Developer Documentation for Location Simulator

Lars Fockenga

April 22, 2025

Contents

1 Installation **3**
1.1 Installing Fastlane . . . 3

2 General Information **4**
2.1 Architektur . . . 4
2.1.1 Data Bindings . . . 4
2.1.2 Further Resources . . . 4

3 Project Structure **5**
3.1 core.util . . . 5
3.2 data . . . 5
3.2.1 repository . . . 5
3.2.2 source . . . 5
3.2.3 storageManager . . . 5
3.3 di . . . 5
3.4 domain . . . 5
3.4.1 model . . . 6
3.4.2 repository . . . 6
3.4.3 useCase . . . 6
3.5 network . . . 6
3.6 presentation . . . 7
3.6.1 add . . . 7
3.6.2 delay . . . 8
3.6.3 editTimeline . . . 9
3.6.3.1 components . . . 9
3.6.4 exportSettings . . . 10
3.6.5 homescreen . . . 10
3.6.6 previewData . . . 11
3.6.7 run . . . 11
3.6.8 select . . . 12
3.6.8.1 components . . . 12
3.6.9 settings . . . 13
3.6.10 sound . . . 14
3.6.11 trainerScreen . . . 14
3.6.12 universalComponents . . . 15
3.6.13 userSettings . . . 15
3.6.14 util . . . 15

3.7 ui.theme . . . 15
3.8 Non Kotlin Files . . . 16
3.8.1 Assets . . . 16
3.8.2 Res . . . 16
3.8.3 Gradle . . . 16

4 Files outside the main app **17**
4.1 ./projectmanagement . . . 17
4.2 ./docs . . . 17
4.2.1 LaTex/DeveloperDocumentation . . . 17
4.2.2 fastlane-screenshots . . . 17
4.3 ./android-app . . . 17
4.3.1 .run . . . 17
4.3.2 fastlane . . . 17

5 Tests **18**
5.1 AndroidTest . . . 18
5.1.1 di . . . 18
5.1.2 endToEndTests . . . 18
5.1.3 presentation.editTimeline . . . 18
5.1.4 screenshots . . . 18
5.1.4.1 phone . . . 18
5.2 Test . . . 19
5.2.1 domain.model . . . 19
5.2.2 presentation.delay . . . 19

1 Installation

To work with this project, clone the repository. It is recommended to do this manually and not with a tool like Android Studio, as the root of the gradle project is not the very first folder, but instead the android-app folder contained in it.

To now work with the project, open the android-app folder as a new project in Android Studio. Android Studio should now automatically guide you through the rest of the setup and install all dependencies.

1.1 Installing Fastlane

To use fastlane, you first need to install Ruby, and then fastlane. To install fastlane, follow these instructions. More information on fastlane can be found here.

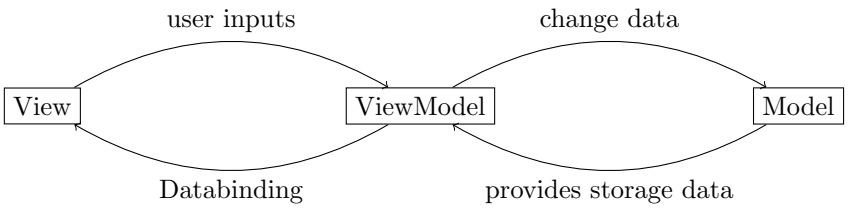


Figure 1: Overview of the MVVM architecture

2 General Information

2.1 Architektur

This app uses the Model View ViewModel (MVVM) architecture, which is a common architecture, splitting the software into three layers:

- **The Model** contains the data, which can be modified by the ViewModel. The data is independent of the other layers, meaning multiple views implementations can use the same model. It might also contain some business logic, like accessing the devices main memory. For this project, the Models will be called XYZState.
- **The ViewModel** takes user inputs from the View and modifies the models data accordingly. It also iffer the view access to the state of the model, so any changes can be reflected in the UI. For this project, the View Models will be called XYZViewModel, with event that can happen being called XYZEvent.
- **The View** reacts to changes in the ViewModel and reflects those changes in the UI. It should contain no business logic. For this project, the Views will be called XYZScreen.

