Fraunhofer Institut für Optronik, Systemtechnik und Bildauswertung

Mario Kaufmann
Pascal Birnstill
Erik Krempel

Entwurf

Version 1.0

Privacy Crash Cam App für Android

Fabian Wenzel
Giorgio Gross
Christoph Hörtnagl
David Laubenstein
Josh Romanowski

17. Januar 2017

Inhaltsverzeichnis

1 Architektur **6**
1.1 Modularisierung . . . 6
1.2 Trennung in Komponenten . . . 6
1.3 Model View Presenter (MVP) . . . 6
1.4 REST-Architektur . . . 8
1.5 Entwurfsmuster . . . 8

2 App **9**
2.1 Architektur . . . 9
2.1.1 Entwurfsmuster . . . 9
Decorator . . . 10
Observer . . . 10
Command . . . 10
Proxy . . . 11
Template Method . . . 11
Strategy . . . 11
2.2 Datenhaltung . . . 12
2.2.1 Nutzerdaten . . . 12
2.2.2 Dateien . . . 12
2.3 Modulübersicht . . . 14
2.3.1 Gui . . . 14
2.3.2 ApplicationLogic . . . 15
2.3.3 Utils . . . 15
2.3.4 Data . . . 15
2.4 Klassenübersicht . . . 16
2.4.1 Gui . . . 16
«Abstract» MainActivity . . . 16
«Abstract» ContainerActivity . . . 17
LogInActivity . . . 18
SettingsActivity . . . 19
LegalActivity . . . 20
VideosActivity . . . 21
CameraActivity . . . 22
2.4.2 ApplicationLogic . . . 24
LogInFragment . . . 24
LogInHelper . . . 26
SettingsFragment . . . 27

LegalFragment . . . 29
VideosFragment . . . 30
CameraView . . . 31
2.4.3 ApplicationLogic.Camera . . . 33
«Interface» CameraHandler . . . 33
CompatCameraHandler . . . 34
TriggeringCompatCameraHandler . . . 38
«Interface» IRecordCallback . . . 40
2.4.4 Utils . . . 41
Encryptor . . . 41
«Interface» IFileEncryptor . . . 42
AESEncryptor . . . 43
«Interface» IKeyEncryptor . . . 44
RSAEncryptor . . . 45
Ringbuffer<E> . . . 46
AsyncPersistor . . . 47
«Interface» IPersistCallback . . . 49
2.4.5 Data . . . 50
Metadata . . . 50
Video . . . 51
Account . . . 52
Settings . . . 53
MemoryManager . . . 54
ServerProxy . . . 57
«Interface» IServerResponseCallback . . . 58
AuthenticateTask . . . 59
VideoUploadTask . . . 60

3 Web-Dienst 62
3.1 Architektur . . . 62
3.1.1 Multithreading . . . 62
3.1.2 Entwurfsmuster . . . 63
Proxy . . . 63
Master-Worker . . . 63
Pipeline . . . 63
Strategie . . . 64
3.2 Datenhaltung . . . 65
3.2.1 Datenbank . . . 65
3.2.2 Temporäre Dateien . . . 65
3.2.3 Verwendete Resourcen . . . 66
3.3 Modulübersicht . . . 67
3.3.1 Server . . . 67
3.3.2 Data . . . 67
3.3.3 Manager . . . 67

3.3.4 VideoProcessing . . . 68
3.3.5 VideoProcessing.Chain . . . 68
3.4 Klassenübersicht . . . 69
3.4.1 Server . . . 69
Main . . . 69
ServerProxy . . . 71
3.4.2 Data . . . 74
DatabaseManager . . . 74
Account . . . 77
Metadata . . . 78
VideoInfo . . . 79
LocationConfig . . . 80
3.4.3 Manager . . . 81
AccountManager . . . 81
VideoManager . . . 83
3.4.4 VideoProcessing . . . 85
VideoProcessingManager . . . 85
VideoProcessingChain . . . 87
EditingContext . . . 89
«Interface» IStage . . . 90
3.4.5 VideoProcessing.Chain . . . 91
Decryptor . . . 91
«Interface» IKeyDecrytor . . . 92
RSADecryptor . . . 93
«Interface» IFileDecryptor . . . 94
AESDecryptor . . . 95
«Abstract» AAnonymizer . . . 96
Anonymizer . . . 97
«Interface» IAnalyzer . . . 98
ExampleAnalyzer . . . 99
«Interface» IFilter . . . 100
ExampleFilter . . . 101
Persistor . . . 102

4 Web-Interface 103
4.1 Architektur . . . 103
4.1.1 Entwurfsmuster . . . 103
Proxy . . . 103
Zustandsmuster . . . 104
Schablonenmethode . . . 104
4.2 Modulübersicht . . . 105
4.2.1 Gui . . . 105
4.2.2 Gui.Navigation . . . 105
4.2.3 DataManagement . . . 105

4.2.4 ServerConnection . . . 105
4.2.5 MailService . . . 105
4.3 Klassenübersicht . . . 106
4.3.1 Gui . . . 106
MyUI . . . 106
LoginView . . . 108
VideoView . . . 110
AccountView . . . 111
ImpressumView . . . 113
PrivacyView . . . 114
VideoTable . . . 115
4.3.2 Gui.Navigation . . . 116
Menu . . . 116
4.3.3 DataManagement . . . 117
AccountDataManager . . . 117
Account . . . 119
VideoDataManager . . . 120
Video . . . 122
4.3.4 ServerConnection . . . 123
ServerProxy . . . 123
4.3.5 MailService . . . 125
MailService . . . 125

5 Anhang 126
5.1 Sequenzdiagramme . . . 126

1 Architektur

1.1 Modularisierung

An vielen Stellen der Privacy-Crash-Cam bietet sich eine Modularisierung an. D.h. in sich geschlossene Programmabschnitte werden voneinander getrennt und kommunizieren über vereinbarte Schnittstellen (1.4). Dies bietet eine Reihe von Vorteilen:

- Reduziert die Koordination und Kommunikation und verringert dadurch die Komplexität.
- Höhere Flexibilität, da nur einzelne Module an neue Bedingungen (z.B. andere Plattformen) angepasst werden müssen.
- Höhere Wiederverwertbarkeit, da einzelne Module in neue Systeme eingebunden werden können.
- Bessere Austauschbarkeit, da einfach einzelne Module ausgetauscht werden können, solange sie die Schnittstellen beachten.
- Schnellere Fehlersuche, da der Suchbereich eingeschränkt werden kann.

In der Privacy-Crash-Cam findet eine Modularisierung durch vier Maßnahmen statt. 1. Trennung in Komponenten, 2. Model View Presenter (MVP), 3. Modularisierung innerhalb der Komponenten ((2.3), (3.3), (4.2)) und 4. Entwurfsmuster

1.2 Trennung in Komponenten

Die Privacy-Crash-Cam ist in drei Komponenten aufgeteilt. Die App, den Web-Dienst und das Web-Interface.
Da die App und die Webanwendung auf unterschiedlichen Technologien basieren, ist es natürlich diese Komponenten getrennt zu entwickeln. Zudem entsteht eine erhöhte Flexibilität, da so z.B. erlaubt wird die App auch für andere Betriebssysteme zu entwickeln als Android, ohne die Webanwendung zu verändern.
Zusätzlich wird die Webanwendung in Web-Dienst und Web-Interface unterteilt, um die logisch voneinander unabhängigen Komponenten App und Web-Interface zu trennen.

1.3 Model View Presenter (MVP)

Beim Design der Privacy-Crash-Cam wird schnell deutlich, dass eine strikte Trennung zwischen Datenbank, Applikationslogik und Benutzeroberfläche von Vor-

teil ist. Das Model-View-Presenter-Prinzip (MVP) realisiert diese Trennung und wird in der App und in der Webanwendung eingesetzt.

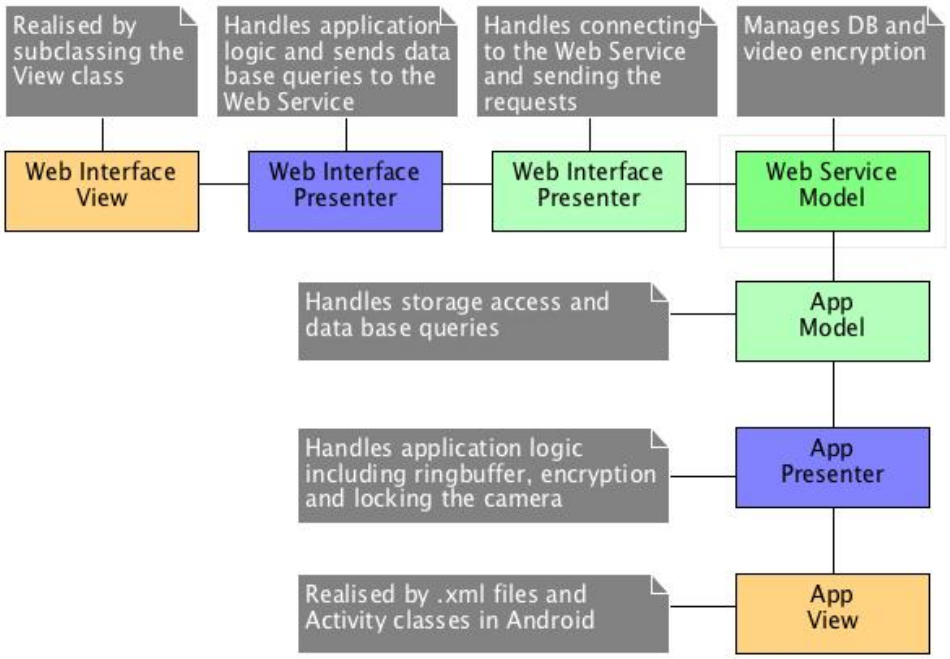

Abbildung 1.1: MVP in der Privacy Crash Cam

Sowohl App als auch Web-Interface besitzen ein eigenes View-Modul. Dieses ist für die graphische Benutzeroberfläche zuständig. In der App wird die View durch XML-Dokumente und Activities implementiert, die das Layout definieren. Das Web-Interface verwendet dafür die von Vaadin bereitgestellte View-Klasse, von der man eigene Klassen ableiten kann.

Die Applikationslogik wird durch die Presenter-Ebene umgesetzt. App und Web-Interface besitzen eigene Presenter-Module, die an die jeweilige Plattform angepasst sind. Der Presenter handhabt Eingaben durch Nutzer oder Sensoren und koordiniert die darauf folgenden Aktionen, wie das Aktualisieren der Ansichten und Änderungen des Models.

Das Model ist dafür zuständig, Daten zu verwalten und zu bearbeiten. Die App hat dafür ein eigenes Modul, das den Speicherzugriff regelt. Das Model des Web-Interfaces ist der Web-Dienst, der alle Nutzer- und Videodaten verwaltet.

1.4 REST-Architektur

Der Web-Dienst stellt bestimmte Funktionalitäten für die App und das Web-Interface bereit. Um diese Funktionen auszuführen, haben wir den Web-Dienst mit einer REST-Architektur versehen. Das bedeutet, dass wir die Prinzipien dieser Architektur in dem Web-Dienst verwirklicht haben.

Das Client-Server Prinzip besagt zum einen, das unser Web-Dienst Funktionen bereit stellt, welche über die Clients angefragt werden können. Eine weiteres Prinzip der REST-Architektur ist den Clients nur eine Schnittstelle zum Server anzubieten. Somit sind die dahinter liegenden Schichten für App und Web-Interface verborgen. Diese einheitliche Schnittstelle lässt sich via URLs von den Clients aus erreichen.

Die Clients versenden POST- und GET-Anfragen an den Server, welcher die REST-API verwendet, um Daten zurückzugeben (GET) oder zu sichern (POST). Dabei wird die URL ausgelesen, die Informationen und Daten über Hypermedia beinhalten kann. Diese werden für die Funktion verwendet und an die Module des Servers weitergeleitet.

1.5 Entwurfsmuster

Der Einsatz weiterer Enwurfsmuster innerhalb der Module erleichtert die Lesbarkeit und begünstigt die Flexibilität des Codes noch weiter. Eingesetzte Muster sind in den entsprechenden Kapiteln für App (2), Web Interface (4) und Web Dienst (3) genauer beschrieben.

2 App

2.1 Architektur

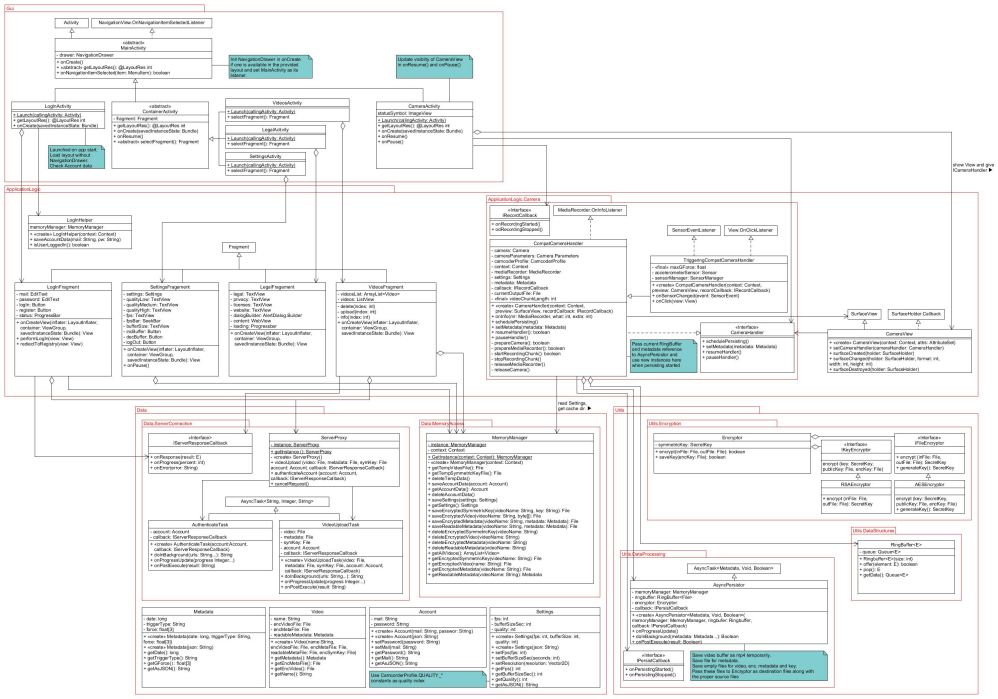

Abbildung 2.1: UML Diagramm der Android App

2.1.1 Entwurfsmuster

In der App werden Entwurfsmuster verwendet, um Komponenten leicht austauschen zu können und um die Nutzeroberfläche und den Funktionsumfang einfach erweitern zu können. Der Einsatz von Entwurfsmustern Verhindert zudem, dass Klassen über Schichten hinweg Zugriff haben und reduziert die Aufrufe von Klassen einer niederen Schicht auf die darüber liegende Schicht. Sie helfen ferner bei der Enkopplung einzelner Komponenten voneinander und unterstützen das Geheimnisprinzip.

Decorator

Das Decorator Muster wird bei der Klasse TriggeringCompatCameraHandler angewendet. Diese dekoriert die Klasse CompatCameraHandler, indem er von dieser erbt und in seinem Konstruktor einen *super* Aufruf tätigt, um den Konstruktor der Elternklasse aufzurufen. Der Decorator delegiert jeden Methodenaufruf an seine Elternklasse, fügt aber Funktionen zum Behandeln von Klick-Events und dem Überwachen des G-Sensors hinzu. Dadurch wird die Funktion des CompatCameraHandlers erweitert, ohne dass er davon wissen muss.

Observer

Das Observer Muster reduziert die Aufrufe von Klassen einer niederen Schicht auf die darüber liegende Schicht. In der App gibt es die Observer «Interface» IRecordCallback, das «Interface» IPersistCallback und das «Interface» IServerResponseCallback. Nützlich ist das Observer Muster vor allem dann, wenn ein asynchroner Task ausgeführt wird und der Aufrufer erst benachrichtigt werden soll, wenn dieser Task zu Ende ist.
Bei der Umsetzung des Observer Musters weicht die App von der herkömmlichen Art ab: Während normalerweise eine Klasse, die als Observer agiert, das Interface, also einen der Callbacks, implementiert, hat in der App im Gegensatz dazu eine Klasse ein Attribut vom Typ des jeweiligen Interfaces. Bei der Instanziierung dieses Attributes findet die Implementierung des Interfaces statt.

Command

Das Command Muster kommt bei jedem asynchronen Task zum Einsatz. Er ermöglicht die Abkopplung der Hintergrundprozesse von den Vordergrundprozessen. Somit kann der Aufrufer direkt mit seinen Aufgaben fortfahen, während der Command im Hintergrund ausgeführt wird. Sobald er seine Aufgaben beented hat wird der Empfänger des Commands benachrichtigt und kann damit beginnen, das Resultat des Commands zu verarbeiten.
In den Konstruktoren der Commands werden diese parametrisiert, durch die *start* Methode wird der Command ausgeführt. Der AuthenticateTask authentifiziert den Nutzer, indem er auf das Netzwerk zurückgreift. Ebenso greift der VideoUploadTask auf das Netzwerk zu um ein Video hochzuladen. Der AsyncPersistor speichert ein Video asynchron ab. Jeder dieser Commands erhält in seinem Konstruktor und durch die *start* Methode alle Daten, die er zur Ausführung benötigt. Das Callback, welches jeder Command übergeben bekommt, dient als Empfänger, die Rolle des Invokers und des Klientens werden beide von der Klasse übernommen, die den Command instanziiert.

Proxy

Das Proxy Muster kommt zum Einsatz, sobald Anfragen an den Server gesendet werden oder Operationen auf dem Speicher ausgeführt werden sollen. Die entsprechenden Klassen sind ServerProxy und MemoryManager. Dadurch muss keine der Klassen, die den ServerProxy oder den MemoryManager verwenden, über die zugrunde liegende Infrastruktur und deren Handhabung Bescheid wissen.

Template Method

Das Template Method Muster, zu Deutsch "Schablonenmethode", findet seinen Einsatz in der «Abstract» ContainerActivity. Die ContainerActivity ist eine Activity, die immer nach dem gleichen Schema aufgebaut ist: Eine Toolbar am oberen Bildschirmrand und ein Fragment mit dem Inhalt darunter. Sie selbst lädt immer die gleiche Ansicht, lässt aber ihre Unterklasse bestimmen, welches Fragment angezeigt wird. Dadurch können ohne großen Aufwand weitere Ansichten eingefügt werden. Der einzige Aufwand besteht im Erben von ContainerActivity und dem Implementieren der *selectFragment* Methode.
Sollte später eine Activity mit einem ViewPager eingefügt werden, der beispielsweise aufgenommene und heruntergeladene Videos in eigenen Tabs anzeigen kann, bietet sich die Schablonenmethode wieder an: In diesem Fall gibt es eine Acitivty, die ein Layout lädt, das über einen ViewPager verfügt. Die Unterklassen bestimmen dann auf welcher Seite welches Fragment eingeblendet wird. Natürlich muss die Nummer des aktuellen Tabs des ViewPagers in der *selectFragment* Methode mitgeliefert werden.

Strategy

Die Strategy Methode kommt in den Klassen «Interface» CameraHandler, «Interface» IFileEncryptor und «Interface» IKeyEncryptor zum Einsatz. Sie ermöglicht die Austauschbarkeit der Kameraimplementierung bzw. der Verschlüsselungsalgorithmen und macht ihre Klienten nur von den Schnittstellen abhängig.

2.2 Datenhaltung

Die App muss sich um die Verwaltung von Accountdaten, Einstellungen, Videos, Metadaten und symmetrischen Schlüsseln kümmern.

2.2.1 Nutzerdaten

Für Accountdaten und Einstllungen bieten sich die von Android bereitgestellten SharedPreferences an, die als Key-Value-Pairs aufgebaut werden. Das Schema ist in Schaubild 2.2 veranschaulicht. Der Key entspricht den Daten die abgefragt werden sollen, also Einstellungen oder Account. Die gelesenen Werte bestehen jeweils aus einem String im JSON Format. Dadurch erreicht man eine Bündelung aller zusammengehörender Werte unter einem Key, unterstützt die Änderbarkeit und Ergänzbarkeit bestehender Daten und vereinfacht die Instanziierung von Account- und Einstellungen-Objekten durch einen Konstruktor, der einen JSON String entgegennimmt.

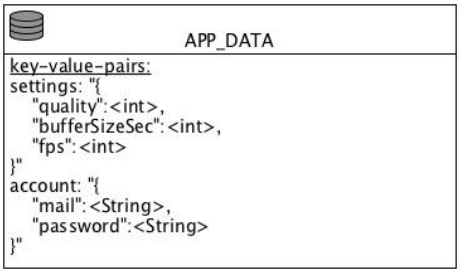

Abbildung 2.2: Key-Value-Pairs der SharedPreferences

2.2.2 Dateien

Für die Speicherung von Videos, Metadaten und symmetrischen Schlüsseln erweisen sich die SharedPreferences jedoch als ungeeignet. Hier gibt es zwei Lösungen: Entweder man speichert sie im internen oder im externen Speicher. Wir werden die erste Variante wählen, werden uns aber die Möglichkeit zum Exportieren der Daten in den externen Speicher frei halten. Der naive Ansatz merkt sich in den oben beschriebenen SharedPreferences die Namen aller gespeicherten Dateien und referenziert sie dadurch. Dabei ergibt sich aber folgendes Problem: Nachdem der Nutzer alle appinternen Daten gelöscht hat, hat man keine Chance mehr an extern gespeicherte Daten zu kommen.

