МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королёва»
(Самарский университет)

Институт информатики, математики и электроники

Факультет информатики

Кафедра информационных систем и технологий

ОТЧЕТ

по лабораторной работе №5 по дисциплине

«Менеджмент разработки программного обеспечения»

Выполнили:

Гуреев М.А., гр. 6222-090401D

Елфимов А.Г., гр. 6222-090401D

Филатов В.В., гр. 6222-090401D

Проверил:

доцент каф. ИСТ, к.т.н. Крупец Н.Г.

Самара 2020

ЦЕЛЬ РАБОТЫ

Разработать мобильное приложение для контролёра энергосбытовой компании.

КРАТКАЯ ХАРАКТЕРИСТИКА ОБЪЕКТА АВТОМАТИЗАЦИИ

Объектом автоматизации является процесс контрольного снятия показаний с приборов учёта электроэнергии. Он состоит в том, что энегосбытовая организация вынуждена проводить контрольные проверки показаний приборов учёта, для своевременно устранения неисправностей и недопущения получения некорректных показаний.

Мобильное приложение для контролёра

К основным задачам контролёра относятся снятие показаний с приборов учёта, фиксирование неисправностей и отправка данных на сервер. Обеспечим контролёру следующие возможности:

1. Добавление показаний по точке учёта в базу данных, фиксирование состояния прибора.
2. Просмотр информации о профиле.
3. Просмотр назначенных точек учёта.
4. Просмотр информационной рассылки для всех пользователей системы.

Далее представлены скриншоты программы. На рисунке 1 показано окно авторизации пользователя. Рисунок 2 – главное меню программы. На нём есть возможности просмотра назначенных точек учёта (рисунок 3), создать запись о контрольном снятии показаний (рисунок 4), просмотреть информационную рассылку (рисунок 5) и информацию о профиле (рисунок 6).

