

Entwicklerdokumentation für Ortungssimulator

Lars Fockenga

22. April 2025

Inhaltsverzeichnis

1 Installation	3
1.1 Fastlane Installieren	3
2 Allgemeine Informationen	4
2.1 Architektur	4
2.1.1 Datenverbindungen	4
2.1.2 Weitere Ressourcen	4
3 Projektaufbau	5
3.1 core.util	5
3.2 data	6
3.2.1 repository	6
3.2.2 source	6
3.2.3 storageManager	6
3.3 di	7
3.4 domain	7
3.4.1 model	7
3.4.2 repository	8
3.4.3 useCase	8
3.5 network	9
3.6 presentation	10
3.6.1 add	11
3.6.2 delay	12
3.6.3 editTimeline	13
3.6.3.1 components	14
3.6.4 exportSettings	14
3.6.5 homescreen	15
3.6.6 previewData	16
3.6.7 run	17
3.6.8 select	18
3.6.8.1 components	19
3.6.9 settings	20
3.6.10 sound	21
3.6.11 trainerScreen	23
3.6.12 universalComponents	24
3.6.13 userSettings	25
3.6.14 util	26

3.7	ui.theme	27
3.8	Nicht Kotlin Dateien	27
3.8.1	Assets	27
3.8.2	Res	27
3.8.3	Gradle	28
4	Dateien außerhalb von App	29
4.1	./projectmanagement	29
4.2	./docs	29
4.2.1	LaTex/DeveloperDocumentation	29
4.2.2	fastlane-screenshots	29
4.3	./android-app	29
4.3.1	.run	29
4.3.2	fastlane	29
5	Tests	30
5.1	AndroidTest	30
5.1.1	di	30
5.1.2	endToEndTests	30
5.1.3	presentation.editTimeline	30
5.1.4	screenshots	30
5.1.4.1	phone	30
5.2	Test	31
5.2.1	domain.model	31
5.2.2	presentation.delay	31

1 Installation

Um an diesem Projekt zu arbeiten, klonen Sie das Git Repository. Dies sollte manuell geschehen, und nicht via Android Studio, da der root Ordner des gradle Projekts nicht der oberste Ordner ist, sondern der enthaltene android-app Ordner.

Um an dem Projekt zu arbeiten, öffne den android-app Ordner als ein neues Projekt in Android Studio. Android Studio sollte dich jetzt durch die weitere Einrichtung führen und alle Abhängigkeiten automatisch installieren.

1.1 Fastlane Installieren

Um Fastlane nutzen zu können, muss zuerst Ruby, und dann Fastlane installiert werden. Um Fastlane zu installieren, folge diesen Anweisungen. Mehr Informationen zu Fastlane können hier gefunden werden.

2 Allgemeine Informationen

2.1 Architektur

Diese App nutzt die Model View ViewModel Architektur, eine weiterverbreite Architektur, welche die Software in drei Schichten unterteilt:

- **Das Model** enthält die Daten, die vom ViewModel modifiziert werden können. Die Daten des Models sind unabhängig von den anderen Schichten und können daher von unterschiedlichen View Implementierungen verwendet werden. Es kann auch Geschäftslogik in dem Model liegen, wie das Einlesen von Daten aus dem Speicher.
- **Das ViewModel** nehmen Eingaben der UI über den View entgegen und modifizieren ggf. die Daten des Models. Sie bieten dem View einen State des Models, sodass die UI bei einer Änderung dieses States angepasst werden kann.
- **Der View** reagiert auf Änderungen des ViewModels und stellt diese in der UI da. Der View enthält keine Geschäftslogik.

Diese Architektur wird verwendet, da die UI der App mit Jetpack Compose entwickelt wurde, welches auf die Verwendung mit einem ViewModel ausgelegt ist. Des weiteren erlaubt die klare Trennung der Schichten eine unabhängige Entwicklung von UI und Geschäftslogik und erlaubt ein testen der Geschäftslogik ohne komplexe UI Tests.

2.1.1 Datenverbindungen

Jetpack Compose kommt bereits mit fertigen Datenverbindungen, welche dafür sorgen das UI und Model entsprechend der Änderungen angepasst werden.

2.1.2 Weitere Ressourcen

Dieses Youtube Video erklärt die MVVM Architektur im Detail und wurde von den Entwicklern dieser App verwendet, um die Architektur zu lernen.

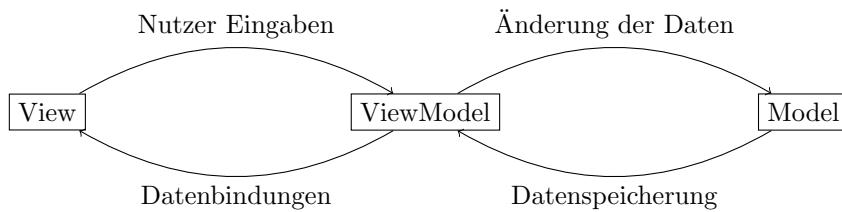
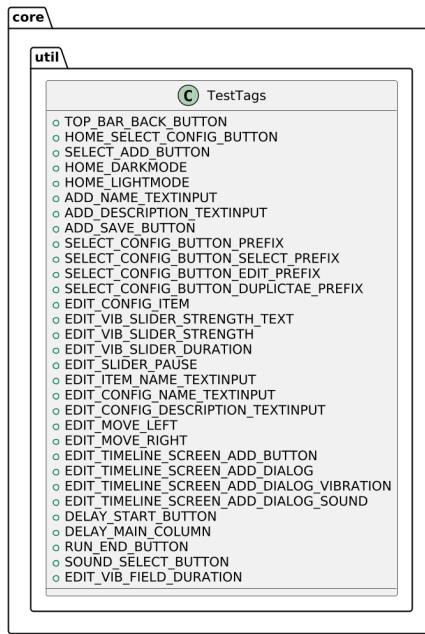


Abbildung 1: Überblick über die MVVM Architektur

3 Projektaufbau

Die folgenden Abschnitte beziehen sich auf die Packages, die in android-app/app/src/main/java/com/ispgr5/locationsimulator gefunden werden können.

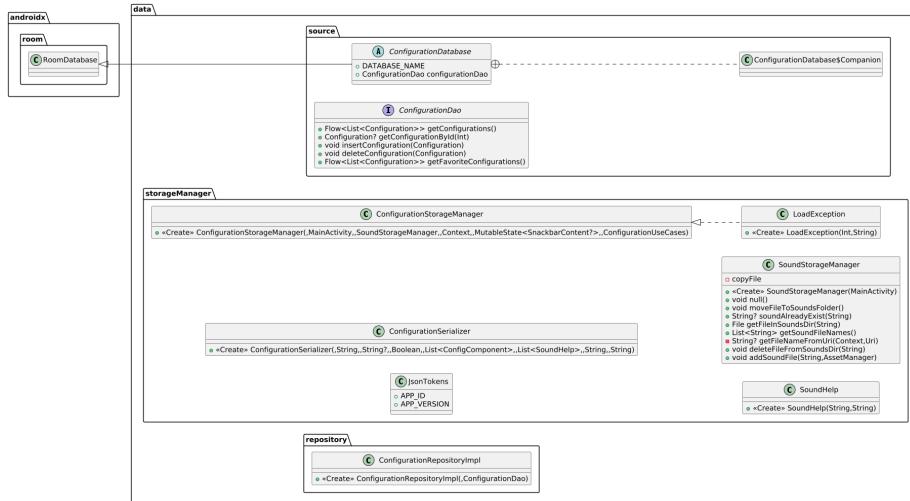
3.1 core.util



- **TestTags** Enthält alle Test Tags die verwendet werden können, um die App über ihre UI zu testen. Typische Verwendungsfälle wären um zu überprüfen, ob ein UI Element existiert, um seinen Wert zu erhalten, oder um auf es zu klicken.