Wir verwenden einen generischeren Ansatz, der gänzlich auf die verwendung der SharedPreferences verzichtet: Es existieren drei Ordner: Ein Video-, ein

Metadata- und ein Keyordner. Jedes Video erhält einen eindeutigen Namen, bestehend aus der exakten Aufnahmezeit. Jede mit diesem Video verwandte Datei, also dessen Metadaten und Schlüssel, besitzen den gleichen Dateinamen.

Bezüglich der Metadaten muss auch beachtet werden, dass sie über die App ausgelesen werden können müssen, sie jedoch beim Speichervorgang des Videos direkt verschlüsselt werden um sie beim Hochladen des Videos verschlüsselt übertragen zu können. Aus diesem Grund werden die Metadaten in zwei Inhaltlich identische Dateien abgelegt. Eine der beiden wird veschlüsselt, die Andere nicht. An den Dateinamen der unverschlüsselten Datei wird zur identifikation schließlich "_readable" angehängt

Zusätzlich zu den erwähnten Ordnern existiert ein weiterer Ordner, der verwendet wird, um temporäre Videodateien abzulegen. Unter einer temporäre Videodatei ist hier ein unverschlüsseltes Video zu verstehen, welche nur zwischengespeichert wird. Android bietet hierzu die Möglichkeit, die temporäre Datei im appinternen Cache-Ordner abzulegen. Dies bietet außerdem den Vorteil, dass nur die App selbst Zugriff darauf hat.

2.3 Modulübersicht

Die App besteht aus fünf Modulen, die sich gemäß MVP einer der Rollen Model, View oder Presenter zuordnen lassen. Dabei unterteilen sich komplexere Module in weitere Unterpakete. Die View-Rolle übernimmt das Gui Modul (2.3) zusammen mit den XML-Layout Dateien. Die Presenter-Rolle wird durch das ApplicationLogic Modul (2.3.1) realisiert. Das Utils-Modul (2.3.2) unterstützt es dabei mit grundlegenden Operationen und Datenstrukturen. Als Model agiert das Modul Data (2.3.3). Schaubild 3.3 veranschaulicht die Einteilung:

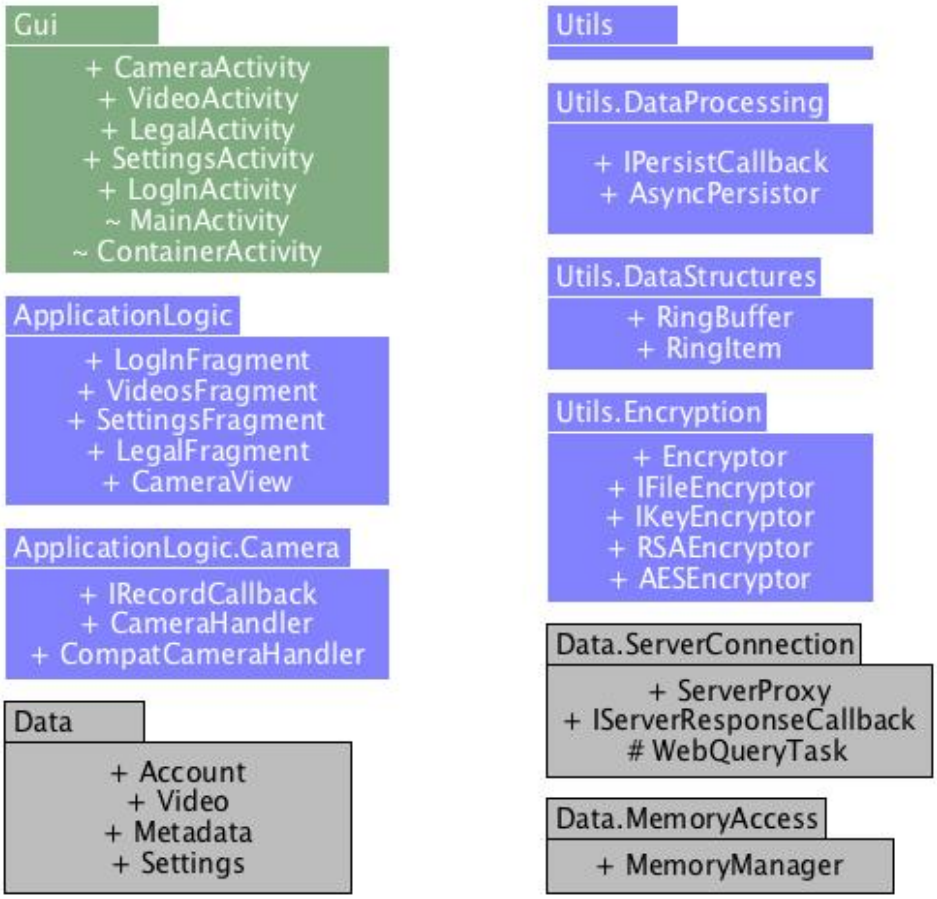

Abbildung 2.3: Module der Android App

2.3.1 Gui

Das Paket Gui binhaltet alle Activites der Android App. Hier findet lediglich die Anbindung der XML-Dateien der Activity-Layouts, sowie die Menüführung

statt. Konkrete Inhalte werden durch Klassen anderer Pakete geladen. Activities sind zudem der Haupteinstiegspunkt für bestimmte Aktionen, insbesondere wird die LogInActivity beim Appstart aufgerufen.

2.3.2 ApplicationLogic

Im Paket ApplicationLogic befinden sich sämtliche Fragments, sowie die **??**. Diese Komponenten stehen in direkter Verbindung zur Benutzeroberfläche und reagieren auf Nutzereingaben. In diesem Paket befindet sich ebenfalls das Unterpaket ApplicationLogic.Camera, welches Klassen zur Instanziierung und Kontrolle der Kamera beinhaltet. Die Klasse CameraView befindet sich nicht in diesem Unterpaket, da sie eine SurfaceView ist und, wie die Fragments und Activities, direkt vom Nutzer zu sehen ist. ApplicationLogic.Camera beinhaltet nur die Klassen, die auch von anderen Komponenten verwendet werden können, um auf die Kamera Zugriff zu erhalten.

2.3.3 Utils

Das Utils Paket ist in drei Unterpakete aufgeteilt: Utils.DataStructures enthält Datenstrukturen, wie den Ringbuffer, sowie generische Klassen für dessen Elemente. Utils.DataProcessing beinhaltet Klassen die notwendig sind, um Inhalte der Datenstrukturen in den Speicher des Geräts zu übertragen. Hierbei handelt es sich lediglich um die dafür erforderliche Logik. Die Speicherung übernimmt schließlich das MemoryAccess Modul. Utils.Encryption bietet Funktionen, um Dateien hybrid zu verschlüsseln.

2.3.4 Data

Das Paket Data beinhaltet die Unterpakete MemoryAccess und ServerConnection, die für die Datenabfrage und -manipulation zuständig sind.
Das Paket MemoryAccess regelt Zugriffe auf den internen Speicher des Android Geräts. Dazu gehört neben dem Speichern von Dateien auch der Zugriff auf die von Android bereitgestellten SharedPreferences. Ebenfalls enthalten sind Klassen zur Datenbündelung: Account, Video, Metadata und Settings.
Das ServerConnection Paket enthält alle Klassen die notwendig sind, um Anfragen an den Web Dienst zu senden und dessen Antwort zu empfangen. Dazu gehört der entsprechende Proxy sowie die Klassen zur Herstellung der Verbindung und Weiterreichen der Antwort an die aufrufende Klasse.

2.4 Klassenübersicht

2.4.1 Gui

«Abstract» MainActivity

extends Activity
Die MainActivity bildet die Elternklasse aller weiteren Activities und übernimmt als solche die Navigation durch das Menü. Sie agiert dabei als OnNavigationItemSelectedListener, implementiert also dessen Methoden.

Attribute

- **drawer: NavigationDrawer**
  Sichtbarkeit private
  Die NavigationDrawer-Instanz, in der das Menü angezeigt wird.

Konstruktoren
Keine, da der Lebenszyklus dieser Klasse von Android gesteuert wird.

Methoden

- **onCreate (savedInstanceState: Bundle): void**
  Sichtbarkeit public
  Überschreibt die onCreate Methode von Activity. Ruft die Schablonenmethode getLayoutRes() auf und initialisiert daraufhin den NavigationDrawer und die Toolbar und setzt sich selbst als OnNavigationItemSelectedListener.
- **«abstract» getLayoutRes (): int**
  Sichtbarkeit public
  Schablonenmethode,in der die Unterklasse das zu ladende Layout bestimmt.
- **onNavigationItemSelected(item: MenuItem): boolean**
  Sichtbarkeit public
  Implementiert das OnNavigationItemSelectedListener Interface und wechselt immer wenn ein Menüeintrag, also Kamera, Videos, Einstellungen, Impressum oder Datenschutz, angeklickt wurden zu der jeweiligen Ansicht.

«Abstract» ContainerActivity

extends «Abstract» MainActivity
Die ContainerActivity ist die Elternklasse aller Activities, die eine Toolbar und ein Fragment anzeigen. Das Fragment wird dynamisch geladen, je nach dem welche Ansicht der Nutzer sehen möchte. Die ContainerActivity lädt immer das gleiche Layout und zeigt in dessen Container das Fragment an, welches sie über die Schablonenmethode selectFragment() abgefragt hat.

Attribute

- **fragment: Fragment**
  Sichtbarkeit private
  Fragment welches aktuell angezeigt werden soll.

Konstruktoren
Keine, da der Lebenszyklus dieser Klasse von Android gesteuert wird.

Methoden

- **getLayoutRes (): int**
  Sichtbarkeit public
  Überschreibt die getLayoutRes() Methode der Superklasse.
- **onCreate (savedInstanceState: Bundle): void**
  Sichtbarkeit public
  Überschreibt die onCreate() Methode der Superklasse. Ruft die Methode selectFragment() auf und zeigt das erhaltene Fragment an.
- **onResume (): void**
  Sichtbarkeit public
  Überschreibt die *onResume* Methode der Superklasse. Invalidiert das Fragment.
- **«abstract» selectFragment (): Fragment**
  Sichtbarkeit public
  Schablonenmethode, welche die Unterklasse das Fragment wählen lässt, das angezeigt werden soll.

LogInActivity

extends «Abstract» MainActivity
Die LogInActivity behandelt den Anmeldeprozess und ist die erste Activity die gestartet wird. Sie prüft zuerst, ob der Nutzer bereits vorher seine Anmeldedaten eingegeben hat und reagiert dementsprechend in ihrer *onCreate* Methode.

Attribute

- **logInFragment: LogInFragment**
  Sichtbarkeit private
  Die LogInFragment Instanz, die angezeigt wird.
- **logInHelper: LogInHelper**
  Sichtbarkeit private
  Die LogInHelper Instanz, mit der Accountdaten gespeichert werden und mit der überprüft wird, ob ein Nutzer angemeldet ist.

Konstruktoren
Keine, da der Lebenszyklus dieser Klasse von Android gesteuert wird.

Methoden

- **Launch (callingActivity: Activity): void**
  Sichtbarkeit public
  statisch

  Startet ein neues Intent durch welches eine LogInActivity Instanz erzeugt und gestartet wird.
- **getLayoutRes (): int**
  Sichtbarkeit public
  Überschreibt die getLayoutRes() Methode der Superklasse.
- **onCreate (savedInstanceState: Bundle): void**
  Sichtbarkeit public
  Überschreibt die onCreate() Methode der Superklasse. Prüft ob der Nutzer angemeldet ist und startet die CameraActivity falls ja. Falls nicht wird das LogInFragment angzeigt.

SettingsActivity

extends «Abstract» ContainerActivity
Die SettingsActivity zeigt das SettingsFragment an.

Attribute

- **settingsFragment: SettingsFragment**
  Sichtbarkeit private
  Die SettingsFragment Instanz.

Konstruktoren
Keine, da der Lebenszyklus dieser Klasse von Android gesteuert wird.

Methoden

- **Launch (callingActivity: Activity): void**
  Sichtbarkeit public
  statisch

  Startet ein neues Intent durch welches eine SettingsActivity Instanz erzeugt und gestartet wird.
- **selectFragment (): Fragment**
  Sichtbarkeit public
  Überschreibt die selectFragment() Methode der Superklasse.

LegalActivity

extends «Abstract» ContainerActivity
Die LegalActivity zeigt das LegalFragment an.

Attribute

- **legalFragment: LegalFragment**
  Sichtbarkeit private
  Die LegalFragment Instanz.

Konstruktoren
Keine, da der Lebenszyklus dieser Klasse von Android gesteuert wird.

Methoden

- **Launch (callingActivity: Activity): void**
  Sichtbarkeit public
  statisch

  Startet ein neues Intent durch welches eine LegalActivity Instanz erzeugt und gestartet wird.
- **selectFragment (): Fragment**
  Sichtbarkeit public
  Überschreibt die selectFragment() Methode der Superklasse.

VideosActivity

extends «Abstract» ContainerActivity
Die VideosActivity zeigt das VideosFragment an.

Attribute

- **videosFragment: VideosFragment**
  Sichtbarkeit private
  Die VideosFragment Instanz.

Konstruktoren
Keine, da der Lebenszyklus dieser Klasse von Android gesteuert wird.

Methoden

- **Launch (callingActivity: Activity): void**
  Sichtbarkeit public
  Bekommt die Activity übergeben, von der aus der Aufruf ausgeht. Startet ein neues Intent durch welches eine VideosActivity Instanz erzeugt und gestartet wird.
- **selectFragment (): Fragment**
  Sichtbarkeit public
  Überschreibt die selectFragment() Methode der Superklasse.

CameraActivity

extends «Abstract» MainActivity
Die CameraActivity zeigt die CameraView an, instanziiert eine CompatCameraHandler Instanz, sowie eine «Interface» IRecordCallback Instanz und manipuliert die graphische Nutzeroberfläche abhängig von den Methoden, die auf der IRecordCallback Instanz aufgerufen werden. Nach dem Start blendet die CameraActivity ein Symbol ein, welches die Bereitschaft der App signalisisert. Die CameraActivity stellt einen Observer des CompatCameraHandler dar.

Attribute

- **statusSymbol: ImageView**
  Sichtbarkeit private
  ImageView die verwendet wird, um das Symbol einzublenden, welches die Bereitschaft bzw. die Aufnahme der App signalisiert.
- **recordCallback: «Interface» IRecordCallback**
  Sichtbarkeit private
  Implementiert das IRecordCallback Interface. Der Aufruf der Mehtode onRecordStarted() benachrichtigt die CameraActivity Instanz über den Start der Videoaufnahme. Sie blendet das Symbol ein, welches die Aufnahme signalisiert. Das Symbol, das zuvor die Bereitschaft der App signalisiert hat, wird ausgeblendet. Der Aufruf der Methode onRecordStopped() benachrichtigt die CameraActivity Instanz über das Ende der Videoaufnahme. Sie blendet das Symbol ein, das die Bereitschaft signalisiert. Das Symbol, welches das die Aufnahme der App signalisiert hat, wird ausgeblendet.
- **cameraHandler: CompatCameraHandler**
  Sichtbarkeit private
  CameraHandler Instanz, die die Eingabedaten der Kamera eigenständig verarbeitet und die Aufnahme auslößt. Diesem Feld wird eine Triggering-CompatCameraHandler Instanz zugewiesen.
- **cameraView: CameraView**
  Sichtbarkeit private
  CameraView Instanz, welche verwendet wird, um die Kameravorschau anzuzeigen.

Konstruktoren
Keine, da der Lebenszyklus dieser Klasse von Android gesteuert wird.

Methoden

- **Launch (callingActivity: Activity): void**
  Sichtbarkeit public
  statisch

Startet ein neues Intent durch welches eine CameraActivity Instanz erzeugt und gestartet wird.

- **getLayoutRes (): int**
  Sichtbarkeit public
  Überschreibt die getLayoutRes() Methode der Superklasse.
- **onCreate (savedInstanceState: Bundle): void**
  Sichtbarkeit public
  Überschreibt die onCreate() Methode der Superklasse. Lädt die CameraView Instanz und erstellt die IRecorderCallback und CompatCameraHandler Instanzen.
- **onResume (): void**
  Sichtbarkeit public
  Überschreibt die onResume() Methode der Superklasse. Ruft die Methode setVisibility(..) der CameraView Instanz auf und übergibt den Parameter *View.VISIBLE*.
- **onPause (): void**
  Sichtbarkeit public
  Überschreibt die onPause() Methode der Superklasse. Ruft die Methode setVisibility(..) der CameraView Instanz auf und übergibt den Parameter *View.GONE*.

2.4.2 ApplicationLogic

LogInFragment

extends Fragment
Das LogInFragment lädt die einzelnen Layout-Komponenten der Anmeldeansicht und reagiert auf Nutzereingaben. Es steuert den Kontrollfluss des Anmeldevorgangs und aktualisiert die Bedienoberfläche dementsprechend.
Siehe auch: Anmelden in der App

Attribute

- **mail: EditText**
  Sichtbarkeit private
  Eingabefeld für die Email-Adresse des Nutzers.
- **password: EditText**
  Sichtbarkeit private
  Eingabefeld für das Passwort des Nutzers.
- **logIn: Button**
  Sichtbarkeit private
  Schaltfläche um den Anmeldevorgang zu starten.
- **register: Button**
  Sichtbarkeit private
  Schaltfläche um zur Registrierung zu gelangen.
- **status: ProgressBar**
  Sichtbarkeit private
  Ladebalken, der während der Überprüfung der Accountdaten angezeigt wird.
- **memoryManager: MemoryManager**
  Sichtbarkeit private
  MemoryManager Instanz um die Accountdaten zu speichern.
- **server: ServerProxy**
  Sichtbarkeit private
  ServerProxy Instanz um Anfragen an den Server zu stellen.
- **serverResponse: «Interface» IServerResponseCallback**
  Sichtbarkeit private
  Implementiert das IServerResponseCallback Interface. Beim Aufruf der onResponse() Methode wird die Antwort des Servers ausgelesen. Bestätigt der Server die eingegebenen Accountdaten ruft das LogInFragment die Launch() Methode der CameraActivity auf. Andernfalls wird dem Nutzer durch einen Toast eine Fehlermeldung angezeigt. Beim Aufruf der onError() Methode wird dem Nutzer die entsprechende Fehlermeldung ebenfalls per Toast angezeigt. Die onProgress() Methode wird ignoriert.

Konstruktoren
Keine, da der Lebenszyklus dieser Klasse von Android gesteuert wird.

Methoden

- **onCreateView (savedInstanceState: Bundle): void**
  Sichtbarkeit public
  Überschreibt die onCreateView() Methode der Superklasse. Lädt alle View Instanzen und instanziiert und übergibt jeder View ihren Listener. Holt sich eine MemoryManager und eine ServerProxy Instanz und instanziiert den IServerResponseCallback.
- **performLogIn (view: View): void**
  Sichtbarkeit public
  Aufgerufen, wenn der Nutzer den Anmeldevorgang startet. Bekommt die View die angeklickt wurde übergeben. Ließt die Texte der TextView Instanzen aus, validiert die Eingaben und ruft die authenticateAccount() Methode auf der ServerProxy Instanz auf.
- **redirectToRegistry (view: View): void**
  Sichtbarkeit public
  Aufgerufen, wenn der Nutzer auf die Registrieren-Schaltfläche tippt. Bekommt die View die angeklickt wurde übergeben. Öffnet die Internetseite zur Registrierung mit dem Standardbrowser des Smartphones.

LogInHelper

Der logInHelper ist eine Fassade, der die Überprüfung, ob der Nutzer sich bereits zuvor angemeldet hat, vereinfacht und die Speicherung dieser Daten zulässt. Durch den LogInHelper ist es nicht notwendig, der Presenter-Ebene der App direkten zugriff auf die Model-Ebene zu geben.

Attribute

- **memoryManager: MemoryManager**
  Sichtbarkeit public
  Die MemoryManager Instanz, mit der der LogInHelper Accountdaten abfragt und speichert.

Konstruktoren

- **LogInHelper(context: Context)**
  Sichtbarkeit public
  Konstruktor, der den MemoryManager instanziiert und ihm den erhaltenen Kontext übergibt.

Methoden

- **saveAccountData (mail: String, pw: String): void**
  Sichtbarkeit public
  Erhält Mail-Adresse und Passwort, die der Nutzer eingegeben hat. Kapselt diese in ein Account Object und übergibt dieses der saveAccountData(..) Methode des MemoryManagers.
- **isUserLoggedIn (): boolean**
  Sichtbarkeit public
  Ruft die getAccountData() Methode des MemoryManagers auf. Ist die Rückgabe null, wird *false* zurückgegeben, andernfalls *true*.

SettingsFragment

extends Fragment
Das SettingsFragment lädt die einzelnen Layout-Komponenten der Einstellungenansicht, reagiert auf Nutzereingaben und veranlasst das Speichern gemachter Änderungen. Jede antippbare Layout-Komponente bekommt einen anonymen OnClickListener, der die mit der jeweiligen Komponente verknüpfte Aktion ausführt.