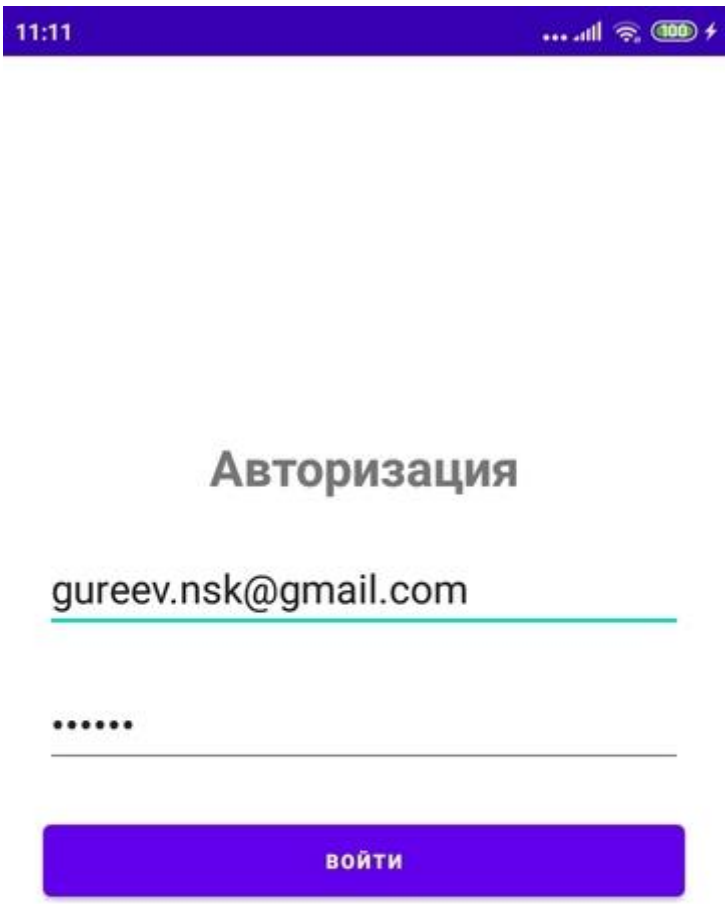

Рисунок 1 – Экран авторизации

Рисунок 2 – Экран главного меню

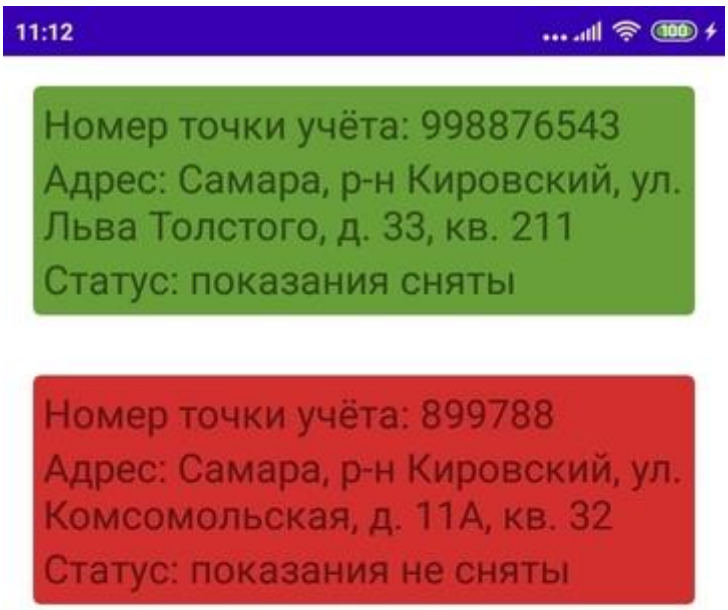

Рисунок 3 – Экран просмотра назначенных точек учёта

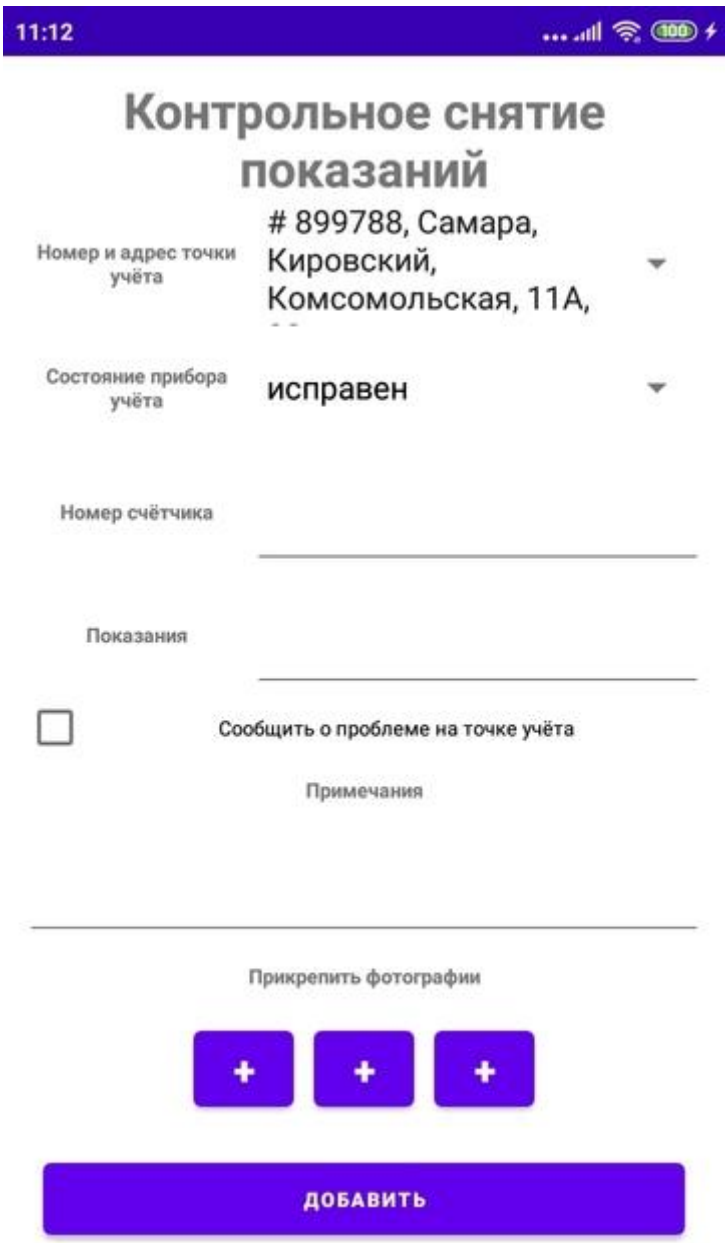

Рисунок 4 – Экран добавления записи о контрольном снятии показаний

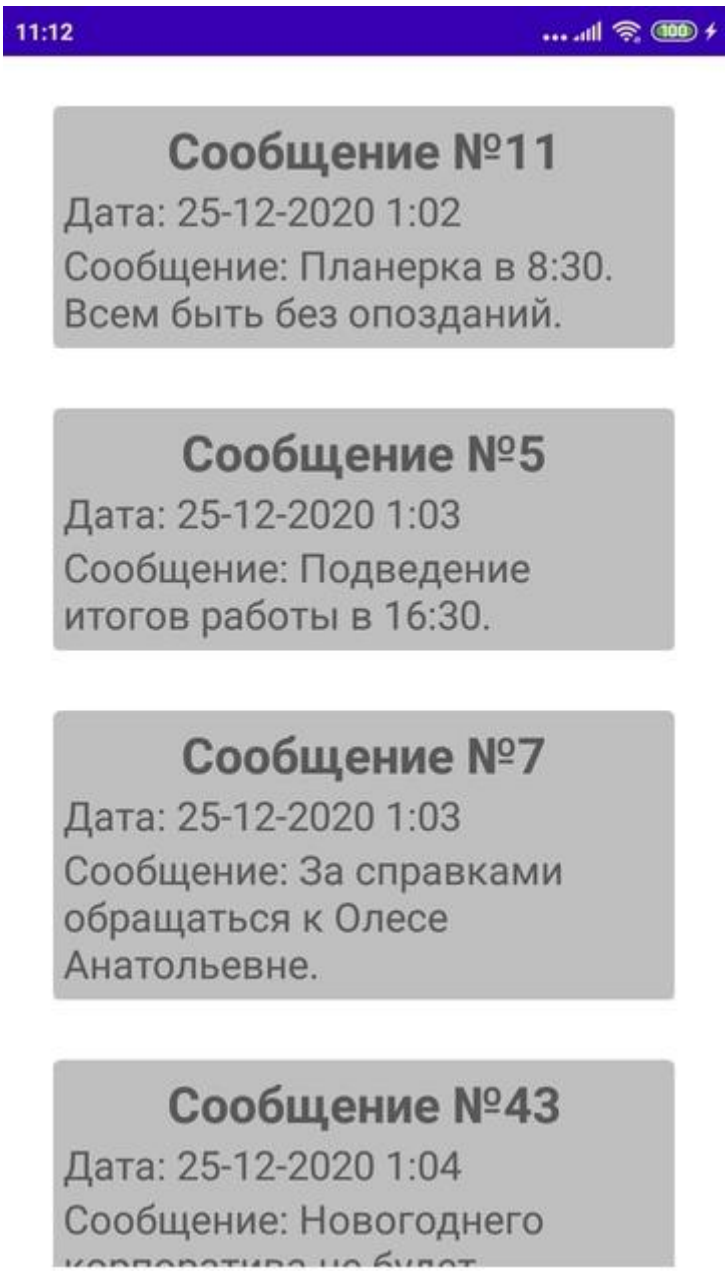

Рисунок 5 – Экран просмотра информационной рассылки

Рисунок 6 – Экран информации о пользователе

Диаграмма переходов экранов приложения

В общем виде структура экранов приложения представлена на рисунке 7. Главным стартовым экраном является экран авторизация пользователя. Далее система определяет тип пользователя и перенаправляет его в соответствующее меню. Функциональные возможности диспетчера были описаны в предыдущей лабораторной работе, в данной работе в разделе выше были описаны возможности для контролёра. Общими экранами являются просмотр информационной рассылки и информации о профиле.

