переданная строка является MAC-адресом.

\*/

public static boolean isMacValid(String mac) {

return mac != null && mac.matches("^([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})$");

}

}

В итоге было разработано ещё некоторое множество кода, обеспечивающего функционал приложения. По окончанию разработки необходимо протестировать разработанный модуль.

3 ОТЛАДКА, ТЕСТИРОВАНИЕ И ОПТИМИЗАЦИЯ ПРОГРАММНЫХ МОДУЛЕЙ

Тестирование и отладка приложения проводилось с использованием виртуального устройства в Android Studio.

3.1 Тестирование логики программы

Тестирование функциональной части программы представлено в приложении 2. Были протестированы основные возможности программного модуля, указанные в требованиях. В итоге ошибки, приводящие к завершению работы программы, не обнаружены, следовательно разработанный модуль является надёжным и качественным.

3.2 Тестирование интерфейса пользователя

Тестирование интерфейса программы представлено в приложении 3. Тестирование не выявило некорректного поведения приложения. Внешний вид графических элементов приложения не противоречит требованиям. Что говорит об успешной разработке программного модуля.

ВЫВОД

Итак, целью производственной практики была разработка модуля мобильной торговой системы для учёта подключённого оборудования в зданиях. Для достижения этой цели были изучены структура организации и её предметная область, проанализированы требования к разрабатываемому модулю. Была проведена разработка графического интерфейса и функциональной части модуля. Были написаны комментарии к коду, а также несколько модульных тестов. Затем разработанный модуль был протестирован. Дефектов во время тестирования не обнаружено. Следовательно цель практики достигнута.

Разработанный модуль можно в дальнейшем усовершенствовать, например сделать отправку данных из локальной базы на удалённый сервер. В ходе прохождения практики было освоено множество новых технологий, и я познакомился с разработкой под платформу Android.

ПРИЛОЖЕНИЕ 1

Требования к разрабатываемому программному обеспечению.

Описание проблемы:

В сооружениях торговой сети "Макси" есть оборудование, которое подключается к электросети через распределительные щиты. Бывают разные модели щитов, которые могут иметь разное число портов для подключения. Оборудование подключается к щитам с помощью кабелей. Необходима возможность работы со сведениями о подключённом оборудованием.

Суть работы:

Разработка программного модуля мобильной торговой системы, в котором будут возможности просматривать, добавлять, изменять и удалять данные о подключённом оборудовании к сооружениям.

Цель работы:

* Обеспечение возможности быстрого доступа к сведениям о подключённом оборудовании;
* Ускорение сбора данных благодаря использованию мобильной торговой системы.

Системные характеристики

1. Целевая платформа приложения – Android версии 7.1 и более.
2. Используемый язык программирования – Java 8.
3. Минимальная версия SDK – 25.

Пользовательские требования

1. Модуль назначен для использования сотрудниками компании «Макси», связанными с электрооборудованием.
2. В модуле должны быть возможности просматривать, добавлять, изменять и удалять данные о подключённом оборудовании с соответствие с рисунком 5.
3. Открытие модуля.
   1. Модуль должен отображаться в списке модулей мобильной торговой системы и иметь заголовок «Подключённое оборудование».
   2. Открытие осуществляется при нажатии на элемент списка модулей.
4. Структура модуля.
   1. При открытии модуля должна открываться активность со списком подключённого оборудования (ДС-1) и кнопкой «Добавить».

Изображение выглядит как текст, круг, диаграмма, снимок экрана

Автоматически созданное описание

Рисунок 5 – UseCase диаграмма модуля о подключённом оборудовании.

* 1. При нажатии на элемент списка должна открываться активность изменения записи (ДС-2).
  2. При нажатии на кнопку «Добавить» должна открываться активность добавления записи (ДС-2).
  3. При долгом нажатии на элемент списка должно появляться диалоговое окно для подтверждения удаления записи (ДС-3).

Бизнес-правила

1. Цветовая гамма модуля должна соответствовать используемой цветовой гамме мобильной торговой системы. Допустимые цвета: белый, чёрный, серый, зелёный (#3c9e2d, #5eba2f), оранжевый (#ec7404, #fdbf2e).