Attribute

- **settings: Settings**
  Sichtbarkeit private
  Die aktuellen Einstellungen, gekapselt in einer Settings Instanz.
- **qualityLow: TextView**
  Sichtbarkeit private
  Anklickbarer Text um niedrige Auflösung festzulegen. Antippen ändert den in der Settings Instanz gespeicherten Wert *quality*, färbt diese TextView grün und *qualityMedium* und *qualityHigh* grau.
- **qualityMedium: TextView**
  Sichtbarkeit private
  Anklickbarer Text um mittlere Auflösung festzulegen. Antippen ändert den in der Settings Instanz gespeicherten Wert *quality*, färbt diese TextView grün und *qualityLow* und *qualityHigh* grau.
- **qualityHigh: TextView**
  Sichtbarkeit private
  Anklickbarer Text um hohe Auflösung festzulegen. Antippen ändert den in der Settings Instanz gespeicherten Wert *quality*, färbt diese TextView grün und *qualityLow* und *qualityMedium* grau.
- **fps: TextView**
  Sichtbarkeit private
  Text zum Anzeigen der aktuell eingestellten Bilder pro Sekunde.
- **fpsBar: SeekBar**
  Sichtbarkeit private
  Balken zum Tippen und Ziehen um die Einstellung der Bilder pro Sekunde zu ändern. Antippen ändert den in der Settings Instanz gespeicherten Wert *fps* und aktualisiert *fps*.
- **bufferSize: TextView**
  Sichtbarkeit private
  Text zum Anzeigen der aktuell eingestellten Buffergröße.
- **incBuffer: Button**
  Sichtbarkeit private

Schaltfläche um die Buffergröße zu erhöhen. Antippen erhöht die in der Settings Instanz gespeicherten Wert *bufferSize* und aktualisiert die TextView *bufferSize*.

- **decBuffer: Button**
  Sichtbarkeit private
  Schaltfläche um die Buffergröße zu verringern. Antippen verringert die in der Settings Instanzgespeicherten Wert *bufferSize* und aktualisiert die TextView *bufferSize*.
- **logOut: Button**
  Sichtbarkeit private
  Schaltfläche um AccountDaten zu löschen und alle Appansichten außer die LogInActivity unzugänglich zu machen. Antippen ruft die Methode deleteAccountData() des MemoryManagers und die Methode Launch() der LogInActivity auf.
- **memoryManager: MemoryManager**
  Sichtbarkeit private
  MemoryManager Instanz um gespeicherte Einstellungen abzufragen und neue Einstellungen zu speichern.

Konstruktoren
Keine, da der Lebenszyklus dieser Klasse von Android gesteuert wird.

Methoden

- **onCreateView (savedInstanceState: Bundle): void**
  Sichtbarkeit public
  Überschreibt die *onCreateView* Methode der Superklasse. Lädt alle View Instanzen, instanziiert und übergibt jeder View ihren Listener, holt sich eine MemoryManager Instanz und ruft die getSettings() Methode des MemoryManagers auf. Wenn sie das Settings Objekt von diesem Methodenaufruf erhalten hat, weißt sie es der *settings* Variable zu und aktualisiert alle View-Komponenten, sodass sie die in *settings* gespeicherten Daten widerspiegeln.
- **onPause (): void**
  Sichtbarkeit public
  Überschreibt die onCreateView() Methode der Superklasse. Ruft die saveSettings(..) Methode des MemoryManagers auf und übergibt das Settings Objekt.

LegalFragment

extends Fragment
Das LegalFragment lädt die einzelnen Layout-Komponenten der Impressumsansicht und reagiert auf Nutzereingaben.

Attribute

- **legal: TextView**
  Sichtbarkeit private
  Anklickbarer Text mit Titel "Impressum". Zeigt einen Dialog nach Antippen an, in dem eine WebView das Impressum anzeigt.
- **privacy: TextView**
  Sichtbarkeit private
  Anklickbarer Text mit Titel "Datenschutz". Öffnet die Internetseite mit der Datenschutzerklärung nach Antippen mit dem Standardbrowser.
- **licenses: TextView**
  Sichtbarkeit private
  Anklickbarer Text mit Titel "Lizenzen". Zeigt einen Dialog nach Antippen an, in dem eine WebView alle Lizenzen aller Drittanbieter-Bibliotheken anzeigt.
- **website: TextView**
  Sichtbarkeit private
  Anklickbarer Text mit Titel "Website". Öffnet die Internetseite der App nach Antippen mit dem Standardbrowser.
- **dialogBuilder: DialogBuilder**
  Sichtbarkeit private
  Wird verwendet um die Dialoge für Impressum und Lizenzen zu erstellen
- **content: WebView**
  Sichtbarkeit private
  WebView die durch die Dialoge angezeigt wird.
- **loading: ProgressBar**
  Sichtbarkeit private
  Ladebalken, der angezeigt wird, während die Dialoge die WebView Laden.

Konstruktoren
Keine, da der Lebenszyklus dieser Klasse von Android gesteuert wird.

Methoden

- **onCreateView (savedInstanceState: Bundle): void**
  Sichtbarkeit public
  Überschreibt die onCreateView() Methode der Superklasse. Lädt alle View Instanzen und instanziiert und übergibt jeder View ihren Listener.

VideosFragment

extends Fragment
Das VideosFragment lädt die einzelnen Layout-Komponenten der Videosansicht, sowie die Liste aller aufgezeichneter und verschlüsselter Videos, reagiert auf Nutzereingaben und zeigt Videodetails an, oder veranlasst das Löschen von Videos nach entsprechender Nutzeriengabe. Jede antippbare Layout-Komponente bekommt einen anonymen OnClickListener der die mit der jeweiligen Komponente verknüpfte Aktion ausführt.
Siehe auch: Video in der App löschen

Attribute

- **videosList: ArrayList<Video>**
  Sichtbarkeit private
  ArrayList, die mit Videodaten aller Videos, die von der App gespeichert wurden und verschlüsselt vorliegen, gefüllt wird. Als Listenelement wird dabei ein Video-Objekt verwendet.
- **videos: ListView**
  Sichtbarkeit private
  ListView Instanz, die die Videos die in *videosList* vorliegen anzeigt. Dazu wird ein eigener Adapter verwendet, der von *BaseAdapter* erbt und ein eigenes Layout anzeigt. Die Schaltflächen jedes Listeneintrags zum Anzeigen der Metadaten eines Videos und zum Löschen eines Videos werden zudem mit einem Listener ausgestattet, der die Metadaten einblendet bzw. den MemoryManager verwendet, um das ausgewählte Video zu löschen. Soll das Video gelöscht werden wird zudem das Video von der ListView und der *videosList* entfernt.
- **memoryManager: MemoryManager**
  Sichtbarkeit private
  MemoryManager Instanz um Videodaten vom Speicher zu laden.

Konstruktoren
Keine, da der Lebenszyklus dieser Klasse von Android gesteuert wird.

Methoden

- **onCreateView (savedInstanceState: Bundle): void**
  Sichtbarkeit public
  Überschreibt die onCreateView() Methode der Superklasse. Lädt alle View Instanzen, instanziiert und übergibt jeder View ihren Listener, holt sich eine MemoryManager Instanz und ruft die getAllVideos() Methode des MemoryManagers auf. Weißt die Liste der erhaltenen Video-Objekte der *videosList* ArrayList zu und fügt sie in die *videos* ListView ein.

CameraView

extends CameraView
implements SurfaceHolder.Callback
Die CameraView ist eine SurfaceView, die die Vorschaubilder der Kamera anzeigt. Da sie das SurfaceHolder.Callback implementiert wird sie über Erstellung, Änderung und Zerstörung ihres Inhaltes durch Android benachrichtigt.

Attribute

- **cameraHandler: «Interface» CameraHandler**
  Sichtbarkeit private
  CameraHandler Instanz, deren Methoden resumeHandler() und pauseHandler() aufgerufen werden, wenn die surfaceCreated() oder die surfaceDestroyed() Methoden der CameraView aufgerufen wurden.

Konstruktoren

- **CameraView(context: Context, attrs: AttributeSet)**
  Sichtbarkeit public
  Konstruktor, der von Android beim Instanziieren der CameraView verwendet wird. Durch Verwendung dieses Konstruktors kann die CameraView direkt in ein XML-Layout eingebunden werden und muss nicht von der CameraActivity instanziiert werden.

Methoden

- **setCameraHandler (cameraHandler: CameraHandler): void**
  Sichtbarkeit public
  Setter, der dem *cameraHandler* Feld eine CameraHandler Instanz zuweist. Falls die CameraView bereits angezeigt wird, wird die Methode resumeHandler() auf dem CameraHandler aufgerufen.
- **surfaceCreated(holder: SurfaceHolder): void**
  Sichtbarkeit public
  Implementiert die Methode *surfaceCreated* des SurfaceHolder.Callback Interfaces. Falls *cameraHandler* nicht *null* ist wird die Methode resumeHandler() des CameraHandlers aufgerufen.
- **surfaceChanged(holder: SurfaceHolder, format: int, width: int, height: int): void**
  Sichtbarkeit public
  Implementiert die Methode surfaceChanged() des SurfaceHolder.Callback Interfaces. Falls *cameraHandler* nicht *null* ist wird die Methode pauseHandler() auf dem CameraHandler aufgerufen und anschließend die Methode resumeHandler() aufgerufen.

- **surfaceDestroyed(holder: SurfaceHolder): void**
  Sichtbarkeit public
  Implementiert die Methode surfaceDestroyed() des SurfaceHolder.Callback Interfaces. Falls *cameraHandler* nicht *null* ist wird die Methode pauseHandler() auf dem CameraHandler aufgerufen.

2.4.3 ApplicationLogic.Camera

«Interface» CameraHandler

Der CameraHandler sorgt dafür, dass alle Komponenten, die die Kamerafunktionen der App nutzen möchten, eine einheitliche Schnittstelle zur Verfügung gestellt bekommen. Dadurch kann die Beobachtung, sowie die Aufnahme gestartet und gestoppt werden, ohne dass die aufrufende Komponente Details über den Speicher oder die Kamera wissen muss. Demzufolge kann z.B. die in CompatCameraHandler verwendete Camera API leicht gegen die Camera2 API ausgetauscht oder die Art der Speicherung verändert werden. Da die Kamerafunktionen so leicht zugänglich und kontrollierbar werden, erleichtert der CameraHandler ebenfalls den Zugriff auf die Kamera durch z.B. einen Service oder ähnlichem. Der CameraHandler setzt folglich das Strategy Muster um, wobei Klienten nur von ihm anstatt von seiner Implementierung abhängen.

Methoden

- **schedulePersisting(): void**
  Sichtbarkeit public
  Kann aufgerufen werden um die Persistierung auszulösen. Die CameraHandler-Implementierung entscheidet selbst, wann und wie genau die Persistierung abläuft.
- **setMetadata (metadata: Metadata): void**
  Sichtbarkeit public
  Schnittstelle, um der CameraHandler-Implementierung von außen Metadaten übergeben zu können.
- **resumeHandler(): void**
  Sichtbarkeit public
  Kann aufgerufen werden, um die Beobachtung zu starten bzw. sie fortzusetzen, falls sie zuvor unterbrochen wurde.
- **pauseHandler(): void**
  Sichtbarkeit public
  Kann aufgerufen werden, um die Beobachtung zu unterbrechen.

CompatCameraHandler

implements MediaRecorder.OnInfoListener, «Interface» CameraHandler
Ziel einer CameraHandler-Implementierung ist es, eine Komponente darzustellen, die grundlegende Funktionen für die Bedienung der Kamera zur Verfügung zu stellen. Neben der Verwaltung der Kamera gehören Zwischenspeicherung und Persistierung ebenfalls zu diesen Grundfunktionen. Der CompatCameraHandler verwendet dabei die mit älteren Androidversionen kompatible Camera API zum Bedienen der Kamera, Anzeigen der Vorschau und Aufnahme von Videos und kümmert sich darum, den Ringbuffer zu befüllen und die Persistierung seines Inhaltes zu koordinieren. Beschrieben wird der Ringbuffer mit kurzen Videostücken. Eben diese Videostücke muss der CompatCameraHandler aufzeichnen. Das Auslösen der Aufnahme gehört nicht zu seinen Aufgaben, dies wird durch seine Unterklassen oder durch die ihn instanziierenden Komponenten realisiert. Um eine Kommunikation zwischen aufrufender Komponente, z.B. einem Service oder einer Activity, und dem CameraDataHandler zu ermöglichen, muss diese den «Interface» IRecordCallback implementieren. Die aufrufende Komponente agiert dann als Beobachter des CameraDataHandler.
Siehe auch: Videoaufnahme in der App - Beobachtung

Attribute

- **camera: Camera**
  Sichtbarkeit private
  Die Camera Instanz, deren Vorschaubilder angezeigt werden und die zur Aufnahme verwendet wird.
- **cameraParameters: Camera.Parameters**
  Sichtbarkeit private
  Die CameraParameter Instanz, die verwendet wird, um Höhe und Breite der Vorschau einzustellen.
- **camcorderProfile: CamcorderProfile**
  Sichtbarkeit private
  Die CamcorderProfile Instanz, die verwendet wird, um Höhe, Breite und Qualität des Videos und der Vorschau einzustellen.
- **preview:SurfaceView**
  Sichtbarkeit private
  Die SurfaceView Instanz, die zur Ausgabe der Vorschaubilder verwendet werden soll.
- **context: Context**
  Sichtbarkeit private
  Aktuelle Context Instanz, die verwendet wird, um eine Instanz des MemoryManagers zu erhalten.
- **mediaRecorder: MediaRecorder**
  Sichtbarkeit private

MediaRecorder Instanz, die zur Aufnahme der Videstückchen verwendet wird.

- **settings: Settings**
  Sichtbarkeit private
  Von MemoryManager abgefragte Settings Instanz mit den Einstellungen des Nutzers.
- **metadata: Metadata**
  Sichtbarkeit private
  Die Metadaten, die gespeichert werden sollen und dem AsyncPersistor übergeben werden.
- **callback: «Interface» IRecordCallback**
  Sichtbarkeit private
  IRecordCallback Instanz die als Observer agiert. Auf ihr werden Methoden aufgerufen, sobald die Aufnahme gestartet oder gestoppt wird.
- **currentOutputFile: File**
  Sichtbarkeit private
  Aktuell verwendete Datei in die das momentan aufgezeichnete Videostückchen geschrieben wird. Diese Datei wird in den RingBuffer eingefügt, wenn die Aufnahme des Videstückchens abgeschlossen ist.
- **«final» videoChunkLength: int**
  Sichtbarkeit private
  Länge in Sekunden eines Videostückchens.
- **memoryManager: MemoryManager**
  Sichtbarkeit private
  MemoryManager Instanz um Einstellungen und Speicherort für die Videostückchen abzufragen. Wird dem AsyncPersistor bei dessen Instanziierung übergeben.
- **persistCallback: «Interface» IPersistCallback**
  Sichtbarkeit private
  IPersistCallback Impementierung, die als Observer des AsyncPersistors agiert und die benachrichtigt wird, sobald die Persistierung gestartet oder gestoppt wird. Wird die Persistierung gestartet, also die Methode onPersistingStarted() aufgerufen, ruft die IPersistCallback Implementierung die onRecordingStopped() Methode des IRecordCallbacks auf und instanziiert Metadata und Ringbuffer neu, so dass auf den neuen Instanzen weitergearbeitet wird und die alten Instanzen ohne Kollisionen persistiert werden können. Der Aufruf der onPersistingStopped() Methode wird ignoriert.
- **asyncPersistor: AsyncPersistor**
  Sichtbarkeit private
  Asynchroner Task der die Persistierung des Ringbuffer-Inhaltes übernimmt.

- **ringBuffer: Ringbuffer<E>**
  Sichtbarkeit private
  FIFO-artiger Buffer der die einzelnen Videostücke zwischenspeichert und dem AsyncPersistor übergeben wird.

Konstruktoren

- **CameraHandler(context: Context, recordCallback: IRecordCallback)**
  Sichtbarkeit public
  Konstruktor, der das *context*, *preview* und das *callback* Feld zuweist, sich eine MemoryManager Instanz holt, die vom MemoryManager abgefragte Settings Instanz *settings* zuweist, den Ringbuffer instanziiert und *cameraProfile* initialisiert.

Methoden

- **onInfo(mr: MediaRecorder, what: int, extra: int): void**
  Sichtbarkeit public
  Implementiert die onInfo() Methode des MediaRecorder.OnInfoListener Interface. Aufgerufen, sobald ein Videostückchen die Länge *videoChunkLength* erreicht. Diese Intention wird durch Vergleichen des Parameters *what* überprüft. Fügt *currentOutputFile* in den RingPuffer ein, ruft die Methode stopRecordingChunk() und anschließend die Methoden prepareMediaRecorder() und startRecordingChunk() auf.
- **setMetadata (metadata: Metadata): void**
  Sichtbarkeit public
  Implementiert die setMetadata(..) Methode des CamerHolder Interface.
- **schedulePersisting(): void**
  Sichtbarkeit public
  Implementiert die schedulePersisting() Methode des CamerHolder Interface. Ruft die onRecordingStarted() Methode des IRecordCallbacks auf. Instanziiert dann einen neuen AsyncPersistor, übergibt diesem eine Referenz auf den RingBuffer und den MemoryManager und ruft dessen start() Methode auf, der sie die Metadaten übergibt.
- **resumeHolder(): boolean**
  Sichtbarkeit public
  Implementiert die resumeHolder() Methode des CamerHolder Interface. Ruft die Methoden prepareCamera(), prepareMediaRecorder() und startRecordingChunk() auf. Gibt *true* zurück falls keiner dieser Aufrufe fehlschlägt.
- **pauseHolder(): void**
  Sichtbarkeit public
  Implementiert die pauseHolder() Methode des CamerHolder Interface. Fügt *currentOutputFile* in den RingBuffer ein und ruft die Methoden stopRecordingChunk(), releaseMedaRecorder() und releaseCamera() auf.

- **prepareCamera(): void**
  Sichtbarkeit private
  Öffnet die Kamera und weist sie dem *camera* Feld zu. Danach werden ihr die Camera.Parameters geholt und mit der Vorschaugröße modifiziert. Schließlich werden die modifizierten Camera.Parameters sowie den SurfaceHolder der SurfaceView Instanz zugewiesen. Behandelt Exceptions. Gibt *true* zurück, falls keine Exceptions auftreten.
- **prepareMediaRecorder(): void**
  Sichtbarkeit private
  Instanziiert einen MediaRecorder, ruft die Methode unlock() der Kamera auf und weist die Kamera dem MediaRecorder zu. Danach wird dem MediaRecorder eine VideoSource, das CamcorderProfile, das gewünschte MPEG-4 Ausgabeformat, die gewünschte Framerate, die Zieldatei in die das Video geschrieben werden soll, die Videolänge die in videoChunkLength gespeichert ist und die Surface des SurfaceHolders übergeben. Zuletzt wird der CompatCameraHandler als OnInfoListener gesetzt. Behandelt Exceptions. Gibt *true* zurück falls keine Exceptions auftreten.
- **startRecordingChunk(): void**
  Sichtbarkeit private
  Ruft die start() Methode des MediaRecorders auf und behandelt Exceptions. Gibt *true* zurück falls keine Exceptions auftreten.
- **stopRecordingChunk(): void**
  Sichtbarkeit private
  Ruft die stop() Methode des MediaRecorders auf und behandelt Exceptions.
- **releaseMediaRecorder(): void**
  Sichtbarkeit private
  Ruft die release() Methode des MediaRecorders auf und behandelt Exceptions.
- **releaseCamera(): void**
  Sichtbarkeit private
  Ruft die release() Methode der Camera auf und behandelt Exceptions.

TriggeringCompatCameraHandler

extends CompatCameraHandler
implements SensorEventListener, View.OnClickListener
Der TriggeringCompatCameraHandler ist ein CompatCameraHandler, der seine Videoaufnahme nach Nutzereingabe, oder nachdem die gemessenen Beschleunigungswerte des G-Sensors einen Maximalwert überschreiten, auslößt.

Attribute

- **maxGForce: float**
  Sichtbarkeit private
  Maximalwert für die gemessene Beschleunigung. Überschreiten die Messwerte des Sensors diesen Wert wird die Videoaufnahme gestartet.
- **accelerometerSensor: Sensor**
  Sichtbarkeit private
  Beschleunigungssensor, der überwacht werden soll. CompatCameraDataHandler ist ein Observer dieses Sensors.
- **sensormanager: SensorManager**
  Sichtbarkeit private
  SensorManager, der verwendet wird, um den *accelerometerSensor* zuzuweisen und den TriggeringCompatCameraHandler als SensorEventListener zu registrieren.

Konstruktoren

- **CompatCameraHandler(context: Context, preview: CameraView, recordCallback: «Interface» IRecordCallback)**
  Sichtbarkeit public
  Konstruktor, der *super* aufruft und der Elternklasse die erhaltenen Parameter übergibt.

Methoden

- **onSensorChanged (event: SensorEvent): void**
  Sichtbarkeit public
  Implementiert die onSensorChanged() Methode des SensorEventListener Interfaces. Überprüft das *values* Feld des erhaltenen SensorEvents und vergleicht die Einträge dieses Arrays mit *maxGForce*. Falls die gemessenen Werte *maxGForce* überschreiten wird eine Metadata Instanz erstellt und die Methoden setMetadata(..) und schedulePersisting() aufgerufen, die in der Elternklasse implementiert sind.
- **onClick (view: View): void**
  Sichtbarkeit public
  Implementiert die onClick() Methode des View.OnClickListener Interfaces.