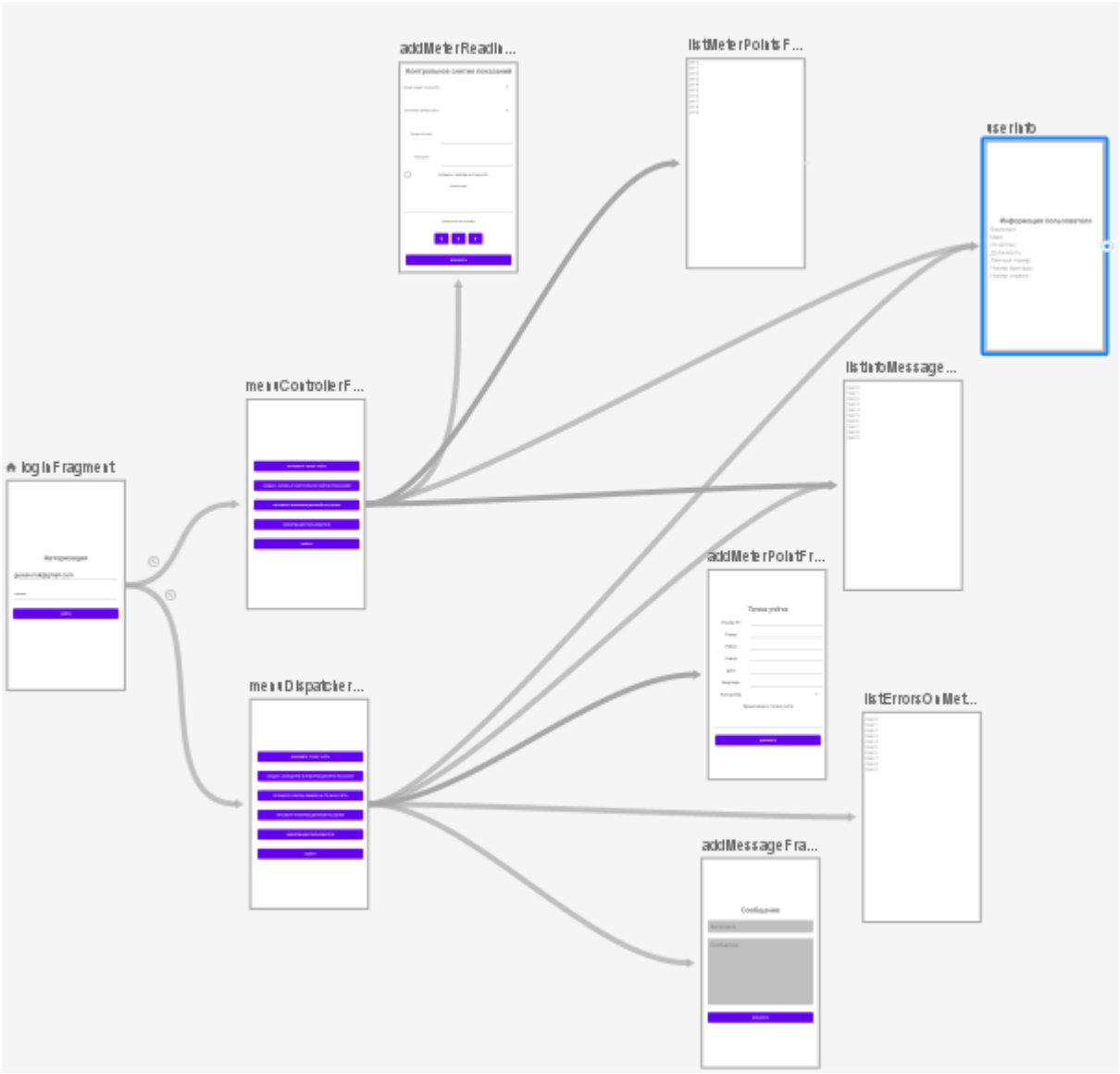

Рисунок 7 – Диаграмма переходов экранов приложения

Приложение 1.