3.2 data

Enthält eine Vielzahl an Klassen, um mit der Datenbank zu arbeiten.



3.2.1 repository

- **ConfigurationRepositoryImpl** Implementiert alle Funktionen, mit denen Konfigurationen von der Datenbank gelesen oder auf die Datenbank geschrieben werden können.

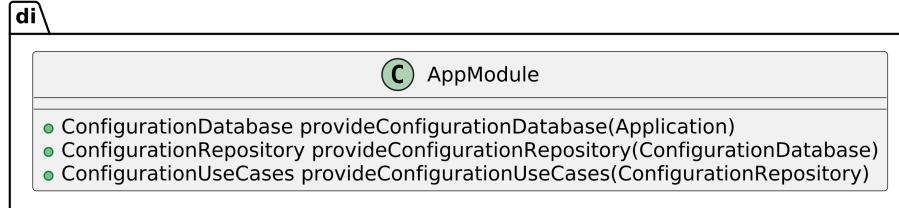
3.2.2 source

- **ConfigurationDao** Data Access Object für die Datenbank, welches die SQL Queries definiert, um auf die Datenbank zuzugreifen.
- **ConfigurationDatabase** Definiert die Datenbank, welche die Konfigurationen speichert.

3.2.3 storageManager

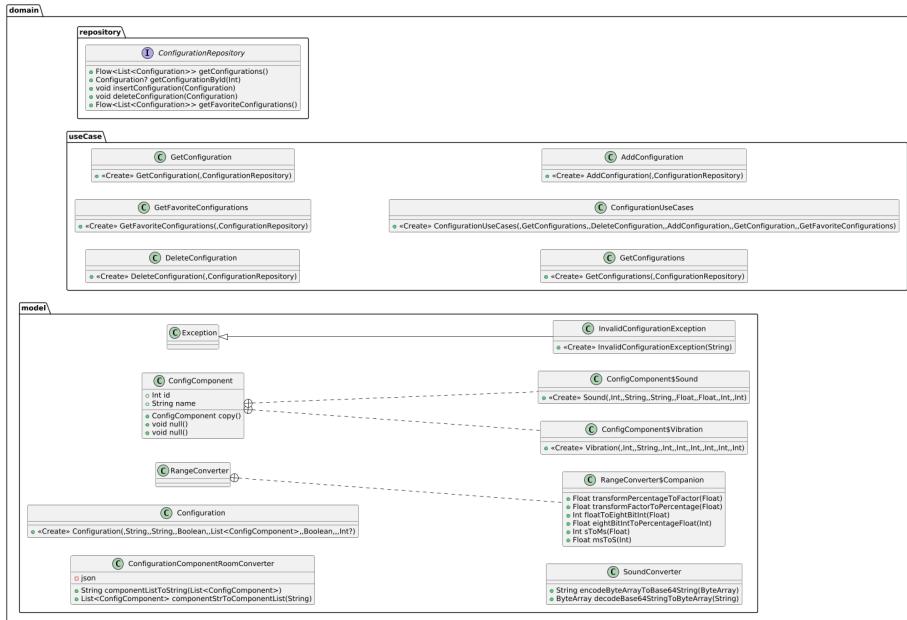
- **ConfigurationStorageManager** Diese Klassen stellt die Funktionalitäten für das Importieren und Exportieren von Konfigurationen zur Verfügung. Dabei werden Audio Dateien in Base64 Strings umgewandelt und die Konfiguration in eine JSON Datei verwandelt, welche mit GZIP komprimiert wird.
- **SoundStorageManager** Diese Funktion ermöglicht es uns die Audio Dateien zu verwenden, welche auf dem Hauptspeicher des Geräts gespeichert sind.

3.3 di



- **AppModule** Diese Data Injection Modul lädt die Datenbank und gibt den View Models ein Interface, um mit der Datenbank zu interagieren.

3.4 domain



Enthält eine Vielzahl an Klassen, um eine Konfiguration auf der Datenbank zu repräsentieren.

3.4.1 model

- **ConfigComponent** Superklasse für jegliche Configurations Komponenten. Enthält Subklassen für Vibration und Sound.
- **Configuration** Klasse für eine gesamte Konfiguration, die auch auf der Datenbank gespeichert wird.
- **ConfigurationComponentRoomConverter** Wandelt eine List von Konfigurations Komponenten in einen String um, und umgekehrt.

- **RangeConverter** Wandelt nutzerfreundliche Zahlenwerte in technische Werte um, und umgekehrt.
- **Sound Converter** Wandelt Sound Dateien in Base64 Strings um, und umgekehrt.

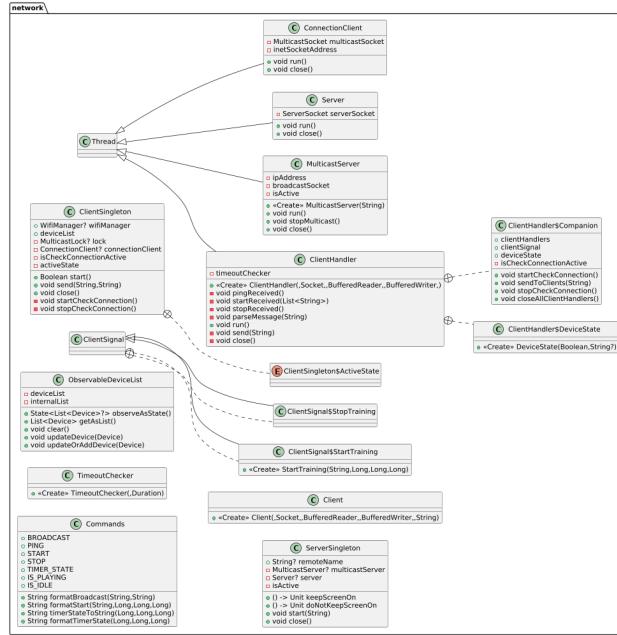
3.4.2 repository

- **ConfigurationRepository** Interface für das Datenbank Repository, definiert, welche Funktionen die Datenbank bereitstellt.

3.4.3 useCase

- **AddConfiguration** Manager um eine Konfiguration der Datenbank hinzuzufügen.
- **ConfigurationUseCases** Interface für alle Datenbank Operationen.
- **DeleteConfiguration** Manager um eine Konfiguration von der Datenbank zu löschen.
- **GetConfiguration** Manager um eine Konfiguration anhand ihrer ID von der Datenbank zu lesen.
- **GetConfigurations** Manager um alle Konfigurationen von der Datenbank zu lesen.
- **GetFavoriteConfigurations** Manager um die favorisierten Konfigurationen von der Datenbank zu lesen.