This architecture was used as the apps UI used Jetpack Compose, which is made to work with a ViewModel. The separation of the layers also allows for independent changes between UI and business logic, as well as bester testability, as no complex UI tests are needed to test the business logic.

2.1.1 Data Bindings

For databindings, Jetpack Compose already provides states that take care of updating and recomposing the UI.

2.1.2 Further Resources

This YouTube video further explains the MVVM architecture and was used by the developers of this app to learn it.

3 Project Structure

The following section refer to all the packages found in android-app/app/src/main/java/com/ispgr5/locationsimulator.

3.1 core.util

- **TestTags** Contains all the test tags which can be used to test the app with it's UI. The most common use cases would be to check if a UI element exists, to get it's value or to click on it.

3.2 data

This package contains a variety of classes for working with the database.

3.2.1 repository

- **ConfigurationRepositoryImpl** Implements all the function that can be called to read configurations from the database, or write them to it.

3.2.2 source

- **ConfigurationDao** Data Access Object for the Database where SQL queries are defined.
- **ConfigurationDatabase** Defines the database which stores the configurations.

3.2.3 storageManager

- **ConfigurationStorageManager** This class is responsible for importing and exporting configurations. It turns sound files into a base64 string, creates a json for the configuration, and compresses them using gzip.
- **SoundStorageManager** This class allows us to use the sound files, which are stored on the devices main storage.

3.3 di

- **AppModule** This data injection module loads the database when starting the app and provides it's interface to the view models.

3.4 domain

Contains a variety of classes to represent a configuration on the database.

3.4.1 model

- **ConfigComponent** Super Class for any configuration components. Contains subclasses for both vibration and sound components.
- **Configuration** Class that represents the configuration, allows it to be stored on the database.
- **ConfigurationComponentRoomConverter** Converts a list of configuration components to a string, and vise versa.
- **RangeConverter** Converts user friendly number values into technical numbers, and vise versa.
- **Sound Converter** Converts sound files to base64 strings, and vise versa.

3.4.2 repository

- **ConfigurationRepository** Interface for the database repository, defines which functions the database provides.

3.4.3 useCase

- **AddConfiguration** Manager to add a configuration to the database.
- **ConfigurationUseCases** Interface for all database operations.
- **DeleteConfiguration** Manager to delete a configuration from the database.
- **GetConfiguration** Manager to get a configuration by it's id.
- **GetConfigurations** Manager to get all configurations from the database.
- **GetFavoriteConfigurations** Manager to get favored configurations from the database.

3.5 network

Contains the backend classes for the network connection, necessary to enable the remote control function.

- **Client** The client running on the trainer device.
- **ClientHandler** Handles communication with the client for the server.
- **NetworkUtils** General use network tools, like the valid commands.
- **Server** Server running on the hidden devices.

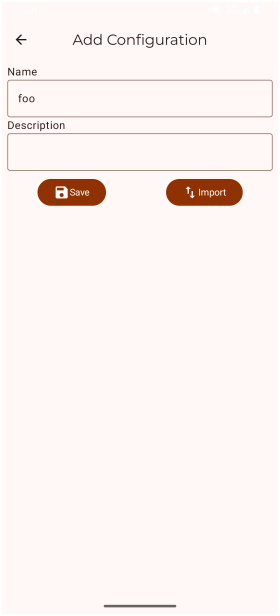
3.6 presentation

Contains the classes for all the different screens in the app.

- **MainActivity** Entry point for the app, handles many essential functions, like navigation, starting/stopping the background task, loading preferences, initial setup of the app on install and some more.

3.6.1 add

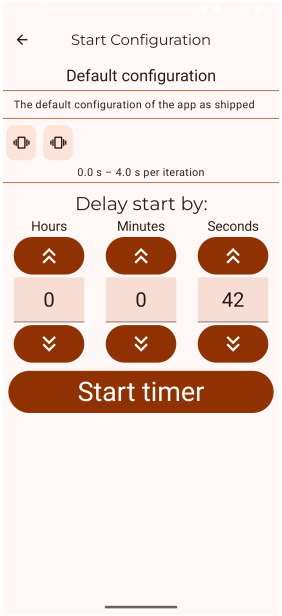
Contains the classes for the "Add Configuration" screen.