Код основных операций программы.

```
public class AddMeterReadingFragment extends Fragment {

    private List<MeterPoint> meterPointList;
    private List<String> deviceStatusList;


    private Spinner spinnerAddresses;
    private Spinner spinnerMeterStatus;

    private EditText editTextProblemDescription;
    private EditText editTextNumMeterReading;
    private EditText editTextMeterReading;

    private CheckBox isProblemOnMeterPoint;
    private MeterPoint currentMeterPoint;

    private Button buttonAddPhoto1;
    private Button buttonAddPhoto2;
    private Button buttonAddPhoto3;

    private Button buttonAddMeterReadingPoint;

    private NavController navController;
    private AddMeterReadingViewModel mViewModel;

    AlertDialog dialog;

    private static final int REQUEST_CODE_PERMISSION_RECEIVE_CAMERA = 102;

    static final int REQUEST_TAKE_PHOTO = 1;
    private String mCurrentPhotoPath;
    private String photoName = "";
    private Uri photoURI;

    public static AddMeterReadingFragment newInstance() {
        return new AddMeterReadingFragment();
    }

    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup
container,
                             @Nullable Bundle savedInstanceState) {
        navController = Navigation.findNavController(getActivity(),
R.id.nav_host_fragment);
        return inflater.inflate(R.layout.add_meter_reading_fragment, container, false);
    }

    @Override
    public void onActivityCreated(@Nullable Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        mViewModel = new ViewModelProvider(this).get(AddMeterReadingViewModel.class);
        // TODO: Use the ViewModel
    }

    @Override
    public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState)
{
```

```
        super.onViewCreated(view, savedInstanceState);

        checkPermission();

        meterPointList = new ArrayList<>();

        deviceStatusList = new ArrayList<>();
        deviceStatusList.add(Config.DEVICE_STATUS_GOOD);
        deviceStatusList.add(Config.DEVICE_STATUS_BROKEN);
        deviceStatusList.add(Config.DEVICE_STATUS_BAD);

        spinnerAddresses = view.findViewById(R.id.spinner_list_of_addresses);
        spinnerMeterStatus = view.findViewById(R.id.spinner_list_of_meter_status);

        editTextNumMeterReading = view.findViewById(R.id.textField_num_meter_reading);
        editTextMeterReading = view.findViewById(R.id.textField_meter_reading);
        editTextProblemDescription =
view.findViewById(R.id.textField_problem_description);

        isProblemOnMeterPoint = view.findViewById(R.id.checkBox_isProblem);

        buttonAddMeterReadingPoint =
view.findViewById(R.id.button_create_meter_reading);
        buttonAddPhoto1 = view.findViewById(R.id.button_add_photo1);
        buttonAddPhoto2 = view.findViewById(R.id.button_add_photo2);
        buttonAddPhoto3 = view.findViewById(R.id.button_add_photo3);

        buttonAddMeterReadingPoint.setOnClickListener(onClickListener);
        buttonAddPhoto1.setOnClickListener(onClickListener);
        buttonAddPhoto2.setOnClickListener(onClickListener);
        buttonAddPhoto3.setOnClickListener(onClickListener);


        ArrayAdapter<MeterPoint> meterPointArrayAdapter = new
ArrayAdapter<>(getContext(), android.R.layout.simple_selectable_list_item);
        ArrayAdapter<String> deviceStatusArrayAdapter = new
ArrayAdapter<>(getContext(), android.R.layout.simple_spinner_dropdown_item);

        Repository.getInstance(getActivity()).getNetworkService().getMeterPoints();

Repository.getInstance(getActivity()).getNetworkService().getMeterPointsList().observe(
getViewLifecycleOwner(), meterPoints -> {
            meterPointList = filterMeterPoint(meterPoints);
            meterPointArrayAdapter.clear();
            meterPointArrayAdapter.addAll(meterPointList);
            spinnerAddresses.setAdapter(meterPointArrayAdapter);
            if (!meterPointArrayAdapter.isEmpty()) {
                currentMeterPoint = meterPointList.get(0);
                buttonAddPhoto1.setEnabled(true);
                buttonAddPhoto2.setEnabled(true);
                buttonAddPhoto3.setEnabled(true);
            } else {
                buttonAddPhoto1.setEnabled(false);
                buttonAddPhoto2.setEnabled(false);
                buttonAddPhoto3.setEnabled(false);
            }

        });

        deviceStatusArrayAdapter.addAll(deviceStatusList);
        spinnerMeterStatus.setAdapter(deviceStatusArrayAdapter);
```

```
        spinnerAddresses.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener() {
            @Override
            public void onItemSelected(AdapterView<?> parent, View view, int position,
long id) {
                if (!meterPointList.isEmpty()) {
                    currentMeterPoint = meterPointList.get(position);
                }
            }

            @Override
            public void onNothingSelected(AdapterView<?> parent) {

            }
        });

        spinnerMeterStatus.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener() {