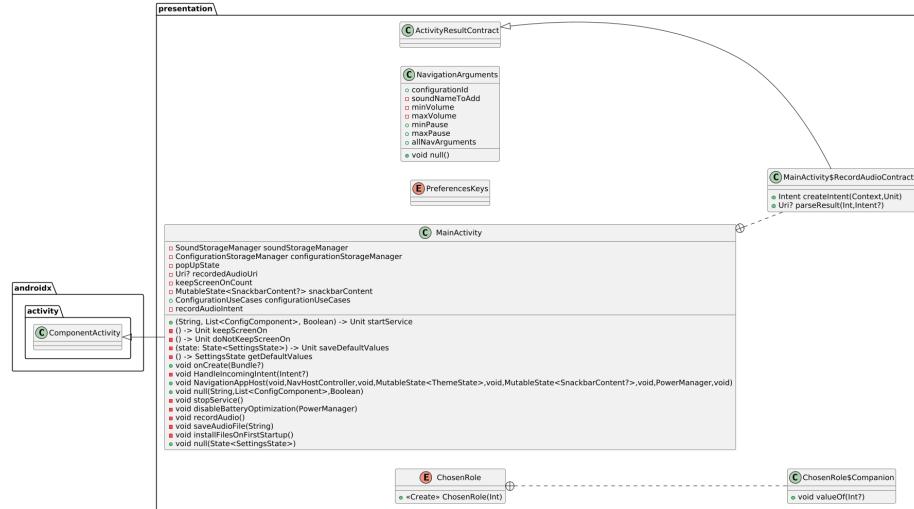
3.5 network



Enthält die Backend Klassen für die Netzwerkverbindung, welche nötig ist um die Fernsteuerungsfunktion zu ermöglichen.

- **Client** Der Client, welcher auf dem Trainer Gerät läuft.
- **ClientHandler** Kümmert sich für den Server um die Kommunikation mit dem Client.
- **NetworkUtils** Allgemeine Netzwerk Tools, wie die List an validen Befehlen.
- **Server** Server, welcher auf den versteckten Geräten läuft.

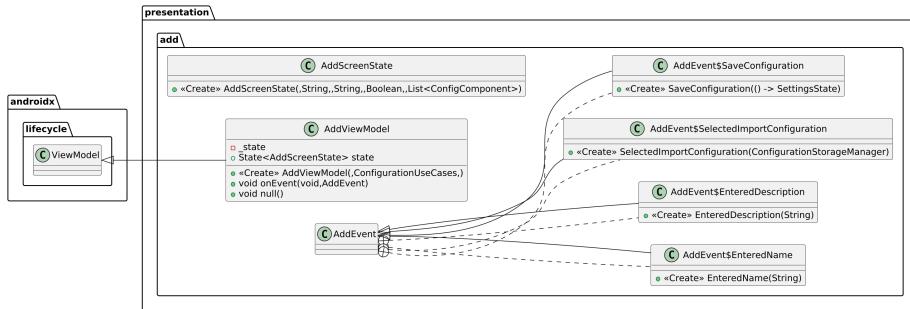
3.6 presentation



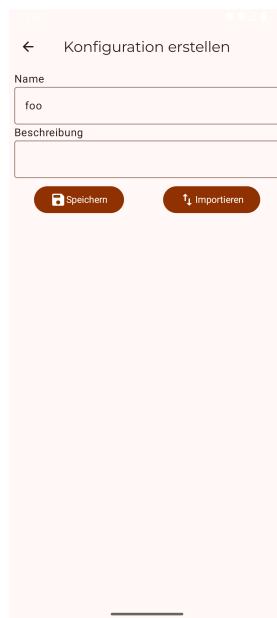
Enthält alle Klassen für die verschiedenen Ansichten der App.

- **MainActivity** Einstiegspunkt für die App, verantwortlich für viele essentielle Funktionen, wie Navigation, starten/stoppen des Laufens im Hintergrund, laden der Nutzerpräferenzen, einrichten der App nach der Installation, un einige mehr.

3.6.1 add

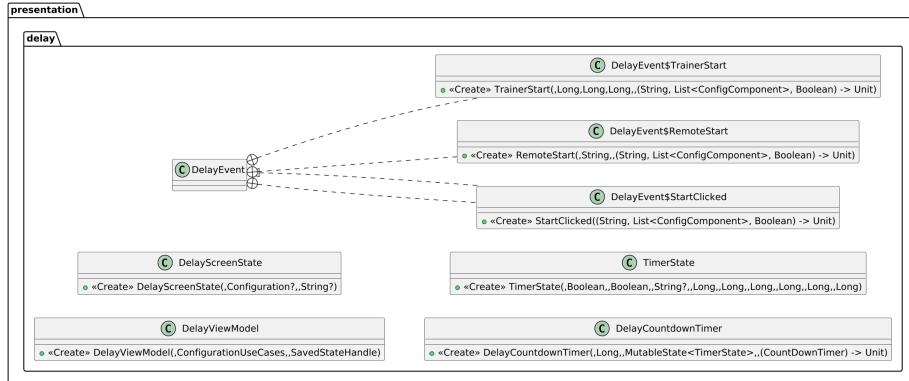


Enthält alle Klassen für die "Konfiguration erstellen" Ansicht.



- **AddEvent** Definiert alle Events die in der Hinzufügen Ansicht auftreten können.
- **AddScreenState** View für die Hinzufügen Ansicht, enthält alle Composable, aus denen diese Ansicht aufgebaut ist.
- **AddViewModel** View Model der Hinzufügen Ansicht. Reagiert auf alle UI Events in dieser Ansicht.

3.6.2 delay



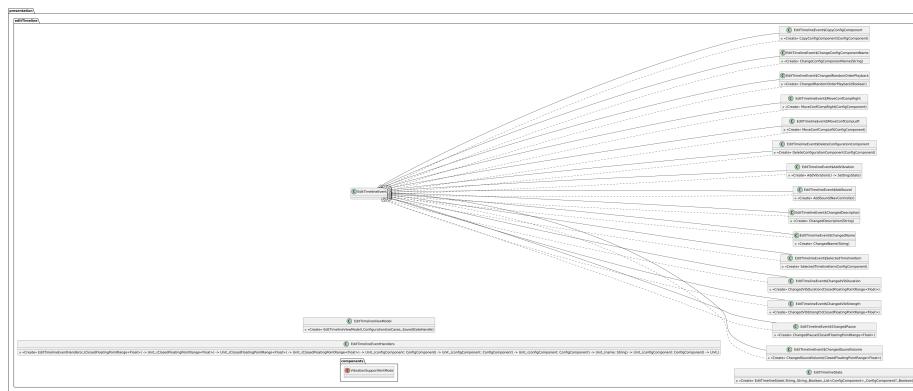
Enthält alle Klassen für die "Konfiguration starten" Ansicht, in welcher der Nutzer die Konfiguration starten kann und eine Verzögerung dafür setzen kann.



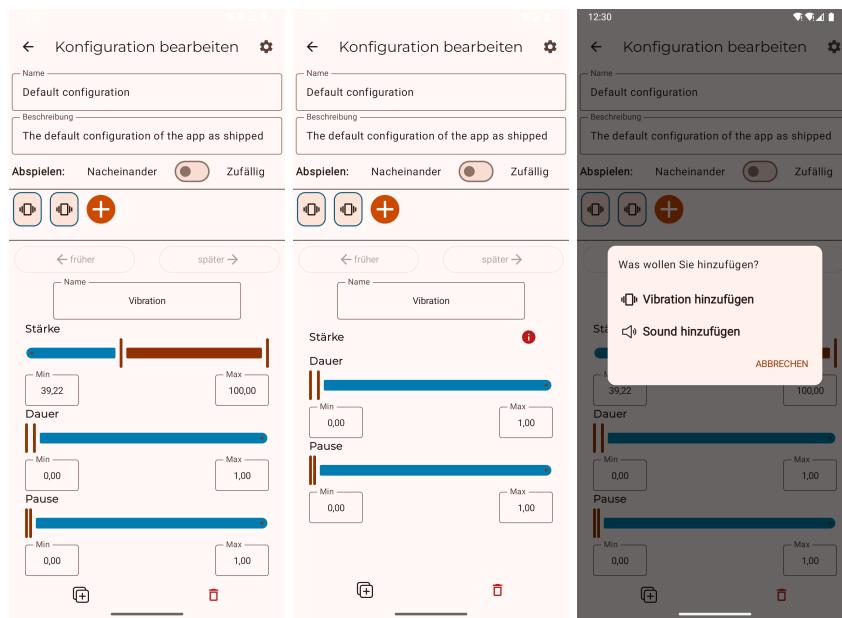
- **DelayEvent** Definiert alle Events die in der Verzögern/Start Ansicht auftreten können.
- **DelayScreen** View für die Verzögern/Start Ansicht, enthält alle Composable, aus denen diese Ansicht aufgebaut ist.
- **DelayScreenState** Speichert den Zustand der Verzögern/Start Ansicht.
- **DelayTimer** Implementierung des Verzögerungstimers. Enthält sowohl Geschäftslogik, als auch Composables.