- **AddEvent** Defines all events which can happen on the add screen.
- **AddScreen** View for the add screen, contains all the composable that make up the add screen.
- **AddScreenState** Contains the state of the add screen.
- **AddViewModel** View Model for the add screen. Reacts to all UI events on this screen.

3.6.2 delay

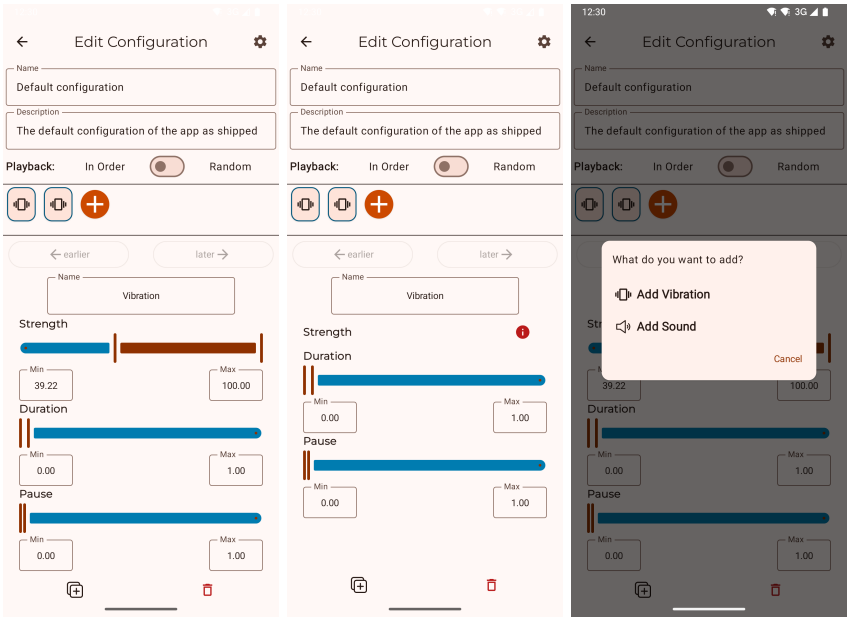
Contains all classes for the "Start Configuration" screen, where the user can start the configuration and set a delay for the start.



- **DelayEvent** Defines all events which can happen on the delay/start screen.
- **DelayScreen** View for the delay/start screen, contains all the composable that make up the delay/start screen.
- **DelayScreenState** Contains the state of the delay/start screen.
- **DelayTimer** Implementation for the delay timer. Includes both busness logic, as well as composables.
- **DelayViewModel** View Model for the delay/start screen. Reacts to all UI events on this screen.

3.6.3 editTimeline

Contains all classes for the "Edit Configuration" screen.



- **EditTimelineEvent** Defines all events which can happen on the edit-configuration screen.
- **EditTimelineScreen** View for the edit-configuration screen, contains the composable that make up the edit-configuration screen. All composables for the timeline and below heavily rely on those defined in 3.6.3.1.
- **EditTimelineScreenState** Contains the state of the edit-configuration screen.
- **EditTimelineViewModel** View Model for the edit-configuration screen. Reacts to all UI events on this screen.

3.6.3.1 components

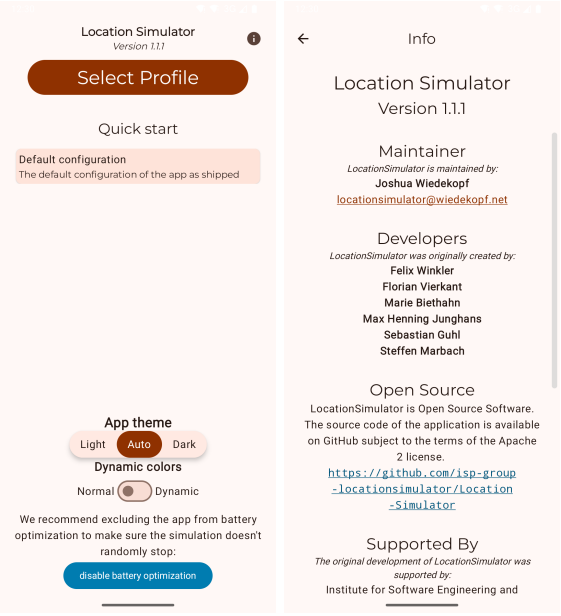
- **AddConfigComponentDialog** View for when the user trys to add a configuration component and needs to decide whether they want to add a vibration or sound.
- **EditConfigComp** Contains all the composables for the view used to edit an element of a configuration. This means everything below the timeline.
- **Timeline** Contains all the composables for the view of the timeline on the edit configuration screen.