Beim Aufruf wird eine Metadata Instanz erstellt und die Methoden setMetadata(..) und schedulePersisting() aufgerufen, die in der Elternklasse implementiert sind.

«Interface» IRecordCallback

Der IRecordCallback dient zum Beobachten der «Interface» CameraHandler Implementierung und wird über Beginn und Ende einer Aufnahme benachrichtigt. Beispielsweise können Benutzeroberflächen ihr Aussehen nach Aufruf einer der Methoden ändern.

Methoden

- **onRecordingStarted(): void**
  Sichtbarkeit public
  Aufgerufen sobald die Aufnahme beginnt. Die Speicherung muss zu diesem Zeitpunkt noch nicht begonnen haben.
- **onRecordingStopped(): void**
  Sichtbarkeit public
  Aufgerufen sobald die Aufnahme endet. Die Speicherung muss zu diesem Zeitpunkt noch nicht abgeschlossen sein.

2.4.4 Utils

Encryptor

Der Encryptor verschlüsselt Daten mithilfe eines hybdriden Verschlüsselungsverfahren. Zunächst wird eine Datei symmetrisch verschlüsselt, danach wird der Schlüssel, der dafür verwendet wurde, selbst asymmetrisch verschlüsselt.

Attribute

- **symmetricKey: SecretKey**
  Sichtbarkeit private

  Symmetrischer Schlüssel zum verschlüsseln von Dateien.

Konstruktoren

- **Encryptor()**
  Sichtbarkeit public

  Generiert einen symmetrischen Schlüssel zur späteren Verschlüsselung von Dateien.

Methoden

- **encrypt (inFile: File, outFile: File): boolean**
  Sichtbarkeit public

  Verschlüsselt eine Datei mithilfe des symmetrischen Schlüssels. Gibt zurück, ob das Verschlüsseln erfolgreich war.
- **saveKey (encKey: File): boolean**
  Sichtbarkeit public

  Verschlüsselt den symmetrischen Key mithilfe eines asymmetrischen öffentlichen Schlüssels und legt diesen in einer Datei ab.

«Interface» IFileEncryptor

IFileEncryptor bietet eine Schnittstelle für Klassen, die mithilfe eines symmetrischen SecretKeys Dateien verschlüsseln.

Methoden

- **encrypt (input :File, key: SecretKey, output: File): boolean**
  Sichtbarkeit public

  Nimmt das input-File und entschlüsselt es mit dem key. Speichert die entschlüsselte Datei im output-File. Gibt zurück, ob die Entschlüsselung erfolgreich war.

- **generateKey (): SecretKey**
  Sichtbarkeit public

  Generiert einen neuen symmetrischen Schlüssel.

AESEncryptor

implements «Interface» IFileEncryptor
Der AESDecryptor bietet eine konkrete Implementierung eines Encryptors für symmetrische Verschlüsselung. Er verwendet hierfür das AES-Verfahren.

Konstruktoren

- **AESEncryptor()**
  Sichtbarkeit public

  Standardkonstruktor

Methoden

- **encrypt (input :File, key: SecretKey, output: File): boolean**
  Sichtbarkeit public

  Implementiert encrypt(..) von IFileEncryptor.
- **generateKey (): SecretKey**
  Sichtbarkeit public

  Implementiert generateKey() von IFileEncryptor.

«Interface» IKeyEncryptor

IKeyEncryptor bietet eine Schnittstelle für Klassen die SecretKeys mithilfe eines öffentlichen asymmetrischen Schlüssels verschlüsseln.

Methoden

- **encrypt (key :SecretKey, encKey: File): boolean**
  Sichtbarkeit public

  Nimmt den öffentlichen Schlüssel des Webservers und verschlüsselt damit den den key. Legt den Verschlüsselten key als Datei ab. Gibt zurück, ob das Verschlüsseln erfolgreich war.

RSAEncryptor

implements «Interface» IKeyEncryptor
Der RSAEncryptor bietet eine konkrete Implementierung eines asymmetrischen Encyptors für SecretKeys. Er nutzt dazu das RSA Verfahren.

Konstruktoren

- **RSAEncryptor()**
  Sichtbarkeit public

  Standardkonstruktor

Methoden

- **encrypt (key :SecretKey, encKey: File): boolean**
  Sichtbarkeit public

  Implementiert die Methode encrypt(..) von IKeyEncryptor.

Ringbuffer<E>

Der Ringbuffer ist eine nach Ankunftszeit geordnete Datenstruktur, die wenn sie Daten annimmt immer das älteste Element löscht, falls ihre Kapazität überschritten wird.

Attribute

- **length: int**
  Sichtbarkeit private

  Kapazität des Puffers
- **data: Queue<E>**
  Sichtbarkeit private

  Interne Datenstruktur des Puffers

Konstruktoren

- **Ringbuffer(length: int)**
  Sichtbarkeit public

  Erstellt einen neuen leeren Ringpuffer mit der angegebenen Kapazität.

Methoden

- **offer (element: E): boolean**
  Sichtbarkeit public

  Fügt das neue Element am Ende der Datenstruktur hinzu. Falls dadurch die Kapazität überschritten wird, wird am Anfang der Datenstruktur ein Element entfernt. Gibt zurück, ob das Einfügen erfolgreich war.
- **getData (): Queue<E>**
  Sichtbarkeit public

  Gibt die Daten des Ringpuffers zurück.

AsyncPersistor

extends AsyncTask <Metadata, Void, Boolean>
Der AsyncPersistor ist dafür verantwortlich, den Inhalt des Ringbuffers und die Metadaten in Dateien zu schreiben und deren Verschlüsselung zu koordinieren. Dies passiert asynchron zum UI-Thread.

Attribute

- **memoryManager: MemoryManager**
  Sichtbarkeit private
  MemoryManager, von dem der Speicherort der Dateien abgefragt wird.
- **ringbuffer: MemoryManager**
  Sichtbarkeit private
  RingBuffer, der die Videostücke hält, die persistiert werden sollen.
- **encryptor: Encryptor**
  Sichtbarkeit private
  Encryptor, der die Dateien verschlüsselt.
- **callback: «Interface» IPersistCallback**
  Sichtbarkeit private
  IPersistCallback Instanz um mit der Klasse zu Kommunizieren, die den AsyncPersistor instanziiert.

Konstruktoren

- **AsyncPersistor(memoryManager: MemoryManager, ringbuffer: Ringbuffer)**
  Sichtbarkeit public
  Konstruktor, der *memoryManager* und *ringbuffer* zuweißt.

Methoden

- **onProgressUpdate (): void**
  Sichtbarkeit public
  Überschreibt die Methode onProgressUpdate() des AsyncTask. Wird von Android während der Ausführung des AsyncTasks aufgerufen. Ruft die onPersistingStarted() Methode des IPersistCallbacks auf.
- **doInBackground (metadata: Metadata...): void**
  Sichtbarkeit public
  Überschreibt die Methode doInBackground() des AsyncTask. Wird von Android aufgerufen und im Hintergrund ausgeführt. Bekommt ein Array von Metadaten übergeben, in dem der erste Eintrag der Metadaten Instanz entspricht, die der start() Methode übergeben wurde. Fügt die Videostücke des Ringbuffers zu einem Video zusammen, schreibt die Metadaten in eine Datei und verschlüsselt die Video- und Metadaten-Datei. Zum erhalten der

Dateien, in die die Daten geschrieben werden, wird der MemoryManager verwendet. Zum Verschlüsseln der Dateien wird der Encryptor verwendet. Gibt *true* zurück, falls kein Fehler auftritt.

- **onPostExecute (result: boolean): void**
  Sichtbarkeit public
  Überschreibt die Methode onPostExecute() des AsyncTask. Wird von Android aufgerufen nachdem der Hintergrundthread zu Ende gelaufen ist. Ruft die Methode onPersistingStopped() des IPersistCallbacks auf.

«Interface» IPersistCallback

Das IPersistCallback dient zum Beobachten des Speichervorgangs des Videos. Es wird über Beginn und Ende des Speichervorgangs benachrichtigt.

Methoden

- **onPersistingStarted (): void**
  Sichtbarkeit public

  Wird aufgerufen, sobald die Persistierung beginnt.
- **onPersistingStopped (): void**
  Sichtbarkeit public

  Wird aufgerufen, sobald die Persistierung endet.

2.4.5 Data

Metadata

Datencontainer für Videometadaten.

Attribute

- **date: long**
  Sichtbarkeit private

  Aufnahmedatum des Videos
- **triggerType: String**
  Sichtbarkeit private

  Auslöser der Videoaufnahme
- **force: float[ ]**
  Sichtbarkeit private

  G-Sensorwerte zum Auslösezeitpunkt

Konstruktoren

- **Metadata(date: long, triggerType: String, force: float[ ])**
  Sichtbarkeit public

  Weist die Parameter den Attributen zu.
- **Metadata(json: String)**
  Sichtbarkeit public

  Liest die Parameter aus dem json String und weist sie den Attributen zu.

Methoden

- **getAsJSON(): String**
  Sichtbarkeit public

  Verpackt die Attribute als json String und gibt diesen zurück.

Video

Datencontainer für Videoinformationen.

Attribute

- **name: String**
  Sichtbarkeit private

  Name des Videos.
- **encVideoFile: File**
  Sichtbarkeit private

  Ort der verschlüsselten Videodatei
- **encMetaFile: File**
  Sichtbarkeit private

  Ort der verschlüsselten Metadaten
- **readableMetadata: Metadata**
  Sichtbarkeit private

  Metadaten des Videos

Konstruktoren

- **Video(name: String, encVideoFile: File, encMetaFile: File, readableMetadata: Metadata)**
  Sichtbarkeit public

  Weist die Parameter den Attributen zu.

Account

Datencontainer für Accountinformationen eines Benutzers.

Attribute

- **mail: String**
  Sichtbarkeit private

  E-Mail des Benutzers.
- **password: String**
  Sichtbarkeit private

  Passwort des Benutzers.

Konstruktoren

- **Account(mail: String, password: String)**
  Sichtbarkeit public

  Weist die Parameter den Attributen zu.
- **Account(json: String)**
  Sichtbarkeit public

  Liest die Parameter aus dem json String und weist sie den Attributen zu.

Methoden

- **getAsJSON (): String**
  Sichtbarkeit public

  Verpackt die Attribute als json String und gibt diesen zurück.

Settings

Datencontainer für Appeinstellungen.

Attribute

- **fps: int**
  Sichtbarkeit private

  Bildwiederholrate
- **bufferSizeSec: Typ**
  Sichtbarkeit private

  Länge des aufgenommenen Videos bei Persistierung.
- **quality: int**
  Sichtbarkeit private

  Bildqualität der Aufnahme

Konstruktoren

- **Settings(fps: int, bufferSizeSec: Typ, quality: int)**
  Sichtbarkeit public

  Weist die Parameter den Attributen zu.
- **Settings(json: String)**
  Sichtbarkeit public

  Liest die Parameter aus dem json String und weist sie den Attributen zu.

Methoden

- **setResolution (resulution: Vector2D): Rückgabetyp**
  Sichtbarkeit public

  Ändert die Bildqualität
- **getAsJSON (): String**
  Sichtbarkeit public

  Verpackt die Attribute als json String und gibt diesen zurück.

MemoryManager

Die MemoryManager-Klasse verwaltet die Daten auf der App. Er bietet die Funktionalität bestimmte Daten anzufordern, zu sichern oder zu löschen. Die Klasse ist ein Singelton, somit existiert nur eine Instanz mit der man die folgenden Methoden aufrufen kann.

Attribute

- **instance: MemoryManager**
  Sichtbarkeit private
  statisch

  Repräsentiert die Singleton Instanz.
- **context: Context**
  Sichtbarkeit private

  Kontext der App. Beinhaltet die benötigten Speicherdirektorien um Daten in den richtigen Speicherplätzen abzulegen bzw. wiederzufinden.

Konstruktoren

- **MemoryManager(context: Context)**
  Sichtbarkeit private

  Privater Konstruktor der Klasse, der nur aufgerufen wird wenn das instance-Attribut der Klasse noch nicht gesetzt wurde.

Methoden

- **GetInstance(context: Context): MemoryManager**
  Sichtbarkeit public
  statisch

  Statische Methode um sicherzustellen, dass es nur eine Instanz der Klasse gibt. Wenn Instanz noch nicht erzeugt wurde, ruft die Methode den Konstruktor auf. Das instance-Attribut wird dann auf die erzeugte Instanz gesetzt und zurückgegeben. Wenn instance bereits erzeugt wurde, wird dieses direkt zurückgegeben.
- **getTempVideoFile(): File**
  Sichtbarkeit public

  Gibt die temporäre Videodatei in Form eines Files zurück.
- **getTempSymmetricKeyFile(): File**
  Sichtbarkeit public

  Gibt die temporäre symmetrische Schlüsseldatei in Form eines Files zurück.
- **deleteTempData()**
  Sichtbarkeit public

  Löscht alle temporären Daten.

- **saveAccountData(account: Account)**
  Sichtbarkeit public

  Sichert die Accountdaten die per Parameterübergabe mitgegeben werden.
- **getAccountData(): Account**
  Sichtbarkeit public

  Gibt eine Instanz der Klasse Account, in der sich die Accountdaten befinden.
- **deleteAccountData()**
  Sichtbarkeit public

  Löscht die Accountdaten des Benutzers.
- **saveSettings(settings: Settings)**
  Sichtbarkeit public

  Sichert die Einstellungen die über Parameterübergabe mitgegeben werden.
- **getSettings(): Settings**
  Sichtbarkeit public

  Gibt eine Instanz der Klasse Settings zurück, welche die Einstellungen repräsentieren.
- **saveEncryptedSymmetricKey(videoName: String, key: String): File**
  Sichtbarkeit public

  Sichert den verschlüsselten symmetrischen Schlüssel eines Videos und gibt als Rückgabe den Schlüssel als File zurück.
- **saveEncryptedVideo(videoName: String, byte[]): File**
  Sichtbarkeit public

  Sichert das verschlüsselte Video und gibt als Rückgabe das Video als File zurück.
- **saveEncryptedMetadata(videoName: String, metadata: Metadata): File**
  Sichtbarkeit public

  Sichert die verschlüsselten Metadaten eines Videos und gibt als Rückgabe die Daten als File zurück.
- **saveReadableMetadata(videoName: String, metadata: Metadata): File**
  Sichtbarkeit public

  Sichert die lesbaren Metadaten eines Videos gibt als Rückgabe die Daten als File zurück.
- **deleteEncryptedSymmetricKey(videoName: String)**
  Sichtbarkeit public

Löscht den verschlüsselten symmetrischen Schlüssel.

- **deleteEncryptedVideo(videoName: String)**
  Sichtbarkeit public

  Löscht das verschlüsselte Video.

- **deleteEncryptedMetadata(videoName: String)**
  Sichtbarkeit public

  Löscht die verschlüsselten Metadaten.

- **deleteReadableMetadata(videoName: String)**
  Sichtbarkeit public

  Löscht die lesbaren Metadaten.

- **getAllVideos(): ArrayList<Video>**
  Sichtbarkeit public

  Gibt alle Videos in Form einer ArrayList zurück.

- **getEncryptedSymmetricKey(videoName: String): File**
  Sichtbarkeit public

  Gibt den verschlüsselten symmetrischen Schlüssel zu einem Video als File zurück.

- **getEncryptedVideo(name: String): File**
  Sichtbarkeit public

  Gibt die verschlüsselte Videodatei als File zurück.

- **getEncryptedMetadata(videoName: String): File**
  Sichtbarkeit public

  Gibt die verschlüsselten Metadaten zu einem Video als File zurück.

- **getReadableMetadata(videoName: String): Metadata**
  Sichtbarkeit public

  Gibt die lesbaren Metadaten zu einem Video als File zurück.

ServerProxy

Der ServerProxy ist ein Singelton, das bedeutet das nur eine Instanz der Klasse existieren darf. Über diese Instanz können andere Klassen Methoden des ServerProxys aufrufen. Der ServerProxy ist die Verbindungsstelle zwischen App und Web-Dienst. Er kümmert sich um die Authentifizierungs- und Hochladeanfragen der App.

Attribute

- **instance: ServerProxy**
  Sichtbarkeit public
  statisch

  Das Attribut wird an andere Klasse weiterzugeben um öffentliche Methoden dieser Klasse aufzurufen. Dies garantiert die Einzigartigkeit der Instanz der Klasse.

Konstruktoren

- **ServerProxy()**
  Sichtbarkeit private

  Privater Konstruktor der bei der ersten Instanzierung der Klasse eine Instanz bildet. Die Instanz initialisiert dann das instance-Attribut.

Methoden

- **GetInstance (): ServerProxy**
  Sichtbarkeit public
  statisch

  Ruft den privaten Konstruktor auf wenn Klasse zum ersten Mal initialisiert wurde. Gibt dann das instance-Attribut der Klasse zurück.
- **videoUpload (video: File, metadata: File, symKey: File, account: Account, callback: IServerResponseCallback)**
  Sichtbarkeit public

  Erstellt eine Anfrage zum Hochladen der App. Dabei wird eine VideoUploadTask Instanz erstellt und die Parameter weiter zur Task übergeben.
- **authenticateAccount (account: Account, callback: IServerResponseCallback)**
  Sichtbarkeit public

  Erstellt eine Anfrage zum Authentifizieren der App. Hierbei wird eine AuthenticateTask Instanz gebildet und der Account und der Callback übergeben.
- **cancelRequest()**
  Sichtbarkeit public

  Beendet eine Anfrage an den ServerProxy.

«Interface» IServerResponseCallback

Das IServerResponseCallback dient zum Beobachten der Serveranfragen. Es wird benachrichtigt, sobald ein Fehler auftritt, ein Fortschritt erzielt wurde oder die Anfrage abgeschlossen ist.

Methoden

- **onResponse(result: String)**
  Sichtbarkeit public

  Aufgerufen, wenn die Netzwerkabfrage eine Antwort geliefert hat. Der Parameter *result* kann auch einen Fehlercode enthalten.

- **onProgress(percent: int)**
  Sichtbarkeit public

  Aufgerufen, wenn der Hintergrundprozess Fortschritt gemacht hat. Bekommt die Prozentzahl des Fortschritts übergeben.

- **onError(error: String)**
  Sichtbarkeit public

  Aufgerufen, wenn beim Ausführen der Anfrage ein Fehler aufgetreten ist. Dieser Fehler bezieht sich nur auf Fehler, die auf dem Gerät selbst aufgetreten sind. Fehler, die der Web-Dienst meldet, werden hier nicht beachtet.

AuthenticateTask

implements AsyncTask<String, Integer, String>
Der AuthenticateTask authentifiziert einen Nutzer über den Web-Dienst. Er ist als asynchroner Task implementiert damit die App nicht auf die Antwort des Dienstes warten muss.
Siehe auch: Anmelden in der App

Attribute

- **account: Account**
  Sichtbarkeit private

  Benutzeraccount des anmeldenden Nutzers
- **callback: IServerResponseCallback**
  Sichtbarkeit private

  Callback, dessen Methoden aufgerufen werden, um Ergebnisse der Netzwerkabfrage zur UI durchzureichen.

Konstruktoren

- **AuthenticateTask(account:Account, callback: IServerResponseCallback)**
  Sichtbarkeit public

  Weist die Parameter den Attributen zu.

Methoden

- **doInBackground(urls: String...): String**
  Sichtbarkeit public

  Überschreibt doInBackground(..) von AsyncTask. Stellt die Netzwerkverbindung zum Server her, ruft dessen authenticateAccount(..) Methode auf und sendet die im Konstruktor spezifizierten Daten.
- **onProgressUpdate(progress: Integer...)**
  Sichtbarkeit public

  Überschreibt doProgressUpdate(..) von AsyncTask. Wird ignoriert.
- **onPostExecute(result: String)**
  Sichtbarkeit public

  Überschreibt onPostExecute(..) von AsyncTask. Ruft die onResponse(..) Methode des IServerCallbacks auf, falls die Netzwerkabfrage erfolgreich war. Ruft die onError(..) Methode des IServerCallbacks auf, falls ein Fehler auftrat.

VideoUploadTask

implements AsyncTask<String, Integer, String>
Der VideoUploadTask lädt ein Video auf den Web-Dienst hoch. Er ist als asynchroner Task implementiert damit das aufwendige Hochladen des Videos die App nicht aufhält.

Attribute

- **video: File**
  Sichtbarkeit private

  Ort des hochzuladenden Videos.
- **metadata: File**
  Sichtbarkeit private

  Ort der hochzuladenden Metadaten
- **symKey: File**
  Sichtbarkeit private

  Ort des verschlüsselten symmetrischen Schlüssels.
- **account: Account**
  Sichtbarkeit private

  Benutzeraccount des aktiven Nutzers
- **callback: IServerResponseCallback**
  Sichtbarkeit private

  Callback, dessen Methoden aufgerufen werden, um Ergebnisse der Netzwerkabfrage zur UI durchzureichen.

Konstruktoren

- **VideoUploadTask(video: File, metadata: File, symKey: File, account: Account, callback: IServerResponseCallback)**
  Sichtbarkeit public

  Weist die Parameter den Attributen zu.