- **DelayViewModel** View Model der Verzögern/Start Ansicht. Reagiert auf alle UI Events in dieser Ansicht.

3.6.3 editTimeline



Enthält alle Klassen für die "Konfiguration bearbeiten" Ansicht.



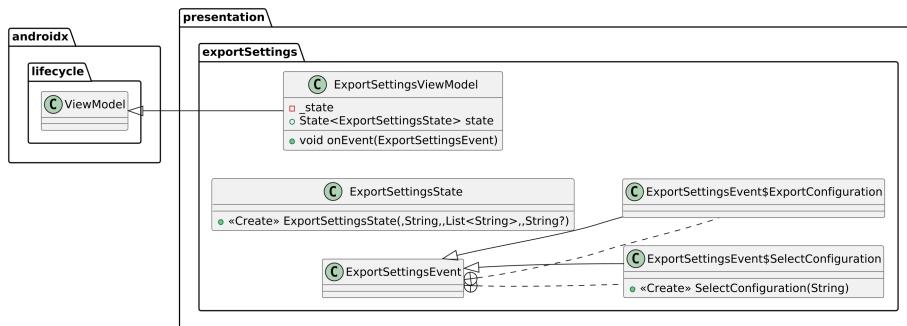
- **EditTimelineEvent** Definiert alle Events die in der Konfiguration bearbeiten Ansicht auftreten können.
 - **EditTimelineScreen** View für die Konfiguration bearbeiten Ansicht, enthält die Composable, aus denen diese Ansicht aufgebaut ist. Alle Composables in der Zeitachse und darunter setzen sich größtenteils aus Composables zusammen, welche in 3.6.3.1 definiert wurden.

- **EditTimelineScreenState** Speichert den Zustand der Konfiguration bearbeiten Ansicht.
- **EditTimelineViewModel** View Model der Konfiguration bearbeiten Ansicht. Reagiert auf alle UI Events in dieser Ansicht.

3.6.3.1 components

- **AddConfigComponentDialog** Der View für wenn der Nutzer ein Element zu einer Konfiguration hinzufügen muss und entscheiden muss, ob er eine Vibration oder einen Sound hinzufügen möchte.
- **EditConfigComp** Enthält alle Komponenten, die in dem View verwendet werden, mit dem ein Element einer Konfiguration bearbeitet werden kann. Dies ist alles unter der Zeitachse.
- **Timeline** Enthält alle Composables für die Zeitachse in der Konfiguration bearbeiten Ansicht.

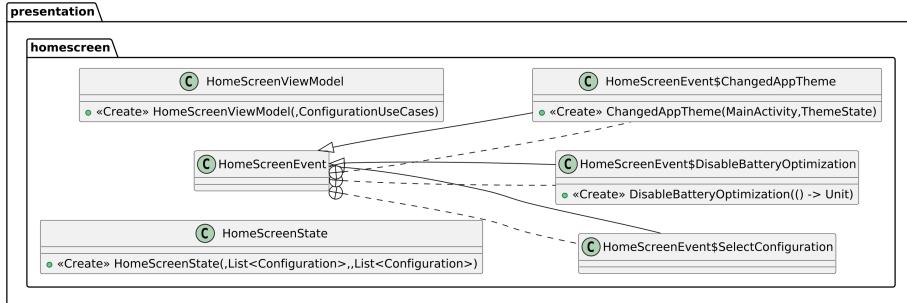
3.6.4 exportSettings



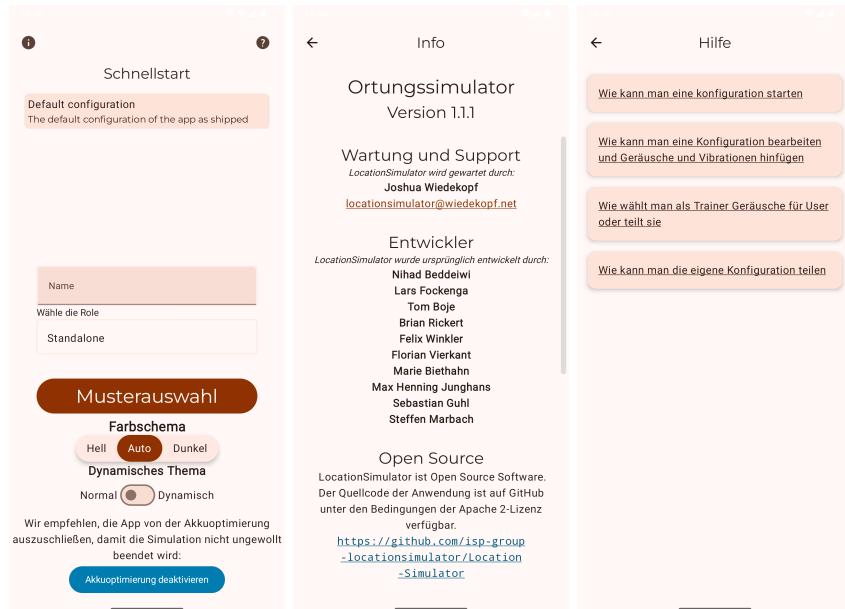
Enthält alle Klassen für die 'Exportieren' Ansicht, in welcher der Trainer eine Konfiguration auf einem Remote Gerät speichern kann. Dieses Feature wurde nicht vollständig implementiert und wird aktuell nicht verwendet.

- **ExportSettingsEvent** Definiert alle Events die in der Hinzufügen Ansicht auftreten können.
- **ExportSettingsScreen** View für die Hinzufügen Ansicht, enthält alle Composables, aus denen diese Ansicht aufgebaut ist.
- **ExportSettingsScreenState** Speichert den Zustand der Hinzufügen Ansicht.
- **ExportSettingsViewModel** View Model der Hinzufügen Ansicht. Reagiert auf alle UI Events in dieser Ansicht.