Contains the classes for the "Export" screen, which allows a trainer to wirelessly save a configuration on a remote device. Currently unfinished and not used.

- **ExportSettingsEvent** Defines all events which can happen on the export settings screen.
- **ExportSettingsScreen** View for the export settings screen, contains all the composable that make up the export settings screen.
- **ExportSettingsScreenState** Contains the state of the export settings screen.
- **ExportSettingsViewModel** View Model for the export settings screen. Reacts to all UI events on this screen.

3.6.4 homescreen

Contains all classes for the home screen, where the user can go to the configuration choice, see the favored configurations, and change the theme. he can also choose his name and remote-control role here. Also contains the info and help screen.



- **HelpScreen** Help screen, which explains to the user how to perform varous tasks.
- **HomeEvent** Defines all events which can happen on the home screen.
- **HomeScreen** View for the home screen, contains all the composable that make up the home screen.
- **HomeScreenState** Contains the state of the home screen.
- **HomeViewModel** View Model for the home screen. Reacts to all UI events on this screen.

- **InfoScreen** The info screen, containing acknowledgements to the developers and maintainers, as well as the license and some more.

3.6.5 previewData

Contains everything necessary to automatically generate previews.

- **PreviewAnnotations** Defines the different settings for the different preview annotations, which can be used to automatically generate previews with different themes, screens sizes, locales, and font sizes. , Sprachen und Schriftgrößen generiert werden können.
- **PreviewData** Defines the standard data to be used in previews.

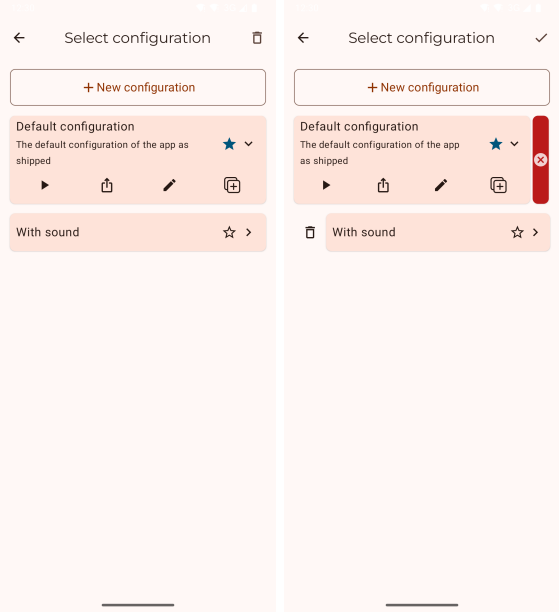
3.6.6 run

Contains all the classes for the screen that is shown, while the configuration is running. Also contains the actual function for running a configuration.

- **RunEvent** Defines all events which can happen on the run screen.
- **RunScreen** View for the run screen, contains all the composable that make up the run screen.
- **RunViewModel** View Model for the run screen. Reacts to all UI events on this screen.
- **SimulationService** The class that actually runs a configuration, causes vibrations and plays sounds.
- **SoundPlayer** Helper-class for playing sounds.

3.6.7 select

Contains all for the "Select configuration" screen, where a user can select a configuration to play, export, edit, copy or delete.



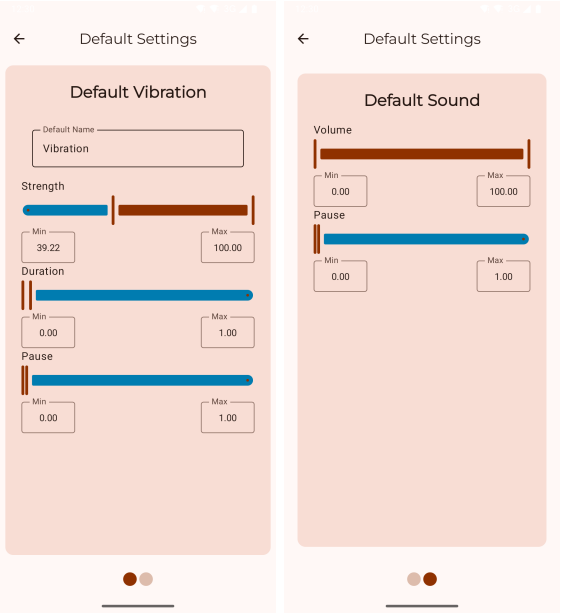
- **SelectEvent** Defines all events which can happen on the select screen.
- **SelectScreen** View for the select screen, contains all the composable that make up the select screen.
- **SelectScreenState** Contains the state of the select screen.
- **SelectViewModel** View Model for the select screen. Reacts to all UI events on this screen.