Methoden

- **doInBackground(urls: String...): String**
  Sichtbarkeit public

  Überschreibt doInBackground(..) von AsyncTask. Stellt die Netzwerkverbindung zum Server her, ruft dessen videoUpload(..) Methode auf und sendet die im Konstruktor spezifizierten Dateien.
- **onProgressUpdate(progress: Integer...)**
  Sichtbarkeit public

  Überschreibt onProgressUpdate(..) von AsyncTask. Ruft die onProgress(..) Methode des IServerResponseCallback auf.

- **onPostExecute(result: String)**
  Sichtbarkeit public

  Überschreibt onPostExecute(..) von AsyncTask. Ruft die onResponse(..) Methode des IServerCallbacks auf, falls die Netzwerkabfrage erfolgreich war. Ruft die onError(..) Methode des IServerCallbacks auf, falls ein Fehler auftrat.

3 Web-Dienst

3.1 Architektur

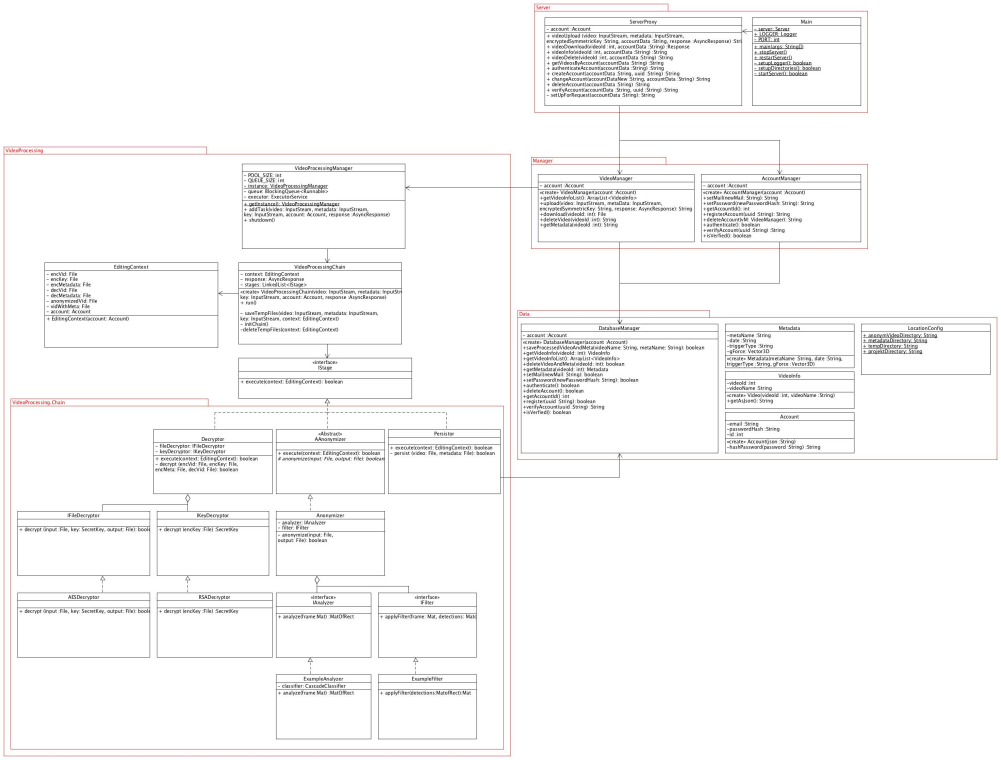

Abbildung 3.1: UML Diagramm des Web-Dienst

3.1.1 Multithreading

Um Anfragen vieler Nutzer unabhängig gleichzeitig zu bearbeiten, nimmt sich der Dienst für jede Anfrage einen einzelnen Thread. Unser verwendeter Webserver Jetty macht dies automatisch, indem er für jede Anfrage einen Thread aus einem internen ThreadPool nimmt, sofern vorhanden.
Da die Bearbeitung der Videos jedoch sehr zeitaufwändig ist, lagern wie die Bearbeitung auf einen eigenen Worker ThreadPool um (3.1.2), damit der Anfrage-Thread möglichst schnell seine Arbeit beenden kann und bereit für neue Anfragen ist.

3.1.2 Entwurfsmuster

Der Web-Dienst benutzt verschiedene Entwurfsmuster um vor allem zwei Ziele zu erreichen: 1. Eine schnelle und zeitlich unabhängige Bearbeitung von Anfragen und 2. Eine hohe Modularität und Austauschbarkeit.

Proxy

Die Webanbindung des Services wird in dem ServerProxy gekapselt. Der Proxy bearbeitet bzw. verschickt lediglich die Http-Anfragen und leitet alle weitere Arbeit weiter. Dies sorgt für eine höhere Austauschbarkeit, da so die Webanbindung unabhängig von der eigentlichen Bearbeitung der Anfragen ausgetauscht werden kann und umgekehrt.

Master-Worker

Da die Bearbeitung der Videos auf dem Web-Dienst rechen- und damit zeitintensiv ist, nutzt er einen eigenen Worker-ThreadPool zur Bearbeitung der Videos. Dadurch muss der für die Serveranfrage verwendete Thread nur ein neues VideoProcessingChain-Objekt erzeugen und in die Warteschlange des VideoProcessingManager einreihen. Diese kann dann direkt zurückkehren um neue Anfragen entgegenzunehmen. Dies erlaubt dem Server schneller und mehr externe Anfragen anzunehmen, bevor dieser blockiert.
Der VideoProcessingManager kann dann unabhängig von externen Anfragen auf seine eigenen Worker-Threads verteilen, um die Warteschlange abzuarbeiten.

Pipeline

Es gibt viele verschiedene Methoden Videos auf personenbezogene Daten zu analysieren. Dabei kann die Anzahl und Reihenfolge der durchlaufenen Arbeitsschritte stark variieren. Um den Web-Dienst in dieser Hinsicht möglichst flexibel zu gestalten, wird das Pipeline-Muster verwendet:
Jeder Arbeitsschritt muss die einheitliche Schnittstelle «Interface» IStage implementieren und somit eine Methode bereitstellen, die den Arbeitsschritt mithilfe des EditingContext ausführt.
Die VideoProcessingChain besitzt eine Liste von Arbeitsschritten, die sie, sofern keine Fehler entstehen, der Reihe nach ausführt. In diese Liste können jeder Zeit Arbeitsschritte eingefügt bzw. entfernt werden, die Reihenfolge der Arbeitsschritte verändert, oder der komplette Ablauf ausgetauscht werden. Theoretisch sind sogar Änderungen zur Laufzeit möglich.
Durch die Kapselung der Arbeitsschritte in einzelne Klassen ist es zudem möglich Arbeitsschritte in verschiedenen Ausführungsreihenfolgen wiederzuverwenden ohne, dass erneuter Initialisierungsaufwand entsteht.

Strategie

Da es für das, von uns bereitgestellte Bearbeitungschema für Videos, viele verschiedene Algorithmen für die einzelnen Schritte gibt, wird das Strategie-Muster verwendet:
Für die Schritte, die man eventuell austauschen möchte (z.B den Filter mit dem Bildbereiche unkenntlich gemacht werden) existiert eine Schnittstelle (z.B «Interface» IFilter). Dann können nach belieben konkrete Implementierungen (hier z.B. ExampleFilter) ausgetauscht werden, indem man bei der aufrufenden Klasse (hier: «Abstract» AAnonymizer). Theoretisch ist dies sogar zur Laufzeit möglich.

3.2 Datenhaltung

Die Daten des Web-Dienstes werden in einer Datenbank und in Ordnern organisiert. Nutzerdaten und Videodaten werden in Tabellen gespeichert. Als Syntax wird PostgreSQL verwendet.

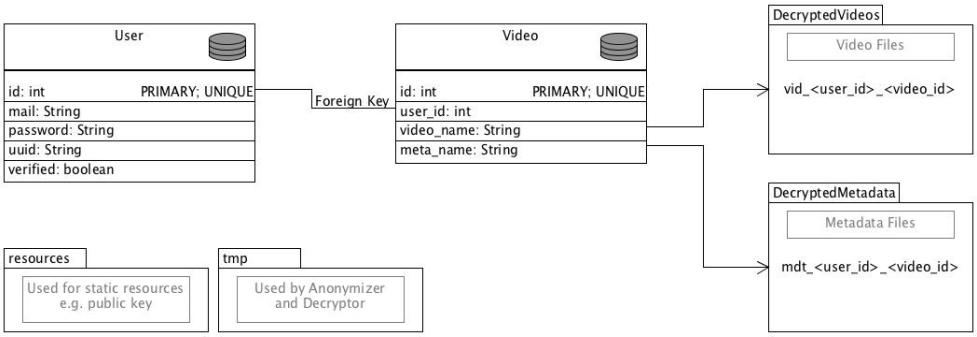

Abbildung 3.2: Datenbankschema

3.2.1 Datenbank

Die User-Tabelle für die Nutzerdaten speichert eine einzigartige Nutzer-Id, die Mail-Adresse, das Passwort und ob der Nutzer bereits seinen Account verifiziert hat.
Die Video-Tabelle für Videodaten speichert eine einzigartige Video-Id, die Nutzer-Id des Nutzers, der das Video hochgeladen hat sowie den Videodateiname und den Metadatendateiname. die Nutzer-Id agiert hier als Fremdschlüssel, die Dateinamen werden zum referenzieren der Dateien aus den Ordnern verwendet.
Video und Metadaten werden in jeweils einem eigenen Ordner organisiert. Dabei teilen sich Videodateien aller Nutzer einen Ordner und Metadatendateien aller Nutzer einen Ordner. Der Dateiname wird stets aus einem statischen Anfang und der Kombination aus Nutzer-Id und Video-Id zusammengesetzt und ist daher einzigartig.

3.2.2 Temporäre Dateien

Die Verwaltung der temporären Dateien für die Bearbeitung der Videos übernimmt das VideoProcessing Modul. Dies ist notwendig, da die entstehenden und benötigten Daten ausschließlich abhängig von den Arbeitsschritten der VideoProcessingChain ist und daher nicht von anderen Modulen des Web-Dienstes beeinflusst werden soll. Der Web-Dienst stellt dafür den *tmp* Ordner bereit. Es existieren keine Schnittstellen um von außen direkt auf Inhalte dieses Ordners zuzugreifen.

Der EditingContext erzeugt dann alle für die Bearbeitung notwendigen Dateipfade in der Form:
benutzerid_ videoname_ attribut.endung

3.2.3 Verwendete Resourcen

Für die Bearbeitung der Videos verwendete Ressourcen (z.B. der private Key des RSADecryptor oder der CascadeClassifier des ExampleAnalyzer) werden im "resources" Ordner abgelegt.

3.3 Modulübersicht

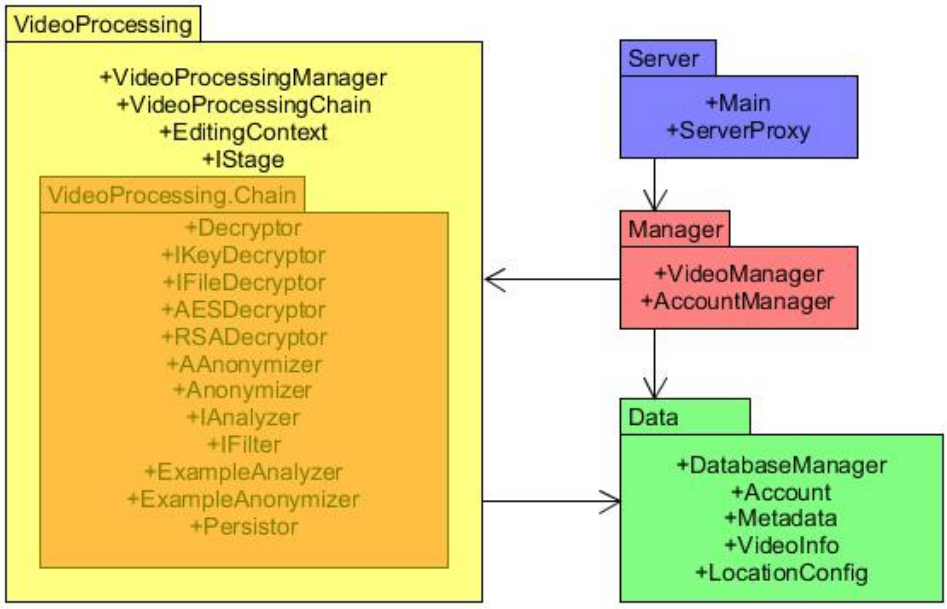

Abbildung 3.3: Modulübersicht

3.3.1 Server

Das Server Modul ist für das Starten und Verwalten des Servers zuständig. Außerdem empfängt es die Webanfragen von App und Web-Interface, entpackt sie und leitet sie an die bearbeitenden Klassen weiter.

3.3.2 Data

Das Data-Modul ist für die Verwaltung der Daten auf der Datenbank da und bietet eine Reihe von Hilfsklassen an, die das Behandeln von zusammengehörenden Daten vereinfacht.

3.3.3 Manager

Das Manager-Paket beinhaltet die Presenter-Klassen des Web-Dienstes. Diese dienen dazu die Bearbeitung von Anfragen zu initiieren. So werden Accounts angelegt oder überprüft, hochgeladene Videos zur Bearbeitung weitergegeben, oder Informationen aus der Datenbank abgefragt.

3.3.4 VideoProcessing

Das VideoProcessing-Modul ist ein elementares Modul der Privacy-Crash-Cam. Es ist für die Anonymisierung der, von der App hochgeladenen Videos zuständig. Das Modul ist in ein Haupt- und ein Unterpaket gegliedert. Das Hauptpaket übernimmt die Verwaltung der Anfragen und die Verteilung der Rechenarbeit auf die vorhandenen Ressourcen. Das Unterpaket VideoProcessing.Chain beinhaltet alle Klassen, die die Bearbeitung des Videos umsetzen.

3.3.5 VideoProcessing.Chain

Das Paket VideoProcessing.Chain beinhaltet alle Klassen, die von der VideoProcessingChain verwendet werden um ein hochgeladenes Video zu bearbeiten.

3.4 Klassenübersicht

3.4.1 Server

Main

Die Klasse Main ist der Haupteinstiegspunkt des Web-Dienstes. Sie sorgt dafür bei Serverstart alles korrekt zu starten und ist dafür zuständig den Server zu stoppen und neu zu starten.

Attribute

- **server: Server**
  Sichtbarkeit private
  static

  Server des Web-Dienstes.
- **LOGGER: Logger**
  Sichtbarkeit public
  static

  Globaler Logger des Web-Dienstes.
- **PORT: Server**
  Sichtbarkeit private
  static

  Port des Servers.

Methoden

- **main (args: String[ ]): void**
  Sichtbarkeit public
  statisch

  Haupteinstiegspunkt des Web-Dienstes. Startet den Logger, erstellt alle Ordner falls nötig und löscht die temporären Dateien.
- **stopServer (): void**
  Sichtbarkeit public
  statisch

  Stoppt den Server.
- **restartServer (): void**
  Sichtbarkeit public
  statisch

  Stoppt den Server setzt ihn neu auf und startet ihn wieder.
- **setupLogger (): void**
  Sichtbarkeit private
  statisch

Konfiguriert den globalen Logger und startet ihn.

- **setupDirectories (): void**
  Sichtbarkeit private
  statisch

  Erstellt alle benötigten Ordner, falls diese noch nicht vorhanden sind und löscht alle temporären Dateien.

- **startServer (): void**
  Sichtbarkeit private
  statisch

  Konfiguriert und startet den Server.

ServerProxy

Der ServerProxy ist die Schnittstelle zwischen Web-Dienst und Web-Interface/App. Die Anfragen an den ServerProxy werden mittels der REST-API übermittelt. Die Anfragen werden in Teilanfragen aufgespalten und mithilfe des Manager-Moduls weiter bearbeitet.
Siehe auch: Authentifizieren auf dem Dienst, Video auf den Dienst hochladen, Videodownload vom Dienst, Video auf dem Dienst löschen

Attribute

- **account: Account**
  Sichtbarkeit private

  Instanz des Accounts, der die Anfrage geschickt hat. Wird in der Methode setUpForRequest() initialisiert.

Konstruktoren

- **ServerProxy()**
  Standardkonstruktor

Methoden

- **videoUpload (video: InputStream, metadata: InputStrem, encryptedSymmetricKey: String, accountData: String, response: AsyncResponse): String**
  Sichtbarkeit public

  Bearbeitet eine VideoUpload Anfrage der App. Übergibt die videobezogenen Parameter (video, metadata und encryptedSymmetricKey) und die AsyncResponse an die upload()-Methode im VideoManager. Die Rückgabe als JSON-String gibt an, ob die Dateien vollständig angekommen sind.

- **videoDownload(videoId: int, accountData: String): Response**
  Sichtbarkeit public

  Leitet einen Download-Anfrage ein. Gibt die VideoId an die download()-Methode des VideoManagers weiter. Mithilfe der Id wird dann das gesuchte Video gefunden. Die Response-Rückgabe ermöglicht den bekannten Download-Dialog auf dem Interface.

- **videoInfo(videoId: int, accountData: String): String**
  Sichtbarkeit public

  Bearbeitet eine Anfrage zur Informationsausgabe eines Videos des Web-Interfaces. Dabei werden die Metadaten des zugehörigen Videos über den VideoManager mithilfe der VideoId gefunden. Die String Rückgabe beinhaltet die relevanten Video Informationen.

- **videoDelete(videoId: int, accountData: String): String**
  Sichtbarkeit public

Bearbeitet eine Anfrage zum Löschen eines Videos des Web-Interfaces. Diese wird über die deleteVideo()-Methode des VideoManagers weiter bearbeitet. Die String Rückgabe gibt Meldung über das Ergebnis der Anfrage.

- **getVideosByAccount(accountData: String): String**
  Sichtbarkeit public

  Bearbeitet eine Anfrage zur Videolistenrückgabe des Web-Interface für einen bestimmten Nutzer. Dabei wird im VideoManager die getVideoList-Methode aufgerufen. Diese bearbeitet die Anfrage und gibt einen JSON-String mit den Videoinformationen zurück. Dieser JSON-String wird aus dem ArrayList<VideoInfo> erstellt. Diese werden dann auch an das Web-Interface zurückgegeben.

- **authenticateAccount(accountData: String): String**
  Sichtbarkeit public

  Bearbeitet eine Anfrage zur Authentifizierungs des Web-Interfaces oder App. Dabei wird auf die Rückgabe von setUpForRequest() gewartet und der String dann an die anfragende Instanz weitergeleitet.

- **createAccount(accountData: String uuid :String): String**
  Sichtbarkeit public

  Bearbeitet eine Anfrage zur Accounterstellung des Web-Interfaces. Die Die Übergabe "uuid" stellt eine Eindeutige id des Accounts dar, die zur Accountverifizierung dient. Dabei wird im AccountManager die Methode registerAccount(uuid :String) aufgerufen. Die Rückgabe gibt Meldung über das Ergebnis der Anfrage an das Web-Interface.

- **changeAccount(accountDataNew: String, accountData: String): String**
  Sichtbarkeit public

  Bearbeitet eine Anfrage zur Passwort/Mailänderung des Web-Interfaces. Dabei werden im AccountManager die Methoden setMail() und setPassword() aufgerufen. Die Rückgabe gibt Meldung über das Ergebnis der Anfrage an das Web-Interface.

- **deleteAccount(accountData: String): String**
  Sichtbarkeit public

  Bearbeitet eine Anfrage zur Löschung eines Accounts des Web-Interfaces. Es wird im AccountManager die Methode deleteAccount(vM :VideoManager) aufgerufen. Die Rückgabe gibt Meldung über das Ergebnis der Anfrage an das Web-Interface.

- **verifyAccount(accountData :String, uuid: String): String**
  Sichtbarkeit public

  Bearbeitet eine Anfrage zur Verifizierung der E-Mail eines Accounts. Die Übergabe "uuid" stellt eine eindeutige Id des Accounts dar. Die Methode

gibt einen JSON-String mit dem Ergebnis zurück.

- **setUpForRequest(accountData: String)**
  Sichtbarkeit private

  Bei jeder Anfrage wird zu Beginn die private Methode setUpForRequest() aufgerufen, um die Klassen des Manager-Moduls zu initialisieren und die Authentifizierung des Accounts zu gewährleisten. Die Rückgabe gibt Meldung über das Ergebnis der Authentifizierung des Accounts. Die Authentifizierung ruft im AccountManager die Methoden getAccountId(), authenticate() und isVerified() auf.

3.4.2 Data

DatabaseManager

Die Klasse DatabaseManager bietet eine Schnittstelle für alle Datenbankanfragen, die vom AccountManager und VideoManager benötigt wird.

Attribute

- **account: Account**
  Sichtbarkeit private

  Benutzeraccount des aktiven Nutzers.

Konstruktoren

- **DatabaseManager(account: Account)**
  Setzt den aktiven Nutzer.