3.6.5 homescreen



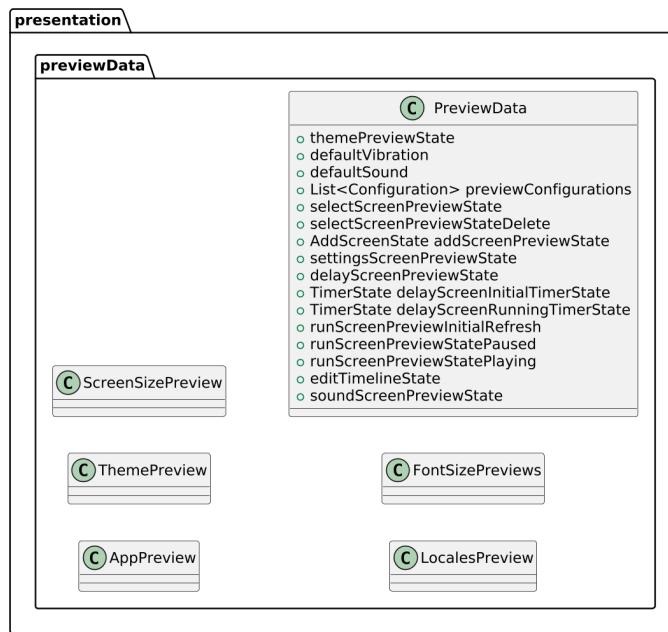
Enthält alle Klassen für den Startbildschirm, wo der Nutzer zur Konfigurations-Auswahl gehen kann, seine favorisierten Konfigurationen sehen kann, und das Farbschema der App ändern kann. Hier kann er auch seinen Namen und seine Rolle für die Fernsteuerungsfunktion wählen. Enthält außerdem die Hilfe und Info Ansichten.



- **HelpScreen** Hilfe Ansicht, in welcher der Nutzer lernen kann, wie er verschiedene Aufgaben bewältigen kann.
- **HomeEvent** Definiert alle Events die in der Home Ansicht auftreten können.
- **HomeScreen** View für die Home Ansicht, enthält alle Composable, aus denen diese Ansicht aufgebaut ist.
- **HomeScreenState** Speichert den Zustand der Home Ansicht.

- **HomeViewModel** View Model der Home Ansicht. Reagiert auf alle UI Events in dieser Ansicht.
- **InfoScreen** Die Informationsansicht, welche Anerkennungen der Entwickler, sowie die Lizenz und einige weitere Informationen enthält.

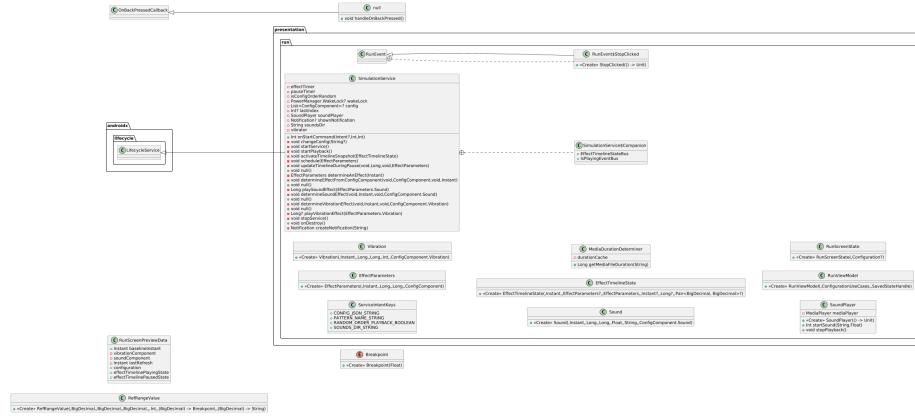
3.6.6 previewData



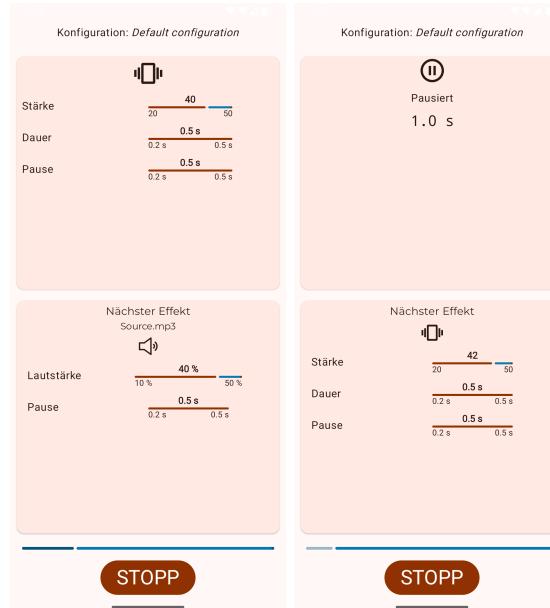
Enthält alles was benötigt wird, um automatische Vorschauen zu erzeugen.

- **PreviewAnnotations** Definiert die verschiedenen Preview Annotations, mit denen automatisch Previews in verschiedenen Farbschemen, Bildschirmgrößen, Sprachen und Schriftgrößen generiert werden können.
- **PreviewData** Definiert die Standartdaten, die für Previews verwendet werden soll.

3.6.7 run



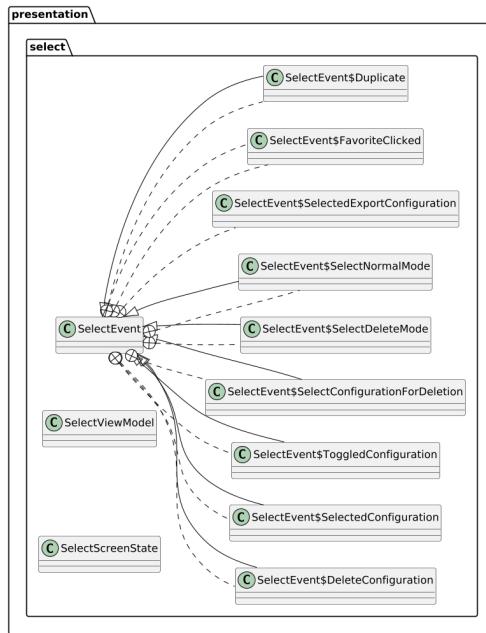
Enthält alle Klassen für die Ansicht die gezeigt wird, während eine Konfiguration läuft. Enthält außerdem die tatsächliche Klasse, welche eine Konfiguration abspielt.



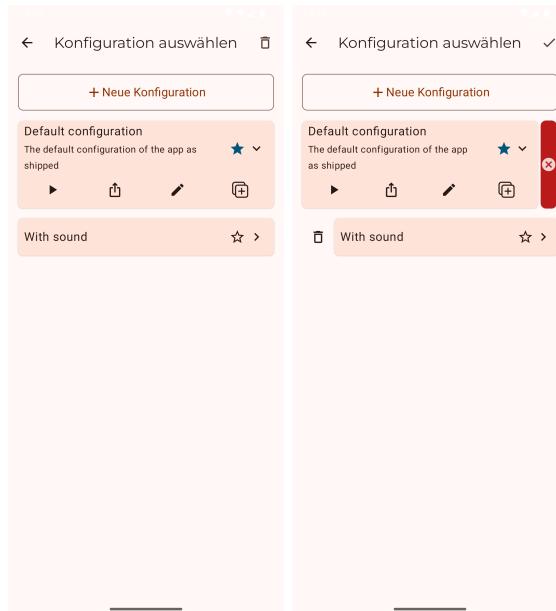
- **RunEvent** Definiert alle Events die in der Laufen Ansicht auftreten können.
- **RunScreen** View für die Laufen Ansicht, enthält alle Composable, aus denen diese Ansicht aufgebaut ist.
- **RunViewModel** View Model der Laufen Ansicht. Reagiert auf alle UI Events in dieser Ansicht.
- **SimulationService** Die Klasse, welche eine Konfiguration tatsächlich abspielt, Vibrationen auslöst und Geräusche abspielt.

- **SoundPlayer** Hilfsklasse, um Geräusche abzuspielen.

3.6.8 select



Enthält alle Klassen für die "Konfiguration auswählen" Ansicht, in der Nutzer eine Konfiguration abspielen, exportieren, bearbeiten, kopieren oder löschen kann.