3.6.7.1 components

- **ConfigurationList** Contains the composables that make up the car for one configuration.

3.6.8 settings

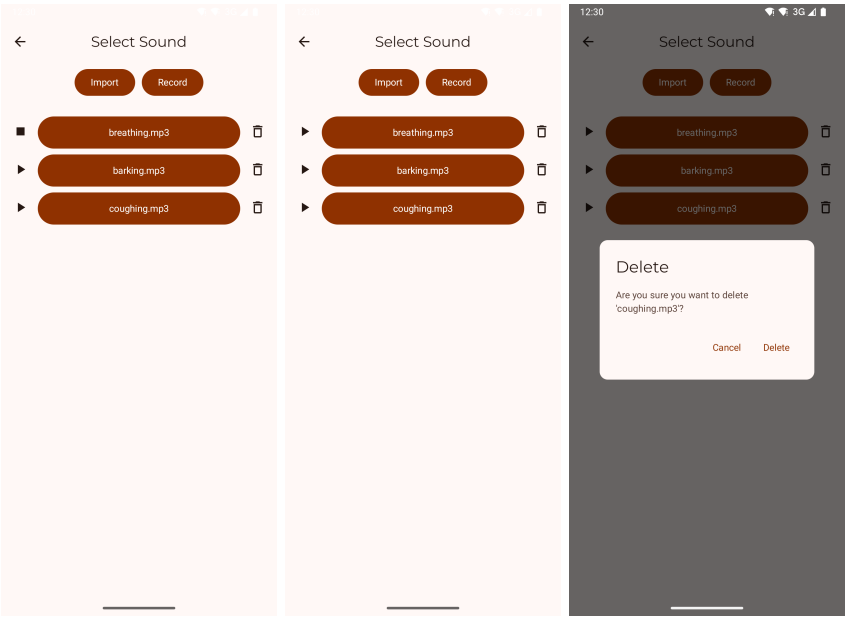
Contains all classes for the "Default Settings" screen.



- **SettingsEvent** Defines all events which can happen on the default settings screen.
- **SettingsScreen** View for the default settings screen, contains all the composable that make up the default settings screen.
- **SettingsScreenState** Contains the state of the default settings screen.
- **SettingsViewModel** View Model for the default settings screen. Reacts to all UI events on this screen.

3.6.9 sound

Contains all classes for the "Select Sound" screen, where a user can choose what sound to add to a configuration.



- **SingleSound** Composable for a single sound file, with a button to play it and a button to delete it.
- **SoundDialog** Composable for giving a newly recorded audio a name.
- **SoundEvent** Defines all events which can happen on the sound screen.
- **SoundScreen** View for the sound screen, contains all the composable that make up the sound screen.
- **SoundScreenState** Contains the state of the sound screen.
- **SoundViewModel** View Model for the sound screen. Reacts to all UI events on this screen.

3.6.10 trainerScreen

Contains the classes for the "Trainer Options" screen. This is where the trainer can control all the remote devices.

- **TrainerScreenEvent** Defines all events which can happen on the trainer screen screen.
- **TrainerScreenScreen** View for the trainer screen screen, contains all the composable that make up the trainer screen screen.

- **TrainerScreenScreenState** Contains the state of the trainer screen screen.
- **TrainerScreenViewModel** View Model for the trainer screen screen. Reacts to all UI events on this screen.

3.6.11 universalComponents

- **Clickable Link** A clickable link with a default look.
- **ConfirmDeleteDialog** Dialog to confirm if you want to delete something.
- **SnackbarContent** Default snackbar for short notifications.
- **TobBar** Default top bar displaying a back button, a title and possibly extra actions like delete.