Детальные спецификации

1. Список подключённого оборудования должен содержать такие данные, как адрес здания, инвентарный номер распределительного щита, модель распределительного щита, порт в который подключено оборудование, IP и MAC адреса подключённого оборудования.
2. Активность работы с записью должна иметь два режима: вставка новой и изменение существующей записей.
   1. Активность работы с записью должна содержать следующие данные для редактирования: адрес здания в виде выпадающего списка; данные об инвентарном номере и модели распределительного щита в виде выпадающего списка, порт распределительного щита в виде поля ввода; модель кабеля в виде выпадающего списка; IP и MAC адреса в виде полей ввода.
   2. Выпадающий список распределительных щитов должен быть заполнен теми щитам, которые относятся к выбранному зданию.
   3. В активности работы с записью должна быть кнопка «Сохранить».
      1. Перед сохранением должна происходить проверка корректности IP и MAC адресов, порта, к которому подключено оборудование.
      2. Если данные не корректны, должно появиться диалоговое окно со списков сообщений об ошибках.
   4. При попытке сохранить запись со значениями порта распределительного щита, к которому уже подключено оборудование должно возникать диалоговое окно «Порт уже занят».
      1. После успешного сохранения активность закрывается, а список подключённого оборудования обновляется.
3. Диалоговое окно для подтверждения удаления записи должно содержать сообщение «Вы точно хотите удалить запись?» и кнопки «Да», «Нет».
   1. При нажатии на кнопку «Нет» окно закрывается, запись не удаляется.
   2. При нажатии на кнопку «Да» окно закрывается, запись удаляется, а список записей обновляется.

Требования к данным

1. Данные должны храниться в СУБД SqLite.

ПРИЛОЖЕНИЕ 2

Тестирование логики программы.

Тестирование функциональных возможностей программы представлено в таблице 1.

Таблица 1. Тестирование логики программы.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Описание | Действие | Ожидаемый результат | Фактический результат | Статус |
| 1 | Открытие модуля подключённого оборудования. | 1. Запустите приложение.  2. Перейдите к списку разделов.  3. Нажмите по строке «Подключённое оборудование». | Открывается список подключённого оборудования. | Открывается список подключённого оборудования. | Пройден |
| 2 | Открытие записи на изменение. | 1. Откройте список подключённого оборудования.  2. Нажмите на любую запись. | Открывается активность изменения записи с заполненными полями. | Открывается активность изменения записи с заполненными полями. | Пройден |
| 3 | Изменение записи. | 1. Откройте на изменение любую запись.  2. Поменяйте IP-адрес.  3. Нажмите кнопку «Сохранить». | Активность изменения записи закрывается, в списке оборудования данные записи соответствуют изменениям. | Активность изменения записи закрывается, в списке оборудования данные записи соответствуют изменениям. | Пройден |
| 4 | Открытие окна на сохранение. | 1. Откройте список подключённого оборудования.  2. Нажмите кнопку «Добавить». | Открывается окно добавления записи. | Открывается окно добавления записи. | Пройден |
| 5 | Добавление записи | 1. Откройте список подключённого оборудования.  2. Нажмите кнопку «Добавить».  3. Выберите адрес, распределительный щит, модель кабеля.  4. Введите порт щита, который ещё не используется, любую длину кабеля и любые корректные IP и MAC адреса.  5. Нажмите «Сохранить». | Активность изменения записи закрывается, в списке оборудования появляется запись с введёнными данными. | Активность изменения записи закрывается, в списке оборудования появляется запись с введёнными данными. | Пройден |

Продолжение таблицы 1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 6 | Попытка сохранить с незаполненными IP, MAC и портом. | 1. Откройте список подключённого оборудования.  2. Нажмите кнопку «Добавить».  3. Нажмите «Сохранить». | Появится окно с сообщением «Номер порта должен быть больше 0; Неверный IP-адрес; Неверный MAC-адрес;». | Появилось окно с сообщением «Номер порта должен быть больше 0; Неверный IP-адрес; Неверный MAC-адрес;». | Пройден |
| 7 | Попытка сохранить с неверными IP, MAC и портом. | 1. Откройте список подключённого оборудования.  2. Нажмите кнопку «Добавить».  3. Введите некорректные IP и MAC адреса.  4. Введите 99999 в поле порта.  5. Нажмите «Сохранить». | Появится окно с сообщением «Номер порта больше позволенного. Максимальный порт модели щитка - ХХ; Неверный IP-адрес; Неверный MAC-адрес;», где ХХ – любое число. | Появилось окно с сообщением «Номер порта больше позволенного. Максимальный порт модели щитка - ХХ; Неверный IP-адрес; Неверный MAC-адрес;», где ХХ – любое число. | Пройден |
| 8 | Сохранение записи с занятым портом. | 1. Откройте список подключённого оборудования.  2. Запомните инвентарный номер щита и порт у любой записи.  3. Попытайтесь создать запись с такими же значениями. | Появится окно с сообщением «Порт уже занят». | Появилось окно с сообщением «Порт уже занят». | Пройден |
| 9 | Удаление записи. | 1. Откройте список подключённого оборудования.  2. Нажмите и удерживайте палец на любой записи.  3. Появится окно подтверждения удаления.  4. Нажмите «Да». | Запись исчезла из списка. | Запись исчезла из списка. | Пройден |
| 10 | Неподтверждённое удаление записи. | 1. Откройте список подключённого оборудования.  2. Нажмите и удерживайте палец на любой записи.  3. Появится окно подтверждения удаления.  4. Нажмите «Нет». | Запись осталась. | Запись осталась. | Пройден |
| 11 | Сохранение изменений в базе данных. | 1. Измените любую запись, добавьте и удалите любые записи.  2. Закройте приложение полностью.  3. Войдите в приложение. | Изменения сохранились. | Изменения сохранились. | Пройден |