            @Override
            public void onItemSelected(AdapterView<?> parent, View view, int position,
long id) {
                if (currentMeterPoint != null && !deviceStatusList.isEmpty()) {
                    currentMeterPoint.setDeviceStatus(deviceStatusList.get(position));
                    if
(deviceStatusList.get(position).equals(Config.DEVICE_STATUS_BAD)) {
                        isProblemOnMeterPoint.setChecked(true);
                    } else {
                        isProblemOnMeterPoint.setChecked(false);
                    }
                }
            }

            @Override
            public void onNothingSelected(AdapterView<?> parent) {

            }
        });


Repository.getInstance(getActivity()).getNetworkService().getIsUploadPhoto().observe(ge
tViewLifecycleOwner(), isUpload -> {
            if (isUpload && photoName.equals(Config.PHOTO_1)) {
                dialog.cancel();
                buttonAddPhoto1.setText("✓");
                buttonAddPhoto1.setEnabled(false);
                buttonAddPhoto1.setBackgroundColor(ContextCompat.getColor(getContext(),
R.color.green));
            } else if (isUpload && photoName.equals(Config.PHOTO_2)) {
                dialog.cancel();
                buttonAddPhoto2.setText("✓");
                buttonAddPhoto2.setEnabled(false);
                buttonAddPhoto2.setBackgroundColor(ContextCompat.getColor(getContext(),
R.color.green));
            } else if (isUpload && photoName.equals(Config.PHOTO_3)) {
                dialog.cancel();
                buttonAddPhoto3.setText("✓");
                buttonAddPhoto3.setEnabled(false);
                buttonAddPhoto3.setBackgroundColor(ContextCompat.getColor(getContext(),
R.color.green));
            }
        });
```

```
    }

    View.OnClickListener onClickListener = v -> {
        switch (v.getId()) {
            case R.id.button_create_meter_reading: {
                if (currentMeterPoint != null &&
                        !editTextNumMeterReading.getText().toString().isEmpty() &&
                        !editTextMeterReading.getText().toString().isEmpty()) {

                    currentMeterPoint.setReadingDone(true);

currentMeterPoint.setDeviceStatus(deviceStatusList.get(spinnerMeterStatus.getSelectedIt
emPosition()));

                    currentMeterPoint.setDateMetering(Config.getCurrentDateTime());

currentMeterPoint.setNumMeter(editTextNumMeterReading.getText().toString());

currentMeterPoint.setMeterReading(editTextMeterReading.getText().toString());


currentMeterPoint.setProblemPoint(isProblemOnMeterPoint.isChecked());

currentMeterPoint.setProblemDescription(editTextProblemDescription.getText().toString()
);


Repository.getInstance(getActivity()).getNetworkService().addMeterPoint(currentMeterPoi
nt);
                    Toast.makeText(getContext(), "Контрольное снятие показаний
выполнено.", Toast.LENGTH_SHORT).show();
                    navController.popBackStack();
                } else if (editTextNumMeterReading.getText().toString().isEmpty() &&
editTextMeterReading.getText().toString().isEmpty()) {
                    Toast.makeText(getContext(), "Заполните все поля.",
Toast.LENGTH_SHORT).show();
                } else if (photoName.equals("") && buttonAddPhoto1.isActivated()) {
                    Toast.makeText(getContext(), "Добавьте хотябы одно фото.",
Toast.LENGTH_SHORT).show();
                } else {
                    Toast.makeText(getContext(), "Нет активных точек учёта для
контрольного снятия показаний.", Toast.LENGTH_LONG).show();

                }
                break;
            }
            case R.id.button_add_photo1: {
                addPhoto(Config.PHOTO_1);
                break;
            }
            case R.id.button_add_photo2: {
                addPhoto(Config.PHOTO_2);
                break;
            }
            case R.id.button_add_photo3: {
                addPhoto(Config.PHOTO_3);
                break;
            }
        }
    };
```

```
    List<MeterPoint> filterMeterPoint(List<MeterPoint> list) {
        List<MeterPoint> tempMeterPoints = new ArrayList<>();
        String userId =
Repository.getInstance(getActivity()).getNetworkService().getCurrentUserInfo().getValue
().getId();
        for (MeterPoint m : list) {
            if (!m.isReadingDone() && m.getUserId().equals(userId)) {
                tempMeterPoints.add(m);
            }
        }
        return tempMeterPoints;
    }

    void checkPermission() {
        //Проверяем разрешение на работу с камерой
        boolean isCameraPermissionGranted =
ActivityCompat.checkSelfPermission(getContext(), android.Manifest.permission.CAMERA) ==
PackageManager.PERMISSION_GRANTED;
        //Проверяем разрешение на работу с внешнем хранилещем телефона
        boolean isWritePermissionGranted =
ActivityCompat.checkSelfPermission(getContext(),
android.Manifest.permission.WRITE_EXTERNAL_STORAGE) ==
PackageManager.PERMISSION_GRANTED;