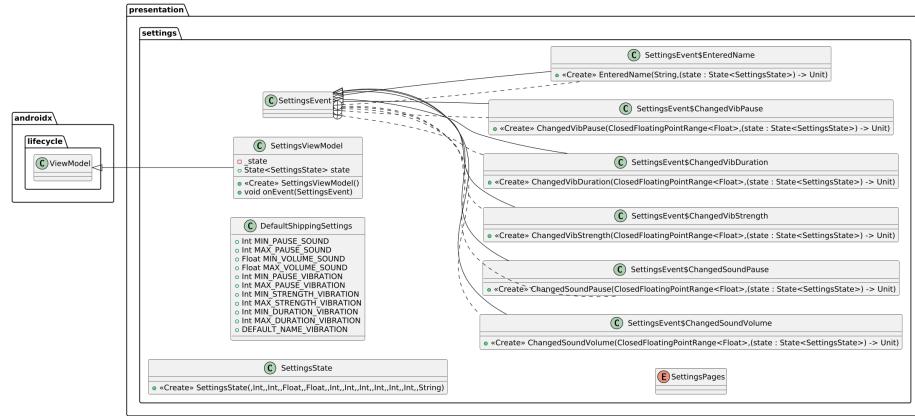


- **SelectEvent** Definiert alle Events die in der Auswahl Ansicht auftreten können.
- **SelectScreen** View für die Auswahl Ansicht, enthält alle Composable, aus denen diese Ansicht aufgebaut ist.
- **SelectScreenState** Speichert den Zustand der Auswahl Ansicht.
- **SelectViewModel** View Model der Auswahl Ansicht. Reagiert auf alle UI Events in dieser Ansicht.

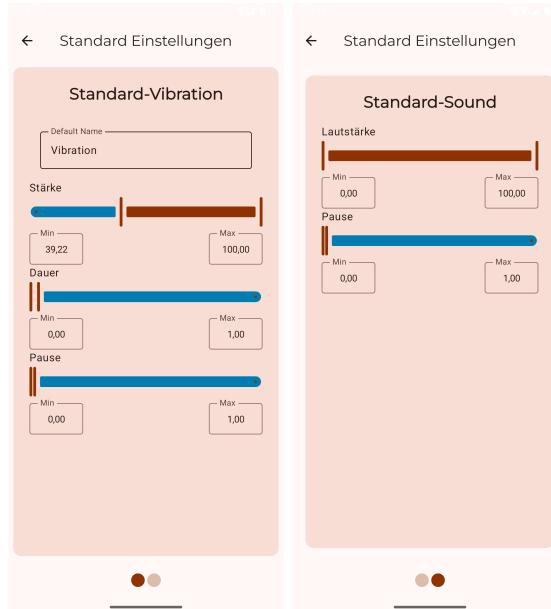
3.6.8.1 components

- **ConfigurationList** Enthält die Composables, aus der eine Karte für eine Konfiguration besteht.

3.6.9 settings

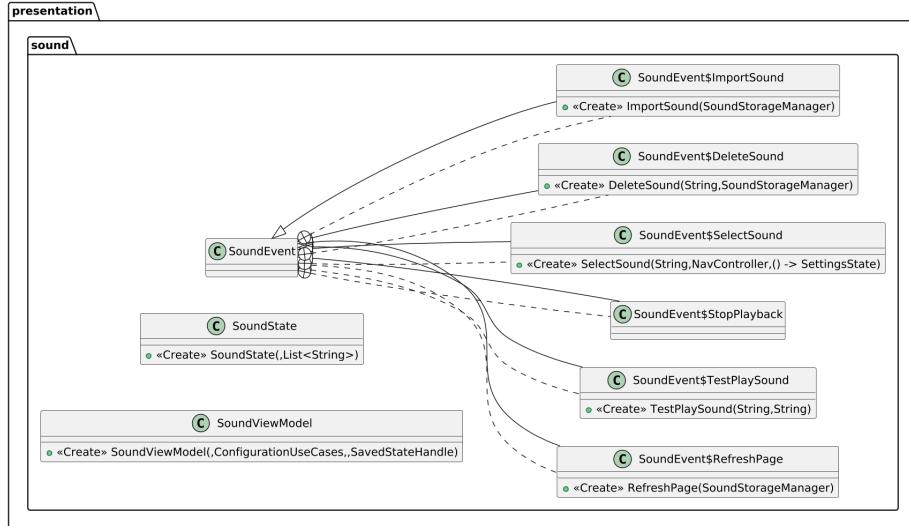


Enthält alle Klassen für die "Standard Einstellungen" Ansicht.

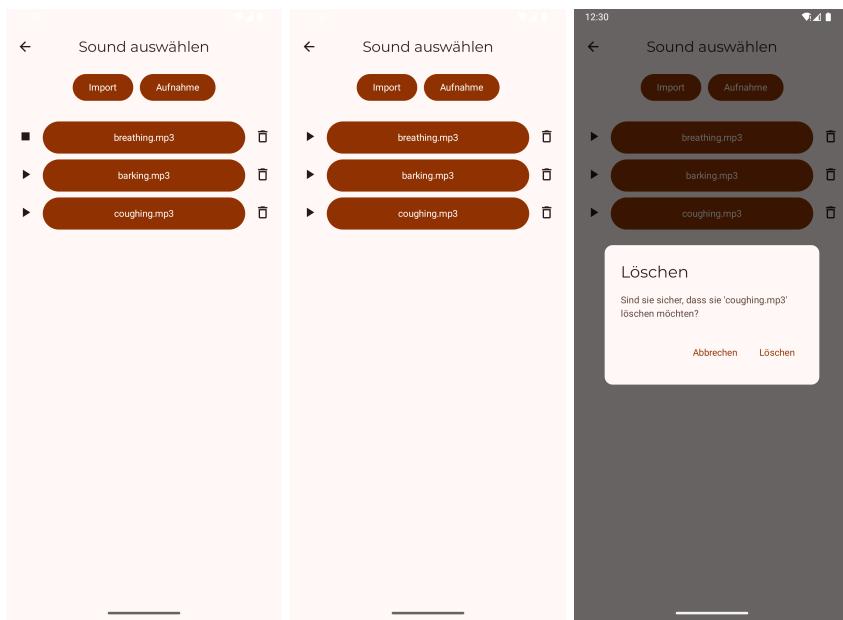


- **SettingsEvent** Definiert alle Events die in der Standard Einstellungen Ansicht auftreten können.
- **SettingsScreen** View für die Standard Einstellungen Ansicht, enthält alle Composable, aus denen diese Ansicht aufgebaut ist.
- **SettingsScreenState** Speichert den Zustand der Standard Einstellungen Ansicht.
- **SettingsViewModel** View Model der Standard Einstellungen Ansicht. Reagiert auf alle UI Events in dieser Ansicht.