Methoden

- **saveProcessedVideoAndMeta(videoName: String, metaName: String): boolean**
  Sichtbarkeit public

  Bekommt den Videofile-Namen, sowie die Metadaten als übergeben und schreibt einen Datenbank-Eintrag in die Video-Tabelle der Datenbank. Die "id" wird generiert, die "user_id" ist das Attribut "id" des Account-Objektes, der "video_name" ist der Übergabeparameter "videoName" und "meta_name" ist der Übergabeparameter "metaName".
- **getVideoInfo(videoId: int): VideoInfo**
  Sichtbarkeit public

  Bekommt die videoId übergeben und gibt Videoinformationen als Video-Info-Objekt zurück. Die Daten werden mithilfe der videoId und den Accountdaten des Nutzers aus der Datenbank geladen.
- **getVideoInfoList(): ArrayList<VideoInfo>**
  Sichtbarkeit public

  Gibt alle Videos eines Benutzers in Form einer Liste von "VideoInfo"-Objekten zurück. Diese wird durch eine Datenbankabfrage mithilfe der Accountdaten ermittelt.
- **deleteVideoAndMeta(videoId: int): boolean**
  Sichtbarkeit public

  Löscht ein Video eines Benutzers. Das Löschen geschieht durch eine Datenbankabfrage mithilfe der videoId und den Accountdaten des aktiven Benutzers. Die Methode gibt zurück, ob das Löschen erfolgreich war.

- **getMetadata(videoId: int): Metadata**
  Sichtbarkeit public

  Ermittelt die Metadaten eines Videos durch eine Datenbankabfrage. Dafür wird die videoId und die Benutzerdaten genutzt.

- **setMail(newMail: String): boolean**
  Sichtbarkeit public

  Die Methode ändert die E-Mail-Adresse des Benutzers. Die E-Mail-Adresse wird als "newMail" übergeben und mithilfe des Accounts kann auf das Element in der Datenbank zugegriffen werden. Mit dem entsprechenden Datenbankbefehl wird die neue E-Mail gesetzt. Es wird zurückgegeben, ob das aktualisieren erfolgreich war, oder nicht.

- **setPassword(newPassword: String): boolean**
  Sichtbarkeit public

  Die Methode ändert das Passwort des Benutzers. Der neue Passwort-Hash wird als Parameter "newPasswordHash" übergeben und mithilfe des Accounts kann durch ein Datenbankbefehl der neue Passwort-Hash gesetzt werden. Ein "boolean" wird zurückgegeben, je nachdem, ob die Operation erfolgreich war oder nicht.

- **authenticate(): boolean**
  Sichtbarkeit public

  Die Methode authentifiziert den Benutzer. Durch das den Account stehen alle benötigten Informationen zur Verfügung. Durch eine Datenbankabfrage wird überprüft, ob E-Mail und Passwort-Hash übereinstimmen. Es wird zurückgegeben, ob das aktualisieren erfolgreich war, oder nicht.

- **deleteAccount(): boolean**
  Sichtbarkeit public

  Die Methode löscht einen Account. Durch den Account stehen alle Informationen zur Verfügung. Zunächst werden alle Videos und Metadaten des Nutzers gelöscht. Danach werden die Video-Datenbankeinträge in der Tabelle "Video" und dann der Account in der Tabelle "User" gelöscht.

- **getAccountId() :int**
  Sichtbarkeit public

  Die Methode gibt die "id" des Accounts zurück. Diese ermittelt sie durch eine Datenbankabfrage mit der Adresse des Accounts. Falls der Account nicht existiert, wird "-1" zurückgegeben.

- **register(uuid :String) :boolean**
  Sichtbarkeit public

  Die Methode legt einen neuen Benutzer an. Die Die Übergabe "uuid" ist ein eindeutiger Wert, der zur Accountverifizierung dient. Die zur Erstellung des Account benötigten Informationen liegen in dem Attribut account

vor. Daraus werden E-Mail und Passwort genommen und mit einer Datenbankabfrage wird in der Tabelle “User” ein neuer Eintrag hinzugefügt.

- **verifyAccount(accountData :String, uuid: String): String**
  Sichtbarkeit public

  Bearbeitet eine Anfrage zur Verifizierung der E-Mail eines Accounts. Die Übergabe “uuid” stellt eine eindeutige Id des Accounts dar. Die Methode gibt einen JSON-String mit dem Ergebnis zurück.

- **isVerfied(): boolean**
  Sichtbarkeit public

  Die Methode überprüft, ob ein Nutzer verifiziert ist.

Account

Die Klasse Account repräsentiert den Account eines Benutzers.

Attribute

- **email: String**
  Sichtbarkeit private

  E-Mail-Adresse des Accounts.
- **passwordHash: String**
  Sichtbarkeit private

  PasswordHash des Accounts.
- **id: int**
  Sichtbarkeit private

  Die "id" ist identisch mit der "id" in der Tabelle User der Datenbank.

Konstruktoren

- **Account(json: String)**
  Sichtbarkeit public

  Der Konstruktor nimmt ein JSON-Objekt entgegen, wertet die Informationen aus und setzt die Attribute. Die Information "password" wird mit der Methode "hashPassword(password: String)" gehasht und dann gesetzt.

Methoden

- **hashPassword(password: String): String**
  Sichtbarkeit private

  Die Methode nimmt das im Klartext gegebene Passwort und hasht dieses.

Metadata

Die Klasse Metadata beinhaltet alle wichtigen Informationen zu den Metadaten eines Videos.

Attribute

- **date: String**
  Sichtbarkeit private

  Datum, an dem das Video aufgenommen wurde.
- **triggerType: String**
  Sichtbarkeit private

  Art, wie das Video ausgelöst wurde (G-Sensor, manuelle Auslösung).
- **gForce: Vector3D**
  Sichtbarkeit private

  Messwerte des G-Sensors zum Auslösezeitpunkt.

Konstruktoren

- **Metadata(metaName :String, date :String, triggerType :String, gForce :Vector3D)**
  Sichtbarkeit public

  Weist die Parameter den Attributen zu.

Methoden

- **getAsJson(): String**
  Sichtbarkeit public

  Die Methode gibt ein JSON-String zurück, welcher aus den Informationen aller Klassenattributen zusammengesetzt wird.

VideoInfo

Die Klasse VideoInfo beinhaltet die benötigten Informationen zum Video.

Attribute

- **videoId: int**
  Sichtbarkeit private

  Die "videoId" ist die "videoId" des zugehörigen Datenbankeintrags.
- **videoName: String**
  Sichtbarkeit private

  Der "videoName" ist der Name des Videos im zugehörigen Datenbankeintrag.

Konstruktoren

- **Video(videoId: int, videoName: String)**

  Weist die Parameterübergabe den zugehörigen Klassenattributen zu.

Methoden

- **getAsJson(): String**
  Sichtbarkeit public

  Die Methode gibt ein JSON-String zurück, der aus den Informationen aller Attribute zusammengesetzt wird.

LocationConfig

Die Klasse LocationConfig beinhaltet alle Pfade als statischen Strings, die für das Projekt benötigt werden.

Attribute

- **anonymVideoDirectory: String**
  Sichtbarkeit public
  statisch

  Pfad zu allen anonymisierten Videos.
- **metadataDirectory: String**
  Sichtbarkeit public
  statisch

  Pfad zu allen entschlüsselten Metadaten.
- **projectDirectory: String**
  Sichtbarkeit public
  statisch

  Pfad der temporären Dateien.
- **projectDirectory: String**
  Sichtbarkeit public
  statisch

  Pfad des ausgeführten Projekts.

3.4.3 Manager

AccountManager

Verbindet zwischen dem Data-Modul und dem Server-Modul. Er bearbeitet die Anfragen des ServerProxys und gibt die gefragten Ergebnisse zurück.

Attribute

- **account: Account**
  Sichtbarkeit public

  Account-Instanz, welche zur Bearbeitung der verschiedenen Anfragen benötigt wird.

Konstruktoren

- **AccountManager(account: Account)**
  Sichtbarkeit public

  Erstellt eine Instanz eines AccountManagers mit dem dazugehörigen Account. Dieser wird vom ServerProxy über die Parameter übergeben und dann gesetzt.

Methoden

- **setMail(newMail: String): String**
  Sichtbarkeit public

  Bearbeitet eine Anfrage vom ServerProxy zur Ersetzung einer Mail eines Accounts. Die übergebene Mail wird dann im DatabaseManager mit der Methode setMail() gesetzt.
- **setPassword(newPasswordHash: String): String**
  Sichtbarkeit public

  Bearbeitet eine Anfrage vom ServerProxy zur Ersetzung eines Passwort eines Accounts. Das übergebene Passwort wird dann im DatabaseManager mit der Methode setPassword() gesetzt.
- **getAccountId(): int**
  Sichtbarkeit public

  Bearbeitet eine Anfrage vom ServerProxy zur Bestimmung der AccountId zu einer Mail-Adresse. Die Anfrage wird zum DatabaseManager weitergeleitet und somit die Methode getAccountId() aufgerufen. Von dort aus wird sie bis zum ServerProxy zurückgegeben.
- **registerAccount(uuid :String): String**
  Sichtbarkeit public

  Bearbeitet eine Anfrage vom ServerProxy zur Accountregistrierung. Die Die Übergabe “uuid” stellt eine eindeutige Id des Accounts dar, die zur Accountverifizierung dient. Dabei wird beim DatabaseManager die Methode

register() aufgerufen und die Accountdaten im zugehörigen Accountattribut gesetzt.

- **deleteAccount(vm :VideoManager): String**
  Sichtbarkeit public

  Bearbeitet die Löschung eines Accounts. Zunächst werden alle Videos des Accounts ermittelt und es wird mit dem Übergabeparameter "vM" auf den VideoManager zugegriffen, der alle Videos und Metadaten löscht. Nun wird im DatabaseManager "deleteAccount()" aufgerufen und der Account wird in der Datenbank gelöscht.

- **authenticate(): boolean**
  Sichtbarkeit public

  Bearbeitet eine Anfrage vom ServerProxy zur Authentifizierung des Accounts. Die Methode authenticate() vom DatabaseManager wird hierbei aufgerufen.

- **verifyAccount(accountData :String, uuid: String) :String**
  Sichtbarkeit public

  Bearbeitet eine Anfrage zur Verifizierung der E-Mail eines Accounts. Die Übergabe "uuid" stellt eine eindeutige Id des Accounts dar. Die Methode gibt einen JSON-String mit dem Ergebnis zurück.

- **isVerfied(): boolean**
  Sichtbarkeit public

  Bearbeitet eine Anfrage vom ServerProxy zum Verifizierungsstatus eines Accounts. Die Anfrage wird zum DatabaseManager weitergeleitet und die Methode isVerified() aufgerufen. Von dort wird der boolean bis zum ServerProxy zurückgegeben.

VideoManager

Der VideoManager verbindet zwischen dem Server-Modul und dem Data- bzw. VideoProcessing-Modul. Er bearbeitet die Anfragen des ServerProxys und gibt die gefragten Ergebnisse zurück.
Siehe auch: Videodownload vom Dienst, Video auf dem Dienst löschen, Video auf den Dienst hochladen

Attribute

- **account: Account**
  Sichtbarkeit private

  Account-Instanz, welche zur Bearbeitung der verschiedenen Anfragen benötigt wird.

Konstruktoren

- **VideoManager(account :Account)**
  Sichtbarkeit public

  Erstellt eine Instanz eines VideoManagers mit dem dazugehörigen Account. Dieser wird vom ServerProxy über die Parameter übergeben und dann gesetzt.

Methoden

- **getVideoInfoList(): ArrayList<VideoInfo>**
  Sichtbarkeit public

  Bearbeitete eine Anfrage vom ServerProxy zur Videolistenrückgabe eines Accounts. Dabei wird im DatabaseManager die Methode getVideoInfoList() aufgerufen. Die ArrayList Der VideoInfo-Objekte wird dann an den ServerProxy zurückgegeben.
- **upload(video: InputStream, metaData: InputStream, encryptedSymmetricKey: String, response: AsyncResponse): String**
  Sichtbarkeit public

  Bearbeitet eine Anfrage vom ServerProxy zum Hochladen mehrerer Dateien. Da der Upload komplett vom VideoProcessing-Modul übernommen wird, werden hier alle übergebenen Parameter und der Account beim Aufruf der Methode addTask() vom VideoProcessingManager übergeben.
- **download(videoId: int): File**
  Sichtbarkeit public

  Bearbeitet eine Anfrage vom ServerProxy zum Herunterladen eines Videos. Dabei wird vom DatabaseManager die Methode getVideoInfo() aufgerufen, um mittels der VideoId den Videonamen über das VideoInfo Objekt zu bekommen. Somit kann dann das gewünschte File gefunden und zurückgegeben werden.

- **deleteVideo(videoId :int): String**
  Sichtbarkeit public

  Bearbeitet eine Anfrage vom SererProxy zum Löschen eines Videos. Zunächst wird die Methode getVideoInfo() und getMetadata() des DatabaseManagers aufgerufen. Somit bekommen wir über das VideoInfo-Objekt und Metadata-Objekt den Videonamen und Metadatanamen und können diese Files löschen. Daraufhin wird die Methode deleteVideoAndMeta() des DatabaseMangers aufgerufen, welche die beiden Einträge aus der Datenbank löscht.

- **getMetadata(videoId :int): String**
  Sichtbarkeit public

  Bearbeitet die Anfrage vom ServerProxy zum Ausgeben der Metadata eines Videos. Es wird im DatabaseManager die Methode getMetadata() aufgerufen, die ein Metadata-Objekt zurückgibt. Die relevanten Attribute werden in ein JSON-String umgewandelt und dann an den ServerProxy zurückgeben.

3.4.4 VideoProcessing

VideoProcessingManager

Der VideoProcessingManager ist dafür zuständig, die Bearbeitung der hochgeladenen Videos zu koordinieren und auf seine Worker-Threads zu verteilen. Nachdem er vom VideoManager eine Anfrage erhalten hat wird eine VideoProcessingChain erzeugt und in die Warteschlange des Managers eingereiht, aus der die Worker-Threads kontinuierlich Anfragen abarbeiten. Das Ganze passiert asynchron zu der ursprünglichen Anfrage, damit das aufwändige Bearbeiten der Videos nicht die Serveranfragen aufhält. Dafür werden die von der Java API bereitgestellten java.util.concurrent Objekte verwendet.

Attribute

- **POOL_SIZE: int**
  Sichtbarkeit private
  statisch

  Größe des Worker-Thread-Pools.
- **QUEUE_SIZE: int**
  Sichtbarkeit private
  statisch

  Größe der Warteschlange für die Anfragen.
- **instance: VideoProcessingManager**
  Sichtbarkeit private
  statisch

  Einzelstück des VideoProcessingManagers.
- **queue: BlockingQueue**
  Sichtbarkeit private

  Warteschlange für die Anfragen.
- **executor: ExecutorService**
  Sichtbarkeit private

  Koordinator für die Verteilung der Aufgaben auf die Worker-Threads.

Konstruktoren

- **VideoProcessingManager(poolSize: int, queueSize: int)**
  Sichtbarkeit private

  Erzeugt einen neuen VideoProcessingManager mit der angegeben Anzahl Worker-Threads und Größe der Warteschlange. Kann nicht von außen zugegriffen werden, damit sichergestellt ist, dass der VideoProcessingManager ein Einzelstück ist.

Methoden

- **getInstance (): VideoProcessingManager**
  Sichtbarkeit public
  statisch

  Erzeugt einen neuen VideoProcessingManager, falls nötig und gibt das Einzelstück zurück.

- **addTask (video: InputSteam, metadata: InputStream, key: InputStream, account: Account, response :AsyncResponse)**
  Sichtbarkeit public

  Erzeugt aus den gegebenen Parametern eine neue VideoProcessingChain und fügt sie der Warteschlange hinzu. Rückmeldung über Erfolg oder Fehler werden über den response Parameter zurückgegeben. Das Verteilen auf Worker-Threads übernimmt der ExecutorService automatisch.

- **shutdown (): void**
  Sichtbarkeit public

  Beendet den ExecutorService nachdem alle Worker-Threads ihre Arbeit beendet haben.

VideoProcessingChain

implements Runnable
Die VideoProcessingChain ist für die Bearbeitung der Videos zuständig. Zunächst werden alle für die Bearbeitung notwendigen Informationen ermittelt. Dann nimmt die VideoProcessingChain die hochgeladenen Videos entgegen und speichert sie zunächst temporär. Das Video wird entschlüsselt, auf personenbezogene Daten analysiert und diese unkenntlich gemacht. Danach können die Metadaten dem Video hinzugefügt und dieses auf dem Server langfristig hinterlegt werden. Zum Schluss werden die temporären Daten wieder gelöscht.
Siehe auch: Videobearbeitung auf dem Dienst

Attribute

- **context: EditingContext**
  Sichtbarkeit private

  Container für die Informationen, die die einzelnen Arbeitsschritte zum Arbeiten benötigen.
- **response: AsyncResponse**
  Sichtbarkeit private

  Objekt, das genutzt wird um die Anfrage der App bei Fehler oder Erfolg asynchron zu beantworten.
- **stages: List<«Interface» IStage>**
  Sichtbarkeit private

  Liste der zur Bearbeitung des Videos auszuführenden Arbeitsschritte.

Konstruktoren

- **VideoProcessingChain(video: InputSteam, metadata: InputStream, key: InputStream, account: Account, response :AsyncResponse)**
  Sichtbarkeit public
  throws IOException

  Erzeugt eine neue VideoProcessingChain, merkt sich das response Objekt und erzeugt aus den Parametern den EditingContext. Zudem werden alle Dateien temporär gespeichert. Wirft eine IOException falls dies fehlschlägt.

Methoden

- **run (): void**
  Sichtbarkeit public

  Überschreibt die Methode run() von Runnable. Zunächst werden die für die Bearbeitung des Videos notwendigen Arbeitsschritte erzeugt. Danach führt die VideoProcessingChain alle Arbeitsschritte aus. Zuletzt werden alle zuvor erzeugten temporären Dateien wieder gelöscht.

- **saveTempFiles (video: InputSteam, metadata: InputStream, key: InputStream, context: EditingContext): void**
  Sichtbarkeit private
  throws IOException

  Speichert alle von der App erhaltenen Daten temporär. Wirft eine IOException falls dies fehlschlägt.

- **initChain (): void**
  Sichtbarkeit private

  Erzeugt alle Arbeitsschritte.

- **deleteTempFiles (context: EditingContext): void**
  Sichtbarkeit private

  Löscht alle zuvor erstellten temporären Dateien.

EditingContext

Der EditingContext ist eine Containerklasse, die alle für die Bearbeitung der Videos benötigten Daten hält.

Attribute

- **encVid: File**
  Sichtbarkeit private

  Ort der verschlüsselten Videodatei.
- **encKey: File**
  Sichtbarkeit private

  Ort der verschlüsselten symmetrischen Schlüssels.
- **encMetadata: File**
  Sichtbarkeit private

  Ort der verschlüsselten Metadaten.
- **decVid: File**
  Sichtbarkeit private

  Ort der entschlüsselten Videodatei.
- **decMetadata: File**
  Sichtbarkeit private

  Ort der entschlüsselten Metadaten.
- **anonymizedVid: File**
  Sichtbarkeit private

  Ort des anonymisierten Videos.
- **vidWithMeta: File**
  Sichtbarkeit private

  Ort des Videos nachdem die Metadaten hinzugefügt wurden.
- **account: Account**
  Sichtbarkeit private

  Dem Video zugehöriger Benutzeraccount.

Konstruktoren

- **EditingContext(account: Account)**
  Sichtbarkeit public

  Erzeugt aus den Account-Daten des Benutzers alle benötigten temporären Dateipfade.

«Interface» IStage

IStage ist eine einheitliche Schnittstelle für alle Videobearbeitungsschritte. Dadurch wird es erlaubt einfach neue Arbeitsschritte einzufügen oder dynamisch die Bearbeitung zu verändern.

Methoden

- **execute (context: EditingContext): boolean**
  Sichtbarkeit public

  Führt einen Bearbeitungsschritt aus. Die dafür benötigten Daten werden durch den context bereitgestellt. Gibt zurück ob die Bearbeitung erfolgreich war.

3.4.5 VideoProcessing.Chain

Decryptor

implements «Interface» IStage
Der Decryptor ist dafür da, das hybride Verschlüsselungsverfahren der App (2) rückgängig zu machen.

Attribute

- **fileDecryptor**
  Sichtbarkeit private

  Decryptor für symmetrisch verschlüsselte Dateien.
- **keyDecryptor**
  Sichtbarkeit private

  Decryptor für asymmetrisch verschlüsselte SecretKeys.

Konstruktoren

- **Decryptor()**
  Sichtbarkeit public

  Standardkonstruktor

Methoden

- **execute (): boolean**
  Sichtbarkeit public

  Überschreibt die Methode execute() von IStage. Ruft decrypt(..) auf.
- **decrypt (encVid: File, encKey: File, encMeta: File, decVid: File): boolean**
  Sichtbarkeit public

  Erzeugt zunächst den symmetrischen und asymmetrischen Decryptor. Gibt daraufhin den verschlüsselten SecretKey an den keyDecryptor und entschlüsselt mit diesem dann im fileDecryptor die Video- und die Metadata-Datei. Gibt zurück, ob das Entschlüsseln erfolgreich war.

«Interface» IKeyDecrytor

IKeyDecryptor bietet eine Schnittstelle für asymmetrische Decryptoren. Es wird verwendet um verschiedene Entschlüsselungsverfahren für das Entschlüsseln des SecretKeys bei hybrider Verschlüsselung anzubieten.

Methoden

- **decrypt (encKey :File): SecretKey**
  Sichtbarkeit public

  Nimmt den privaten Schlüssel des Webservers und entschlüsselt damit den asymmetrisch verschlüsselten SecretKey. Gibt den entschlüsselten SecretKey zurück. Gibt null zurück, falls das Entschlüsseln fehlschlägt.