        //Если разрешения != true
        if (!isCameraPermissionGranted || !isWritePermissionGranted) {

            String[] permissions;//Разрешения которые хотим запросить у пользователя

            if (!isCameraPermissionGranted && !isWritePermissionGranted) {
                permissions = new String[]{android.Manifest.permission.CAMERA,
android.Manifest.permission.WRITE_EXTERNAL_STORAGE};
            } else if (!isCameraPermissionGranted) {
                permissions = new String[]{android.Manifest.permission.CAMERA};
            } else {
                permissions = new
String[]{android.Manifest.permission.WRITE_EXTERNAL_STORAGE};
            }
            //Запрашиваем разрешения у пользователя
            ActivityCompat.requestPermissions(getActivity(), permissions,
REQUEST_CODE_PERMISSION_RECEIVE_CAMERA);
        }
    }

    void addPhoto(String photoName) {
        this.photoName = photoName;
        dispatchTakePictureIntent();
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (requestCode == REQUEST_TAKE_PHOTO && resultCode == RESULT_OK) {
            Toast.makeText(getContext(), "Загрузка фото.", Toast.LENGTH_SHORT).show();
            setProgressDialog();

Repository.getInstance(getActivity()).getNetworkService().addMeterPointPhoto(photoURI,
currentMeterPoint.getId(), photoName);
        }
    }

    private File createImageFile() throws IOException {
```

```
        // Create an image file name
        String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
        String imageFileName = "JPEG_" + timeStamp + "_";
        File storageDir =
getActivity().getExternalFilesDir(Environment.DIRECTORY_PICTURES);
        File image = File.createTempFile(
                imageFileName,  /* prefix */
                ".jpg",         /* suffix */
                storageDir      /* directory */
        );

        // Save a file: path for use with ACTION_VIEW intents
        mCurrentPhotoPath = image.getAbsolutePath();
        return image;
    }

    private void dispatchTakePictureIntent() {
        Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
        // Ensure that there's a camera activity to handle the intent
        if (takePictureIntent.resolveActivity(getActivity().getPackageManager()) !=
null) {
            // Create the File where the photo should go
            File photoFile = null;
            try {
                photoFile = createImageFile();
            } catch (IOException ex) {
                // Error occurred while creating the File
                Toast.makeText(getContext(), "Error!", Toast.LENGTH_SHORT).show();
            }
            // Continue only if the File was successfully created
            if (photoFile != null) {
                photoURI = FileProvider.getUriForFile(getContext(),
                        "ru.gureev.meteringandmonitorsystem",
                        photoFile);
                takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, photoURI);
                startActivityForResult(takePictureIntent, REQUEST_TAKE_PHOTO);
            }
        }
    }

    public void setProgressDialog() {

        int llPadding = 30;
        LinearLayout ll = new LinearLayout(getContext());
        ll.setOrientation(LinearLayout.HORIZONTAL);
        ll.setPadding(llPadding, llPadding, llPadding, llPadding);
        ll.setGravity(Gravity.CENTER);
        LinearLayout.LayoutParams llParam = new LinearLayout.LayoutParams(
                LinearLayout.LayoutParams.WRAP_CONTENT,
                LinearLayout.LayoutParams.WRAP_CONTENT);
        llParam.gravity = Gravity.CENTER;
        ll.setLayoutParams(llParam);

        ProgressBar progressBar = new ProgressBar(getContext());
        progressBar.setIndeterminate(true);
        progressBar.setPadding(0, 0, llPadding, 0);
        progressBar.setLayoutParams(llParam);

        llParam = new LinearLayout.LayoutParams(ViewGroup.LayoutParams.WRAP_CONTENT,
                ViewGroup.LayoutParams.WRAP_CONTENT);
        llParam.gravity = Gravity.CENTER;
        TextView tvText = new TextView(getContext());
```

```
        tvText.setText("Загрузка ...");
        tvText.setTextColor(Color.parseColor("#000000"));
        tvText.setTextSize(20);
        tvText.setLayoutParams(llParam);

        ll.addView(progressBar);
        ll.addView(tvText);

        AlertDialog.Builder builder = new AlertDialog.Builder(getContext());
        builder.setCancelable(false);
        builder.setView(ll);

        dialog = builder.create();

        dialog.show();
        Window window = dialog.getWindow();
        if (window != null) {
            WindowManager.LayoutParams layoutParams = new WindowManager.LayoutParams();
            layoutParams.copyFrom(dialog.getWindow().getAttributes());
            layoutParams.width = LinearLayout.LayoutParams.WRAP_CONTENT;
            layoutParams.height = LinearLayout.LayoutParams.WRAP_CONTENT;
            dialog.getWindow().setAttributes(layoutParams);
        }
    }
}
```