ПРИЛОЖЕНИЕ 3

Тестирование интерфейса пользователя.

Тестирование интерфейса программного модуля представлено в таблице 2.

Таблица 2. Тестирование интерфейса пользователя.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Описание | Действие | Ожидаемый результат | Фактический результат | Статус |
| 1 | Присутствие необходимых компонентов в активности списка подключённого оборудования. | 1. Запустите приложение.  2. Перейдите к списку разделов.  3. Нажмите по строке «Подключённое оборудование». | Заголовок открывшейся активности имеет текст «Подключённое оборудование». Имеется список подключённого оборудования и кнопка «Сохранить». | Заголовок открывшейся активности имеет текст «Подключённое оборудование». Имеется список подключённого оборудования и кнопка «Сохранить». | Пройден |
| 2 | Присутствие необходимых компонентов ввода в активности изменения подключённого оборудования. | 1. Запустите приложение.  2. Перейдите к списку разделов.  3. Нажмите по строке «Подключённое оборудование».  4. Нажмите на любую запись. | В открывшейся активности есть выпадающий список со зданиями, выпадающий список распределительных щитов, поле ввода порта, выпадающий список кабелей, поле ввода длины кабеля, поля ввода IP и MAC-адресов. | В открывшейся активности есть выпадающий список со зданиями, выпадающий список распределительных щитов, поле ввода порта, выпадающий список кабелей, поле ввода длины кабеля, поля ввода IP и MAC-адресов. | Пройден |
| 3 | Присутствие кнопки «Сохранить» в активности изменения подключённого оборудования. | 1. Запустите приложение.  2. Перейдите к списку разделов.  3. Нажмите по строке «Подключённое оборудование».  4. Нажмите на любую запись. | В открывшейся активности есть кнопка «Сохранить». | В открывшейся активности есть кнопка «Сохранить». | Пройден |
| 4 | Неизменяемость элементов списка модулей при перезагрузке активности. | 1. Откройте список модулей приложения.  2. Нажмите кнопку «Домой» в панели навигации устройства.  3. Откройте приложение снова. | Список модулей приложения не изменился. Новых записей не появилось. | Список модулей приложения не изменился. Новых записей не появилось. | Пройден |

Продолжение таблицы 2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | Неизменяемость элементов списка подключённого оборудования при перезагрузке активности. | 1. Откройте список подключённого оборудования.  2. Нажмите кнопку «Домой» в панели навигации устройства.  3. Откройте приложение снова. | Список подключённого оборудования не изменился. Новых записей не появилось. | Список подключённого оборудования не изменился. Новых записей не появилось. | Пройден |
| 6 | Корректность цветов компонентов активности изменения подключённого оборудования. | 1. Запустите приложение.  2. Перейдите к списку разделов.  3. Нажмите по строке «Подключённое оборудование».  4. Нажмите на любую запись. | Компоненты активности не используют цвета, кроме перечисленных: зелёный, белый, оранжевый, чёрный, серый. | Компоненты активности не используют цвета, кроме перечисленных: зелёный, белый, оранжевый, чёрный, серый. | Пройден |
| 7 | Корректность цветов компонентов активности списка подключённого оборудования. | 1. Запустите приложение.  2. Перейдите к списку разделов.  3. Нажмите по строке «Подключённое оборудование». | Компоненты активности не используют цвета, кроме перечисленных: зелёный, белый, оранжевый, чёрный, серый. | Компоненты активности не используют цвета, кроме перечисленных: зелёный, белый, оранжевый, чёрный, серый. | Пройден |