3.6.10 sound



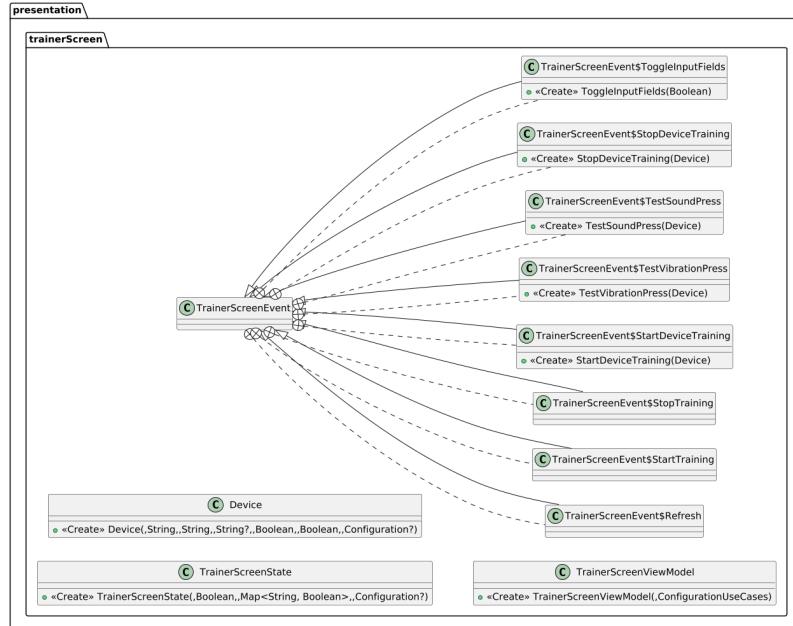
Enthält alle Klassen für die "Sound auswählen" Ansicht, in der der Nutzer entscheiden kann, welches Geräusch der Konfiguration hinzugefügt wird.



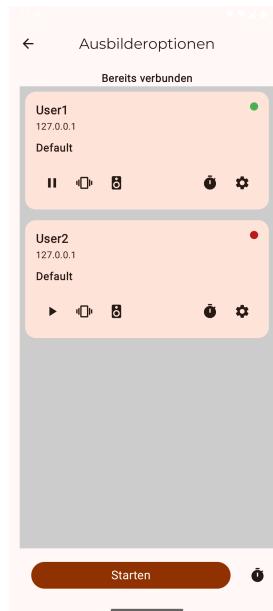
- **SingleSound** Composable um eine einzige Sound Datei anzuzeigen, zusammen mit einem Button zum abspielen, und einem zum löschen.
- **SoundDialog** Composable um einem neu aufgenommen Sound einen Namen zu geben.

- **SoundEvent** Definiert alle Events die in der Sound Ansicht auftreten können.
- **SoundScreen** View für die Sound Ansicht, enthält alle Composable, aus denen diese Ansicht aufgebaut ist.
- **SoundScreenState** Speichert den Zustand der Sound Ansicht.
- **SoundViewModel** View Model der Sound Ansicht. Reagiert auf alle UI Events in dieser Ansicht.

3.6.11 trainerScreen



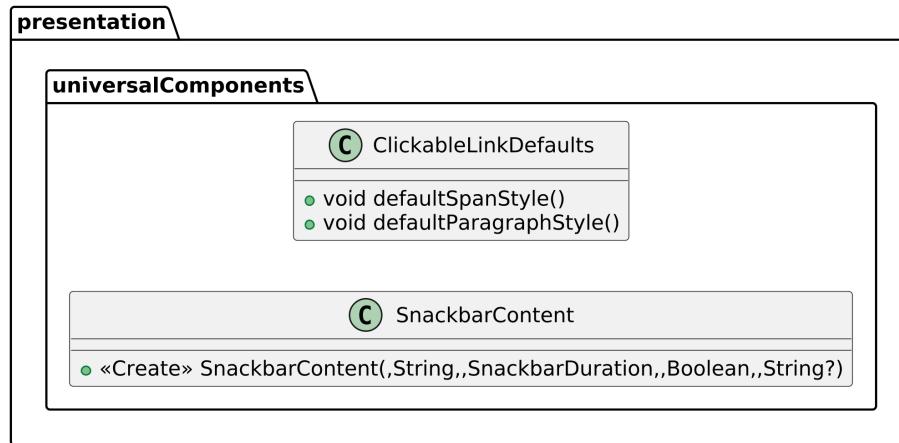
Enthält alle Klassen für die "Ausbilderoptionen" Ansicht. Von hier aus kann der Trainer die Remote Geräte fernsteuern.



- **DeviceCard** Karte, um ein einzelnes Gerät zu repräsentieren.
- **TrainerScreenEvent** Definiert alle Events die in der Hinzufügen Ansicht auftreten können.

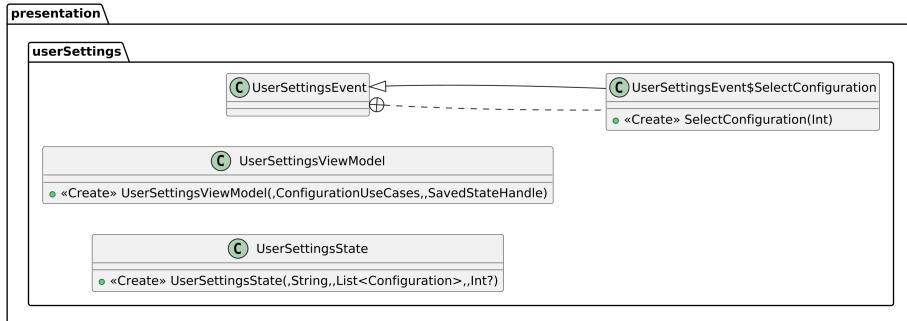
- **TrainerScreenScreen** View für die Hinzufügen Ansicht, enthält alle Composable, aus denen diese Ansicht aufgebaut ist.
- **TrainerScreenScreenState** Speichert den Zustand der Hinzufügen Ansicht.
- **TrainerScreenViewModel** View Model der Hinzufügen Ansicht. Reagiert auf alle UI Events in dieser Ansicht.

3.6.12 universalComponents

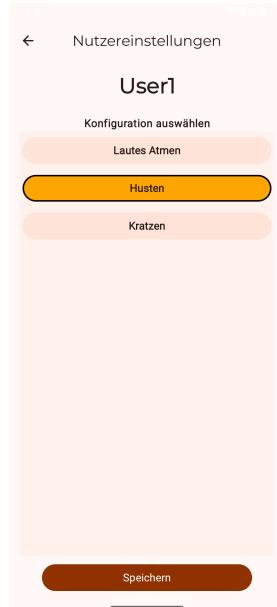


- **Clickable Link** Ein anklickbarer Link mit einem Standard aussehen.
- **ConfirmDeleteDialog** Dialog um zu bestätigen, dass etwas gelöscht werden soll.
- **SnackbarContent** Standard Snackbar für kurze Benachrichtigungen.
- **TobBar** Standard Top Bar die einen Zurück Knopf, einen Titel, und mögliche Sonderaktionen, z.B. Löschen, anzeigt.

3.6.13 userSettings

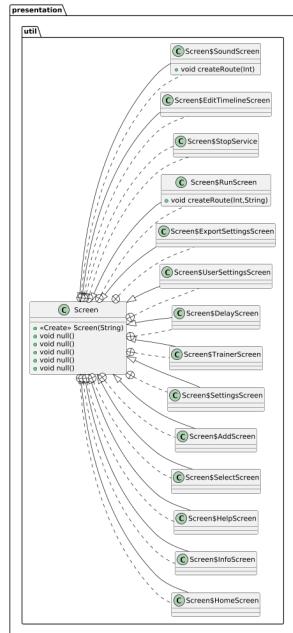


Enthält alle Klassen für die "Nutzereinstellungen" Ansicht. Hier kann der Ausbilder die Einstellungen für eine Remote Gerät ändern.