RSADecryptor

implements «Interface» IKeyDecrytor
Der RSADecryptor bietet eine konkrete Implementierung eines asymmetrischen Decryptors für symmetrisch verschlüsselte SecretKeys. Er nutzt dazu das RSA Verfahren.

Konstruktoren

- **RSADecryptor()**
  Sichtbarkeit public

  Standardkonstruktor

Methoden

- **decrypt (encKey: File): SecretKey**
  Sichtbarkeit public

  Implementiert die Methode decrypt(..) von IKeyDecryptor.

«Interface» IFileDecryptor

IFileDecryptor bietet eine Schnittstelle für Klassen, die mithilfe eines symmetrischen SecretKeys Dateien entschlüsseln.

Methoden

- **decrypt (input :File, key: SecretKey, output: File): boolean**
  Sichtbarkeit public

  Nimmt das input-File und entschlüsselt es mit dem SecretKey. Speichert die entschlüsselte Datei im output-File. Gibt zurück, ob die Entschlüsselung erfolgreich war.

AESDecryptor

implements «Interface» IFileDecryptor
Der AESDecryptor bietet eine konkrete Implementierung eines Decryptors für symmetrisch verschlüsselte Dateien. Er nutzt dazu das AES Verfahren mit dem gegebenen SecretKey.

Konstruktoren

- **AESDecryptor()**
  Sichtbarkeit public

  Standardkonstruktor

Methoden

- **decrypt (input: File, key: SecretKey, output: File): boolean**
  Sichtbarkeit public

  Implementiert die Methode decrypt(..) von IFileDecryptor.

«Abstract» AAnonymizer

implements «Interface» IStage
IAnonymizer bietet eine Schnittstelle für Video-Anonymisierungsverfahren. Die Schnittstelle ist bewusst sehr allgemein gehalten, um verschiedenste Anonymisierungsverfahren zu erlauben.

Methoden

- **execute (context: Context): boolean**
  Sichtbarkeit public

  Implementiert die Methode execute(..) von IStage. Ruft die Methode anonymize(..) auf.

- **«abstract» anonymize (input: File, output: File): boolean**
  Sichtbarkeit public

  Analysiert zunächst das input-Video um Bildbereiche zu identifizieren, die personenbezogene Daten zeigen. Wendet daraufhin einen Bildfilter auf die Bildbereiche an, um diese unkenntlich zu machen.

Anonymizer

extends «Abstract» AAnonymizer
Der Anonymizer bietet eine mögliche konkrete Implementierung eines Video-Anonymizers. Er arbeitet indem er das Video frameweise analysiert, um dann die für die Anonymisierung relevanten Bildbereiche mit einem Filter unkenntlich zu machen. Genutzt werden hierfür OpenCV Schnittstellen.

Attribute

- **analyzer: IAnalyzer**
  Sichtbarkeit private

  Analysierungsalgorithmus zum Erkennen der für die Anonymisierung relevanten Bildbereiche.
- **filter: IFilter**
  Sichtbarkeit private

  Bildfilter zum Anonymisieren von Bildbereichen.

Konstruktoren

- **Anonymizer()**
  Sichtbarkeit public

  Standardkonstruktor

Methoden

- **anonymize(input: File, output: File): boolean**
  Sichtbarkeit public

  Implementiert die abstrakte Methode anonymize() von AAnonymizer. Liest das Video frameweise ein und über gibt den Frame zunächst an den analyzer. Die dort erkannten relevanten Bildbereiche werden dann an den filter gegeben. Das Video wird dann wieder Frame für Frame zusammengeführt und gespeichert. Gibt zurück, ob das anonymisieren erfolgreich war.

«Interface» IAnalyzer

Der IAnalyzer bietet eine Schnittstelle für Klassen, die in einzelnen Frames mithilfe von OpenCV-Algorithmen relevante Bildbereiche erkennen.

Methoden

- **analyzer (frame: Mat): MatOfRect**
  Sichtbarkeit public

  Analysiert den übergebenen Frame und gibt eine Liste der gefundenen relevanten Bildbereiche zurück. Im Fehlerfall wird Null zurückgegeben.

ExampleAnalyzer

implements «Interface» IAnalyzer
Der ExampleAnalyzer bietet eine konkrete Implementierung eines IAnalyzers. Er analysiert einen einzelnen Frame eines Videos mithilfe der in OpenCV bereitgestellten CascadeClassifier und gibt die dort erkannten Gesichter als Liste von Bildausschnitten zurück.

Attribute

- **classifier: CascadeClassifier**
  Sichtbarkeit private

  Liefert den Algorithmus zur Gesichtserkennung.

Konstruktoren

- **ExampleAnalyzer()**
  Lädt den classifier aus einer XML-Datei.

Methoden

- **analyze (frame: Mat): MatOfRect**
  Sichtbarkeit public

  Implementiert die Methode analyze(..) von IAnalyzer.

«Interface» IFilter

Bietet eine Schnittstelle für Klassen die eine Liste von Bildausschnitten entgegen nimmt und diese unkenntlich macht.

Methoden

- **applyFilter (frame: Mat, detections: MatOfRect): Mat**
  Sichtbarkeit public

  Macht alle Bildausschnitte im Ursprungsbild unkenntlich und gibt das Ergebnis zurück. Im Fehlerfall wird null zurückgegeben.

ExampleFilter

implements «Interface» IFilter
Der ExampleAnalyzer bietet eine konkrete Implementierung eines IFilters. Er bearbeitet die erkannten Bildbereiche mithilfe eines OpenCV-Blur-Filters, sodass die Bildbereich unkenntlich gemacht werden.

Konstruktoren

- **ExampleFilter()**
  Standardkonstruktor

Methoden

- **applyFilter (frame: Mat, detections: MatOfRect): Mat**
  Sichtbarkeit public

  Implementiert die Methode applyFilter(..) von IFilter.

Persistor

implements «Interface» IStage
Der Persistor fügt die Metadaten in die Video-Datei ein und legt das Video langfristig auf dem Video ab. Dazu sogt er dafür, dass das Video in die Database aufgenommen wird.

Konstruktoren

- **Persistor()**

  Standardkonstruktor

Methoden

- **execute (context: Context): boolean**
  Sichtbarkeit public

  Implementiert die Methode execute(..) von IStage. Ruft die Methode anonymize(..) auf.

- **persist (video: File, metadata: Metadata): boolean**
  Sichtbarkeit public

  Lädt das Video und fügt alle Metadaten in die Metadaten des Videos ab. Daraufhin wird das Video mithilfe des DatabaseManagers in den permanenten Speicher des Services abgelegt und der Datenbank hinzugefügt.

4 Web-Interface

4.1 Architektur

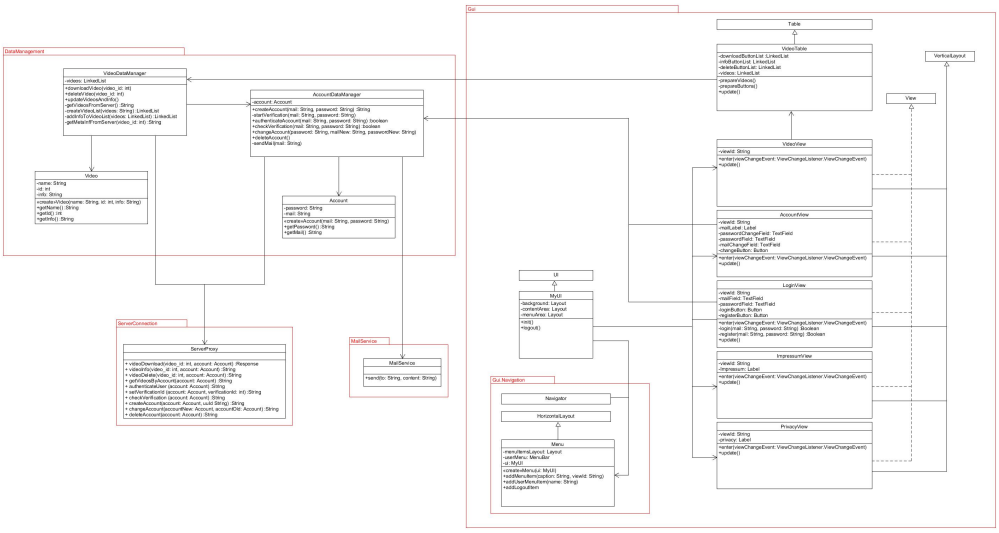

Abbildung 4.1: UML Diagramm des Web-Interface

4.1.1 Entwurfsmuster

Im Web-Interface wurden Entwurfsmuster verwendet um austauschbare Klassen zu erhalten. Zudem helfen sie dabei das Hinzufügen neuer Funktionalität möglichst einfach zu gestalten, was bei der Umsetzung von Wunschkriterien durchaus von Bedeutung sein wird.

Proxy

Da die Datenmanager Daten vom Web-Dienst holen müssen, wurde hier ein Proxy, der ServerProxy, zwischengeschalten um den Zugriff zu regeln. Des Weiteren werden so die Klassen des DataManagement Moduls vom Web-Dienst entkoppelt.

Zustandsmuster

Der Navigator verwendet ein Zustandsmuster um die Views anzuzeigen. Die verschiedenen Views entsprechen dabei den States.

Schablonenmethode

Die Klasse VideoDataManager verwendet eine Schablonenmethode zum erzeugen der Video Liste. In dieser werden die Videos und die Informationen vom Server geholt und dann entsprechend verarbeitet.

4.2 Modulübersicht

Die Komponente Web-Interface bietet dem Benutzer eine Schnittstelle zu unserem Web-Dienst, über die er Videos herunterladen und seinen Account verwalten kann.

4.2.1 Gui

Die graphische Oberfläche besteht aus verschiedenen Views, wobei beim Start immer die LoginView angezeigt wird.

4.2.2 Gui.Navigation

Die Navigation wird durch ein Menü und einen Navigator realisiert. Das Menü bietet dem Benutzer die Möglichkeit, die View zu wechseln. Den eigentlichen Wechsel übernimmt dann der Navigator.

4.2.3 DataManagement

Die beiden Manager (VideoDataManager, AccountDataManager siehe Klassenübersicht) bereiten Daten auf und kommunizieren über den ServerProxy mit dem Web-Dienst.

4.2.4 ServerConnection

Der ServerProxy übernimmt alle Zugriffe auf den Web-Dienst.

4.2.5 MailService

Eine Klasse, die die Aufgabe hat, die Verifizierungsmail zu senden.

4.3 Klassenübersicht

4.3.1 Gui

MyUI

extends UI
Die UI Klasse bildet das Herzstück des Web-Interface. Die init Methode dieser Klasse wird beim Öffnen des Web-Interface aufgerufen. Diese Klasse hat die Aufgabe, alle Komponenten zu initialisieren und bildet dazu die Grundlage für alle graphischen Einheiten.

Attribute

- **background: VerticalLayout**
  Sichtbarkeit private

  Der Background ist die Grundlage der graphischen Oberfläche des Web-Interface. Auf dem Background werden entweder menuArea und contentArea gelegt, oder beim Login die LoginView. Dieses Attribut wird in der init() Methode erzeugt und initialisiert.
- **menuArea: VerticalLayout**
  Sichtbarkeit private

  Die menuArea ist die Grundlage für das Menu. Dieses Attribut wird in der init() Methode erzeugt und initialisiert.
- **contentArea: VerticalLayout**
  Sichtbarkeit private

  Die contentArea ist die Grundlage für die Ansichten. Dieses Attribut wird in der init() Methode erzeugt und initialisiert.
- **menu: Menu**
  Sichtbarkeit private

  Das Menu bildet die Steuereinheit für den Benutzer. Der Navigator wird in der init() Methode erzeugt und initialisiert.
- **navigator: Navigator**
  Sichtbarkeit private

  Der Navigator hat die Aufgabe, die verschiedenen Ansichten in die contentArea zu laden. Der Navigator wird in der init() Methode erzeugt und initialisiert.

Methoden

- **init (request: VaadinRequest): void**
  Sichtbarkeit protected

In dieser Methode wird zuerst die Grundlage für die graphische Oberfläche erzeugt. Dann werden alle graphischen Komponenten, der Navigator und das Menu erzeugt und initialisiert. Am Schluss wird noch die LoginView angezeigt.

- **logout (): void**
  Sichtbarkeit public

  Diese Methode löscht die Account Daten aus dem AccountDataManager und zeigt die LoginView an.

LoginView

extends VerticalLayout
implements View
Diese Klasse erbt von einem Layout, da sie selbst als graphische Komponente verwendet wird. Die LoginView wird bei jedem Start des Web-Interface oder nach einem Logout angezeigt. Nach erfolgreichem Login leitet sie diese Information an eine übergeordnete Komponente weiter.

Attribute

- **viewId: String**
  Sichtbarkeit private

  Die viewId gibt der View eine einzigartige ID über die der Navigator die View identifizieren kann.
- **mailField: TextField**
  Sichtbarkeit private

  Ein einfaches Eingabefeld zur Eingabe der Mail-Adresse.
- **passwordField: TextField**
  Sichtbarkeit private

  Ein einfaches Eingabefeld zur Eingabe des Passwortes.
- **loginButton: Button**
  Dieser Button sendet Mail-Adresse und Passwort an den AccountDataManager zum verifizieren.
- **registerButton: Button**
  Sichtbarkeit private

  Dieser Button sendet Mail-Adresse und Passwort an den AccountDataManager zum erzeugen eines neuen Accounts.

Konstruktoren

- **LoginView(ui: MyUI)**
  Sichtbarkeit public

  Durch die Referenz auf die UI wird nach erfolgreichem Login das Menu und die VideoView geladen.

Methoden

- **enter (viewChangeEvent: ViewChangeListener.ViewChangeEvent): void**
  Sichtbarkeit public

  Diese Methode wird immer bei Eintreten der View aufgerufen.

- **update (): void**
  Sichtbarkeit public

  Diese Methode wird zum Aktualisieren der View verwendet.

- **login (mail: String, password: String): Boolean**
  Sichtbarkeit private

  Diese Methode wird vom loginButton aufgerufen. Sie sendet zur Überprüfung Mail und Passwort and den AccountDataManager. Bei Erfolg wird true zurückgegeben.

- **register (mail: String, password: String): Boolean**
  Sichtbarkeit private

  Diese Methode wird vom registerButton aufgerufen. Sie sendet Mail und Passwort and den AccountDataManager zur Erstellung eines Accounts. Bei Erfolg wird true zurückgegeben.

VideoView

extends VerticalLayout
implements View
Diese Klasse erbt von einem Layout, da sie selbst als graphische Komponente verwendet wird. Diese View dient zum Anzeigen der Videos, die ein Benutzer mit seiner App hochgeladen hat. Zur Anzeige selbst lädt diese Klasse einen VideoTable.

Attribute

- **viewId: String**
  Sichtbarkeit private

  Die viewId gibt der View eine einzigartige ID, mit welcher der Navigator die View identifizieren kann.

Konstruktoren

- **VideoView()**
  Sichtbarkeit public

  Standardkonstruktor

Methoden

- **enter (viewChangeEvent: ViewChangeListener.ViewChangeEvent) :void**
  Sichtbarkeit public

  Diese Methode wird immer bei Eintreten der View aufgerufen.
- **update () :void**
  Sichtbarkeit public

  Diese Methode wird zum Aktualisieren der View verwendet.

AccountView

extends VerticalLayout
implements View
Diese Klasse erbt von einem Layout, da sie selbst als graphische Komponente verwendet wird. Diese View dient zur Anzeige der aktuellen Accountdaten und zum Durchführen von Änderungen an diesen.

Attribute

- **viewId: String**
  Sichtbarkeit private

  Die viewId gibt der View eine einzigartige ID, über die der Navigator die View identifizieren kann.
- **mailLabel: Label**
  Sichtbarkeit private

  Dieses Label dient zur Anzeige der derzeit gültigen Mail-Adresse des aktuell eingeloggten Accounts.
- **passwordChangeField: TextField**
  Sichtbarkeit private

  In diesem Eingabefeld kann bei Wunsch zur Änderung des Passwortes ein neues Passwort eingegeben werden.
- **passwordField: TextField**
  Sichtbarkeit private

  Für alle gewünschten Änderungen muss hier das aktuell aktive Passwort eingegeben werden.
- **mailChangeField: TextField**
  Sichtbarkeit private

  In diesem Eingabefeld kann bei Wunsch zur Änderung der Mail-Adresse eine neue eingegeben werden
- **changeButton: Button**
  Sichtbarkeit private

  Durch Drücken dieses Buttons werden die Änderungsdaten an den AccountDataManager zur Bearbeitung geschickt.

Konstruktoren

- **AccountView()**
  Sichtbarkeit public

  Standardkonstruktor

Methoden

- **enter (viewChangeEvent: ViewChangeListener.ViewChangeEvent): void**
  Sichtbarkeit public

  Diese Methode wird immer bei Eintreten der View aufgerufen.

- **update (): void**
  Sichtbarkeit public

  Diese Methode wird zum Aktualisieren der View verwendet.

ImpressumView

extends VerticalLayout
implements View
Diese Klasse erbt von einem Layout, da sie selbst als graphische Komponente verwendet wird. Diese View hat nur die Aufgabe, das Impressum anzuzeigen.

Attribute

- **viewId: String**
  Sichtbarkeit private

  Die viewId gibt der View eine einzigartige ID über die der Navigator die View identifizieren kann.
- **impressum: Label**
  Sichtbarkeit private

  Dieses Label wird verwendet um das Impressum anzuzeigen.

Konstruktoren

- **ImpressumView()**
  Sichtbarkeit public

  Standardkonstruktor

Methoden

- **enter (viewChangeEvent: ViewChangeListener.ViewChangeEvent): void**
  Sichtbarkeit public

  Diese Methode wird immer bei Eintreten der View aufgerufen.
- **update (): void**
  Sichtbarkeit public

  Diese Methode wird zum Aktualisieren der View verwendet.

PrivacyView

extends VerticalLayout
implements View
Diese Klasse erbt von einem Layout, da sie selbst als graphische Komponente verwendet wird. Diese View hat nur die Aufgabe, die Datenschutzinformationen anzuzeigen.

Attribute

- **viewId: String**
  Sichtbarkeit private

  Die viewId gibt der View eine einzigartige ID über die der Navigator die View identifizieren kann.
- **privacy: Label**
  Sichtbarkeit private

  Dieses Label wird verwendet, um die Datenschutzinformationen anzuzeigen.

Konstruktoren

- **PrivacyView()**
  Sichtbarkeit public

  Standardkonstruktor

Methoden

- **enter (viewChangeEvent: ViewChangeListener.ViewChangeEvent): void**
  Sichtbarkeit public

  Diese Methode wird immer beim Eintreten der View aufgerufen.
- **update (): void**
  Sichtbarkeit public

  Diese Methode wird zum Aktualisieren der View verwendet.

VideoTable

extends Table
Jede Zeile des VideoTables beinhaltet ein Video mit den zugehörigen Buttons zum Downloaden, Löschen und Anzeigen der Infos.

Attribute

- **downloadButtonList: LinkedList**
  Sichtbarkeit private

  Eine Liste an Buttons. Zu jedem Video wird ein Button erzeugt und an die Liste gehängt.
- **infoButtonList: LinkedList**
  Sichtbarkeit private

  Eine Liste an Buttons. Zu jedem Video wird ein Button erzeugt und an die Liste gehängt.
- **deleteButtonList: LinkedList**
  Sichtbarkeit private

  Eine Liste an Buttons. Zu jedem Video wird ein Button erzeugt und an die Liste gehängt.
- **videos: LinkedList**
  Sichtbarkeit private

  Das Attribut ist eine Liste der Videos. Die Videos werden bei Erzeugen oder Updaten des Tables vom VideoDataManager geholt und dann verarbeitet.

Konstruktoren

- **VideoTable()**
  Sichtbarkeit public

  Im Konstruktor werden die Videos über den VideoDataManager geholt. Zudem werden dann für jedes Video die Buttons vorbereitet.

Methoden

- **prepareVideos (): void**
  Sichtbarkeit private

  Die Videos werden zur Anzeige vorbereitet.
- **prepareButtons (): void**
  Sichtbarkeit private

  Die Buttons werden zur Anzeige vorbereitet, d.h. es werden Name und Listener gesetzt.
- **update (): void**
  Sichtbarkeit public

  Dies Methode wird verwendet um den Table zu aktualisieren.

4.3.2 Gui.Navigation

Menu

extends HorizontalLayout
Das Menu stellt die Buttons bereit, die benötigt werden, um den Benutzer zwischen den verschiedenen Ansichten wechseln zu lassen. Dazu kommt noch der Logout Button, über den man zurück zum Login gelangt.

Attribute

- **menuItemLayout: Layout**
  Sichtbarkeit private

  In dieses Layout werden neu erstellte Menu-Items hinzugefügt.
- **userMenu: MenuBar**
  Sichtbarkeit private

  Eine Referenz auf das userMenu.
- **ui: MyUI**
  Sichtbarkeit private

  Eine Referenz auf die ui in der das Menu liegt. Dies wird verwendet um den Navigator aufzurufen und den Logout durchzuführen.