3.6.12 util

- **BackPressGestureDisabler** Disables the back gesture on phones with gesture control.
- **BigDecimal** Allows for Numbers to be converted to BigDecimal, and adds some usefull functions to BigDecimal.
- **Screen** Class to get the route to screens.
- **Snackbar** Defines custom snackbars to make working with them easier and make them more consistent.
- **VibrationAmplituteControl** Custom functions to make working with vibrations across different API versions easier.

3.7 ui.theme

- **Color** Defines all the colors used in the app.
- **Theme** Combines multiple colors into different themes.
- **ThemeState** Stores the current theme state, aka. whether the theme is light, dark, or auto, and weather dynamic colors are on.
- **Type** Defines the fonts for the app.

3.8 Non Kotlin Files

3.8.1 Assets

Found under android-app/app/src/main/assets.

- **Sounds** Contains the default sounds included in the app.

3.8.2 Res

Found under android-app/app/src/main/res/drawable.

- **Drawable** Contains most icons used in the app, defined as vector files, as well as the ISP logo.
- **Mipmap** Contains different versions of the app logo in different resolutions.
- **Values** Contains values for colors, the base theme, font certificates, as well as the strings used in the app.
- **Xml** Includes currently empty backup and data extraction rules, the file path to store exports, as well as the config file for the available locales.

3.8.3 Gradle

As gradle is used to build the app, there are a number of gradle files, namely:

- build.gradle for the whole project, build.gradle for the app
- proguard-rules.pro
- gradle.properties
- gradle-wrapper.properties
- local.properties
- settings.gradle

4 Files outside the main app

4.1 ./projectmanagement

This folder contains the description and images for the release in the Google Play Store.

4.2 ./docs

4.2.1 LaTex/DeveloperDocumentation

Contains the LaTex files for this documentation.

4.2.2 fastlane-screenshots

Folder for fastlane to store it's automatically generated screenshots, which can be used in documentation or otherwise. These screenshots are generated in different themes, states, and languages.

4.3 ./android-app

- **Gemfile** Contains the gems needed for fastlane.
- **Gemfile.lock** Locks all the dependency versions.

4.3.1 .run

Contains all run configurations for Android Studio.

- **Fastlane Screenshots** A run configurations to generate new screenshots using fastlane. Make sure to follow 1.1 first.

4.3.2 fastlane

Contains the fastlane files to automatically generate screenshots.

5 Tests

5.1 AndroidTest

These tests test interacting with the app via the UI.

- **ExampleInstrumentedTest** Short example of an instrumented test.
- **HiltTestRunner** Replaces the real apps application class with an instrumented HiltTestApplication. This is specified in the build.gradle script.

5.1.1 di

Test for automatically testing the di package (3.3).

- **TestAppModule** Creates an in-memory environment for testing, specifically it replaces the database and repository with fake version which can easily be reset between tests.

5.1.2 endToEndTests

These tests test a full workflow, from home screen to the end. This is similar to classical integration test.

- **CreateAndPlayConfigurationTest** Tests going from the home screen to creating a new configuration, editing it and then starting and stopping it.
- **SecondaryEndToEndTests** Test workflows not warrenting a own test class. The are currently arranging configuration components in a timeline, and switching between dark and light mode.

5.1.3 presentation.editTimeline

Tests for automatically testing the editTimeline package (3.6.3).

- **EditTimelineScreenTest** Test adding a component and changing the configuration name and description on the edit screen.

5.1.4 screenshots

Test for automatically creating screenshots using Fastlane.

- **ScreenshotTests** An abstract base class for automatically taking screenshots using Fastlane.

5.1.4.1 phone

- **PhoneScreenshotTests** Implementation of ScreenshotTests for taking screenshots on a phone.

5.2 Test

These tests test function not specifically related to android.

5.2.1 domain.model

These tests test function from the model package (3.4.1).

- **ConfigurationComponentRoomConverterTest** Test that the ConfigurationComponentRoomConverter correctly converts config components into strings, and vice versa.
- **RangeConverterTest** Tests the RangeConverter.
- **SoundConverterTest** Tests the SoundConverter.

5.2.2 presentation.delay

These tests test function from the delay package (3.6.2).

- **TimerTest** Tests that the DelayTimer correctly calculates the timer value from a string.