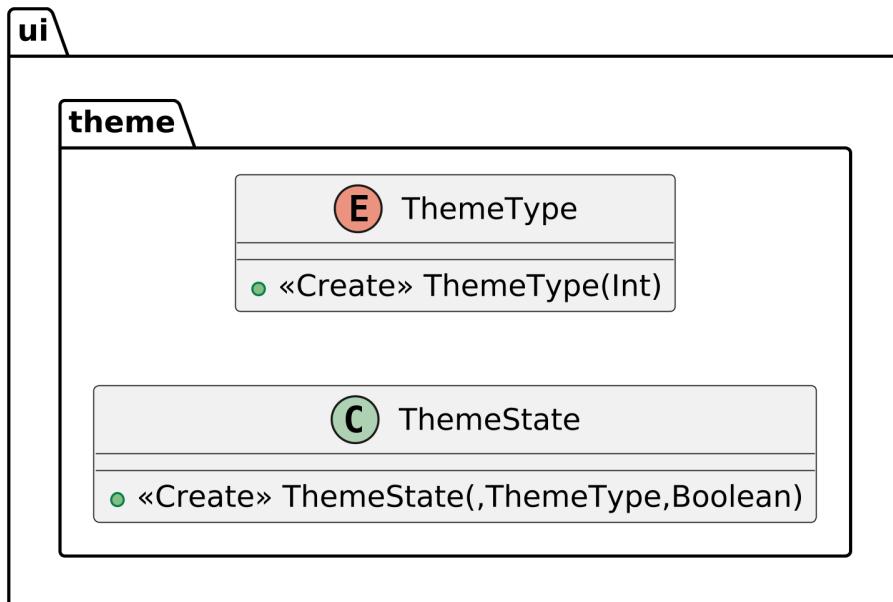
- **UserSettingsEvent** Definiert alle Events die in der Nutzereinstellungen Ansicht auftreten können.
- **UserSettingsScreen** View für die Nutzereinstellungen Ansicht, enthält alle Composable, aus denen diese Ansicht aufgebaut ist.
- **UserSettingsScreenState** Speichert den Zustand der Nutzereinstellungen Ansicht.
- **UserSettingsViewModel** View Model der Nutzereinstellungen Ansicht. Reagiert auf alle UI Events in dieser Ansicht.

3.6.14 util



- **BackPressGestureDisabler** Deaktiviert die Zurück Geste auf Handy mit Gesteuerung.
- **BigDecimal** Erlaubt es Numbers zu BigDecimal konvertiert zu werden, und fügt einige nützliche Funktionen zu BigDecimal hinzu.
- **Screen** Klasse um den Pfad zu Ansichten zu bekommen.
- **Snackbar** Definiert spezielle Snackbars, um das arbeiten mit ihnen zu erleichtern und sie zu vereinheitlichen.
- **VibrationAmplitudeControl** Eigene Funktionen um das abrieten mit Vibrationen zwischen verschiedenen API Versionen zu vereinfachen.

3.7 ui.theme



- **Color** Definiert alle in der App verwendete Farben.
- **Theme** Kombiniert mehre Farben zu verschiedenen Farbschemen.
- **ThemeState** Der aktuelle Themen Zustand, d.h. ob ds Farbschema Hell, Dunkel, oder Auto ist, und ob dynamische Farben an oder aus sind.
- **Type** Definiert die in der App verwendeten Schriftarten.

3.8 Nicht Kotlin Dateien

3.8.1 Assets

Zu finden unter android-app/app/src/main/assets

- **Sounds** Enthält die Standard Sound, welche in der App enthalten sind.

3.8.2 Res

Zu finden unter android-app/app/src/main/res/drawable.

- **Drawable** Enthält die meisten Icons der App in Form von Vektor Dateien. Enthält auch das ISP Logo.
- **Mipmap** Enthält verschiedene Versionen des App Logos in unterschiedlichen Auflösungen.
- **Values** Enthält die Werte für Farben, das Grund-Schema der App, Schriftartenzertificate, so wie die in der App verwendeten Strings.

- **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 Dateien außerhalb von App

4.1 ./projectmanagement

Enthält die Beschreibung und die Screenshots für den Google Play Store.

4.2 ./docs

4.2.1 LaTex/DeveloperDocumentation

Enthält die LaTex Dateien für diese Dokumentation. Die UML Diagramme wurden mithilfe eines Android Studio Plugins erzeugt.

4.2.2 fastlane-screenshots

Hier speichert Fastlane seine automatisch generierten Screenshots, welche in der Dokumentation oder anderweitig verwendet werden können. Alle Screenshots werden in verschiedenen Farbschemen, Zuständen und Sprachen generiert.

4.3 ./android-app

- **Gemfile** Enthält die Gems für Fastlane.
- **Gemfile.lock** Speichert die Version für alle nötigen Abhängigkeiten.

4.3.1 .run

Contains all run configurations for Android Studio.

- **Fastlane Screenshots** Eine Konfiguration um mit Fastlane neue Screenshots zu erzeugen. Zuvor ist 1.1 zu folgen.

4.3.2 fastlane

Enthält die Fastlane Dateien, um automatisch Screenshots zu erzeugen.

5 Tests

5.1 AndroidTest

Diese Tests testen die App durch Interaktionen mit der UI.

- **ExampleInstrumentedTest** Kurzer Beispiel eines instrumentierten Tests.
- **HiltTestRunner** Ersetzte die Application Klasse der echten Anwendung mit einer instrumentierten HiltTestApplication. Dies wird in einem build.gradle Skript spezifiziert.

5.1.1 di

Tests um das di Package (3.3) automatisch zu testen.

- **Test AppModule** Erstellt eine In-Memory Environment, insbesondere eine Datenbank und ein Repository, welche zwischen Tests schnell zurück gesetzt werden können.

5.1.2 endToEndTests

Diese tests testen einen kompletten Arbeitsvorgang, von Home Screen bis zum Ende. Dies ist ähnlich zu klassischen Integrationstests.

- **CreateAndPlayConfigurationTest** Testet von dem Start Bildschirm aus eine Konfiguration zu erstellen, sie zu bearbeiten und sie dann abzuspielen und zu stoppen.
- **SecondaryEndToEndTests** Tests Arbeitsabläufe, welche keine eigene Test Klasse rechtfertigen, wie das anordnen von Konfigurationskomponenten in der Zeitachse, oder das wechseln zwischen hellem und dunklem Farbschema.

5.1.3 presentation.editTimeline

Test um automatisch das editTimeline Package zu testen (3.6.3).

- **EditTimelineScreenTest** Testet das hinzufügen einer Komponente und das bearbeiten von Name und Beschreibung in der "Konfiguration bearbeiten" Ansicht.

5.1.4 screenshots

Test um mit Fastlane automatisiert Screenshots zu erstellen.

- **ScreenshotTests** Eine abstrakte Grundklasse um mit Fastlane automatisch Screenshots zu erstellen.

5.1.4.1 phone

- **PhoneScreenshotTests** Implementierung von ScreenshotTests um Screenshots auf einem Handy aufzunehmen.

5.2 Test

Diese Tests testen Funktion, welche nicht von Android abhängig sind.

5.2.1 domain.model

Diese Tests testen Funktionen aus dem model Package (3.4.1).

- **ConfigurationComponentRoomConverterTest** Überprüft, dass der ConfigurationComponentRoomConverter eine Konfigurations-Komponente korrekt in einen String überträgt, und anders herum.
- **RangeConverterTest** Testet den RangeConverter.
- **SoundConverterTest** Testet den SoundConverter.

5.2.2 presentation.delay

Diese tests testen Funktionen aus dem delay Package (3.6.2).

- **TimerTest** Testet, dass der DelayTimer einen String in die korrekte Verzögerung umwandelt.