Konstruktoren

- **Menu()**
  Sichtbarkeit public

  Standardkonstruktor

Methoden

- **addMenuItem (caption: String, viewId: String): void**
  Sichtbarkeit public

  Diese Methode dient zum Einfügen neuer Menüeinträge, die einen Ansichtswechsel auslösen sollen.
- **addUserMenu (name: String): void**
  Sichtbarkeit public

  Diese Methode dient zum Einfügen eines User-Menüs. Es kann immer nur ein User-Menü geben.
- **addLogoutItem (): void**
  Sichtbarkeit public

  Diese Methode dient zum Einfügen eines Logout-Eintrages. Dieser Eintrag ruft dann den Logout der übergeordneten MyUI auf.

4.3.3 DataManagement

AccountDataManager

Die Klasse dient zur Accountdatenverwaltung und Kommunikation mit dem ServerProxy. Dazu bereitet sie Daten in beide Richtungen auf.

Attribute

- **account: Account**
  Sichtbarkeit private
  statisch

  Eine Referenz auf den Account, der in dieser Sitzung eingeloggt wurde.

Methoden

- **createAccount (mail: String, password: String): String**
  Sichtbarkeit public
  Statisch

  Eine Methode, die Eingaben der LoginView bekommt und diese dann an den ServerProxy in der jeweiligen Methode weitergibt.
- **startVerification (mail: String, password: String): void**
  Sichtbarkeit private
  Statisch

  Diese Methode wird nach dem Registrieren aufgerufen. Sie erzeugt eine UUID und sendet diese einmal in einem Link per Mail an den Benutzer und dann noch an den ServerProxy.
- **authenticteAccount (account: Account): boolean**
  Sichtbarkeit public
  Statisch

  Diese Methode wird beim Login benutzt. Sie gibt an ob Passwort und Mail-Adresse korrekt sind.
- **checkVerification (account: Account): boolean**
  Sichtbarkeit public
  Statisch

  Diese Methode wird beim Login benutzt. Sie gibt an ob ein Account verifiziert ist.
- **changeAccount (mail: String, password: String): void**
  Sichtbarkeit public
  Statisch

  Eine Methode die Eingaben von der AccountView bekommt. Anschließend wird das Passwort mit dem derzeitigen Passwort verglichen. Bei Erfolg werden die Änderungen an den ServerProxy übergeben.

- **deleteAccount (): void**
  Sichtbarkeit public
  Statisch

  Bei deleteAccount werden die derzeitigen Account-Daten an den Server-Proxy in einem delete-Befehl übergeben. Anschließend werden die lokalen Account Daten gelöscht und die Seite neu geladen.

- **sendMail (mail: String): void**
  Sichtbarkeit private
  Statisch

  Diese Funktion wird benutzt um Nutzern eine Mail zur Bestätigung nach Erstellen und Löschen eines Accounts zu senden.

Account

In dieser Klasse werden die Mail-Adresse und das Passwort eines Benutzers zu einem Account zusammengefasst.

Attribute

- **mail: String**
  Sichtbarkeit private

  Ein String zum Speichern der Mail-Adresse.
- **password: String**
  Sichtbarkeit private

  Ein String zum Speichern des Passwortes.

Konstruktoren

- **Account(String: mail, String: password)**
  Sichtbarkeit public

  Nimmt die Parameter und setzt die Attribute.

VideoDataManager

Der VideoDataManager verwaltet die Videodaten und vereinfacht den Zugriff auf den ServerProxy für Klassen, die Videodaten benötigen.

Attribute

- **videos: LinkedList**
  Sichtbarkeit private
  statisch

  Eine Liste in dem der VideoDataManager die Videos hält die er vom ServerProxy bekommt.

Methoden

- **downloadVideo (videoId: int): void**
  Sichtbarkeit public
  Statisch

  Die Methode fügt die Account-Daten hinzu und ruft danach die Methode zum downloaden am ServerProxy auf.
- **deleteVideo (videoId: int): void**
  Sichtbarkeit public
  Statisch

  Die Methode fügt die Account-Daten hinzu und ruft am ServerProxy die Methode zum Löschen eines Videos auf.
- **updateVideosAndInfo (): String**
  Sichtbarkeit public
  Statisch

  Die Methode fügt die Account Daten hinzu und ruft am ServerProxy die Methode auf, welcher die Videos zurückgibt. Anschließend wird parseVideos aufgerufen und die Videos als Attribut gespeichert.
- **getVideosFromServer (): String**
  Sichtbarkeit private
  Statisch

  Die Methode schickt eine Anfrage an den ServerProxy zum Holen der Videos.
- **createVideoList (videos: String): LinkedList**
  Sichtbarkeit private
  Statisch

  Diese Methode parst die Videos in Name und Id. Anschließend erstellt sie eine Liste aus Video Objekten.

- **addInfoToVideoList (videos: LinkedList): LinkedList**
  Sichtbarkeit private
  Statisch

  Diese Methode fügt jedem Listeneintrag die Meta-Informationen hinzu.

- **getMetaInfFromServer (videoId: int): String**
  Sichtbarkeit private
  Statisch

  Die Methode fügt die Account Daten hinzu und ruft am ServerProxy die Methode auf, die die Metadaten als String zurückgibt.

Video

In dieser Klasse werden Name, Id und die Meta-Informationen zu einem Video zusammgefasst.

Attribute

- **name: String**
  Sichtbarkeit private

  Ein String zum Speichern des Namens.
- **id: int**
  Sichtbarkeit private

  Ein Integer zum Speichern der Id.
- **info: String**
  Sichtbarkeit private

  Ein String zum Speichern der Meta-Informationen.

Konstruktoren

- **Account(String: name, int: id, String: info)**
  Sichtbarkeit public

  Nimmt die Parameter und setzt damit die Attribute.

4.3.4 ServerConnection

ServerProxy

Diese Klasse dient zur Kommunikation mit dem Web-Dienst. Sie ist die einzige Klasse, die mit dem Web-Dienst kommunizieren kann.

Methoden

- **videoDownload (videoId: int, account: Account): Response**
  Sichtbarkeit public
  Statisch

  Die Methode sendet die Anfrage zum Download eines Videos an den Web-Dienst.

- **videoInfo (videoId: int, account: Account): String**
  Sichtbarkeit public
  Statisch

  Die Methode holt die Metadaten eines Videos vom Web-Dienst.

- **videoDelete (videoId: int, account: Account): String**
  Sichtbarkeit public
  Statisch

  Die Methode sendet die Anfrage zum Löschen eines Videos an den Web-Dienst.

- **getVideosByAccount (account: Account): String**
  Sichtbarkeit public
  Statisch

  Die Methode sendet eine Anfrage an den Web-Dienst, um alle Videos des Accounts zu bekommen.

- **authenticteUser (account: Account): String**
  Sichtbarkeit public
  Statisch

  Die Methode sendet eine Anfrage zum Überprüfen der Mitgegebenen Accountdaten. Zudem wird überprüft ob der Account verifiziert ist.

- **setVerificationID (account: Account, verificationID: int): String**
  Sichtbarkeit public
  Statisch

  Die Methode sendet die zuvor erzeugte ID an den Web-Dienst, dass dieser sie mit der aus dem Link abgleichen kann.

- **checkVerification (account: Account): String**
  Sichtbarkeit public
  Statisch

Die Methode sendet die zuvor erzeugte ID an den Web-Dienst, dass dieser sie mit der aus dem Link abgleichen kann.

- **createAccount (account: Account): String**
  Sichtbarkeit public
  Statisch

  Die Methode sendet die Anfrage zum Erzeugen eines Accounts an den Web-Dienst.

- **changeAccount (accountNew: Account, accountOld: Account): String**
  Sichtbarkeit public
  Statisch

  Die Methode sendet die Anfrage zum Ändern eines Accounts an den Web-Dienst.

- **deleteAccount (account: Account): String**
  Sichtbarkeit public
  Statisch

  Die Methode sendet die Anfrage zum Löschen eines Accounts an den Web-Dienst.

4.3.5 MailService

MailService

Diese Klasse dient zum Senden der Verifikationsmail.

Methoden

- **send (String: to, String: content): void**
  Sichtbarkeit public
  Statisch

  Die Methode konfiguriert den SMTP und erzeugt eine Mail. Anschließend wird die Mail dann über den SMTP gesendet.

5 Anhang

5.1 Sequenzdiagramme

Abbildung 5.1: Anmelden in der App

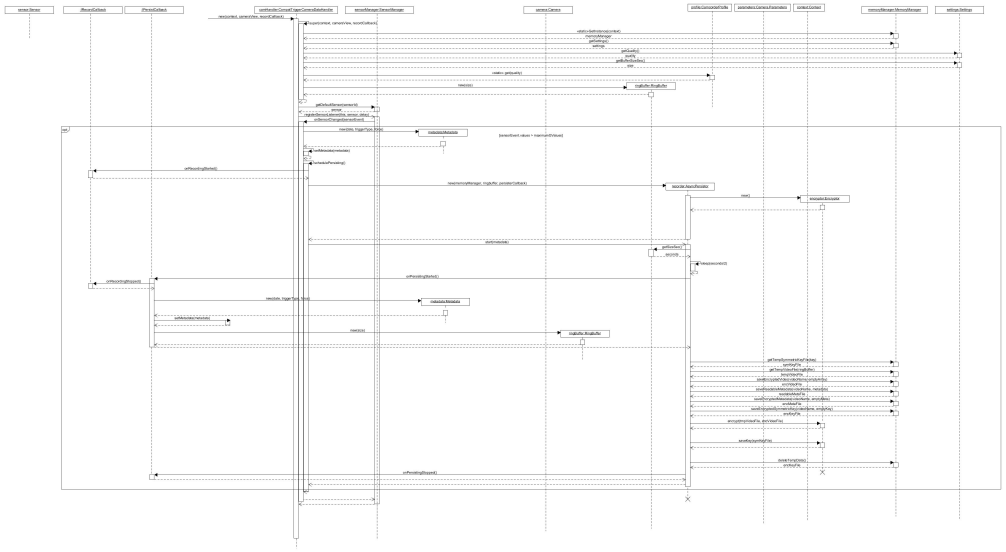

Abbildung 5.2: Videoaufnahme in der App - Initialisierung und Persistierung

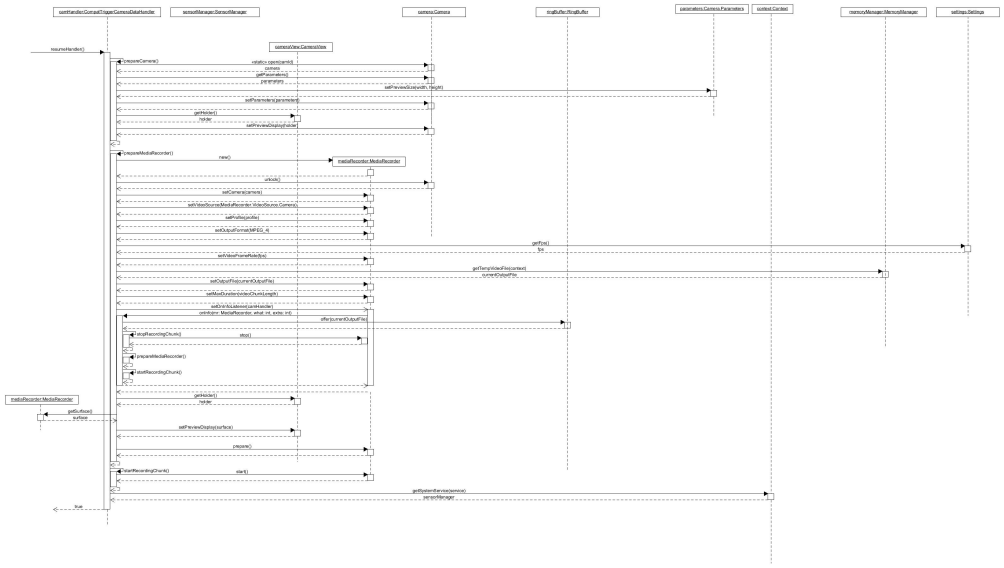

Abbildung 5.3: Videoaufnahme in der App - Beobachtung

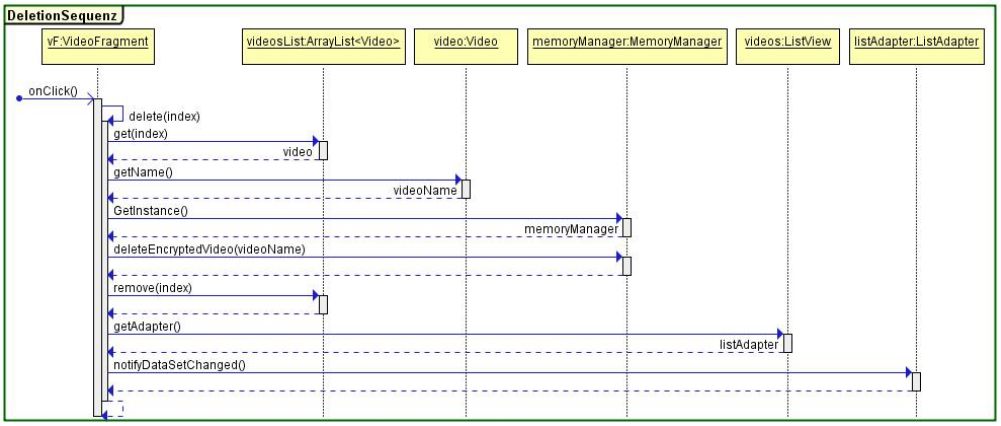

Abbildung 5.4: Video in der App löschen

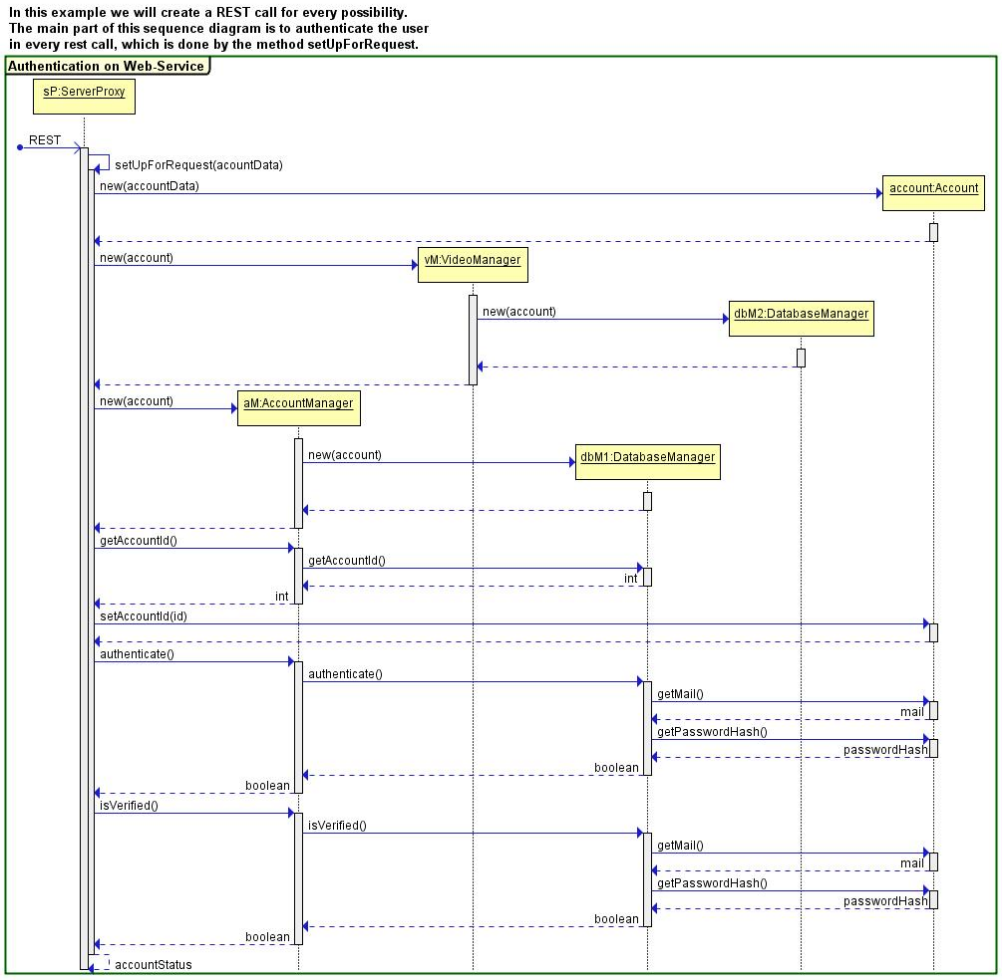

Abbildung 5.5: Authentifizieren auf dem Dienst

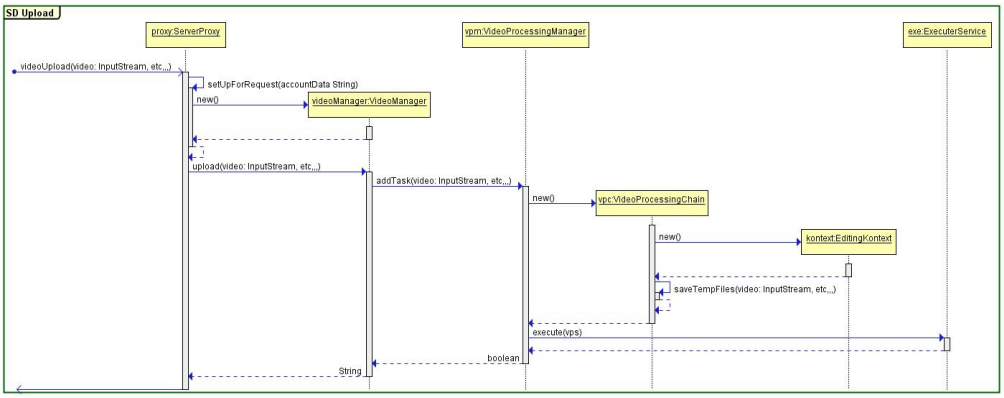

Abbildung 5.6: Video auf den Dienst hochladen

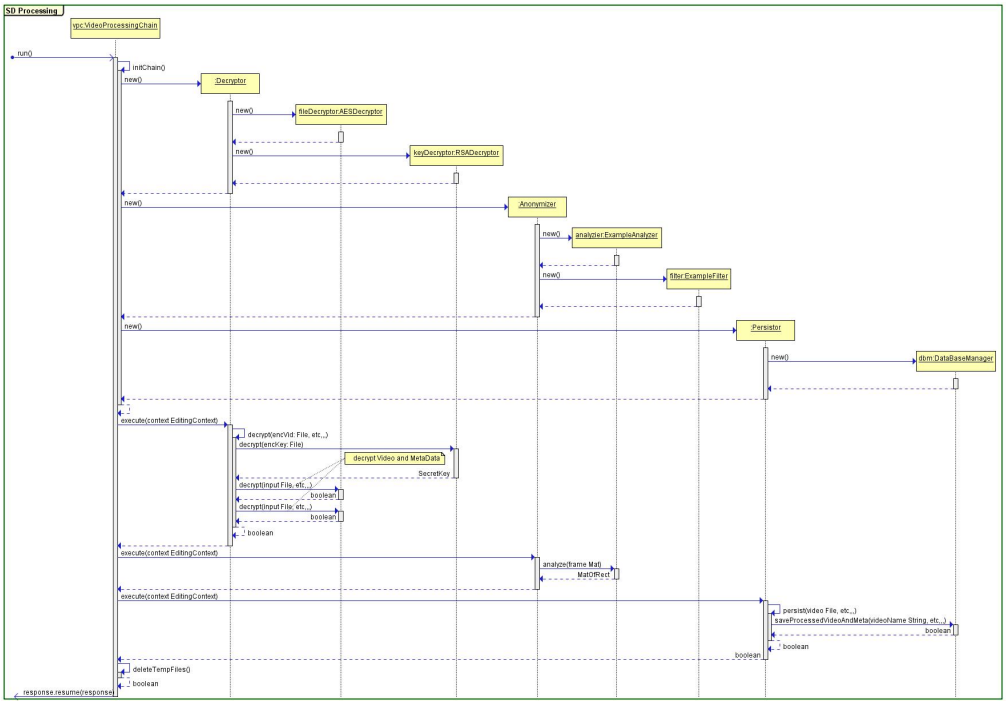

Abbildung 5.7: Videobearbeitung auf dem Dienst

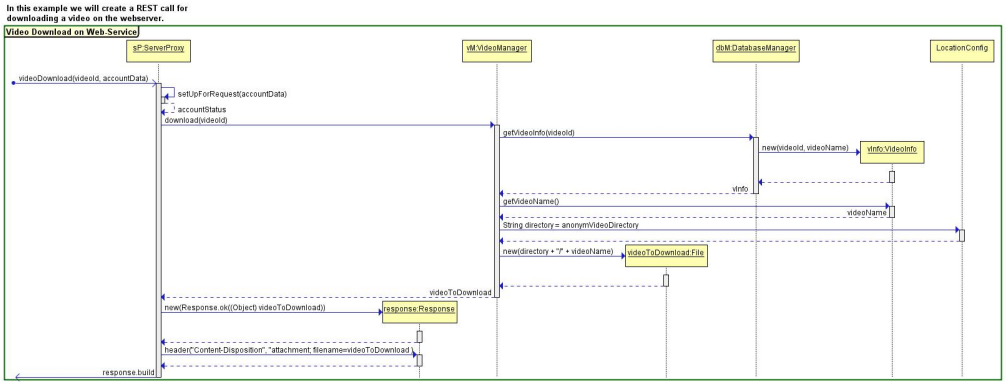

Abbildung 5.8: Videodownload vom Dienst

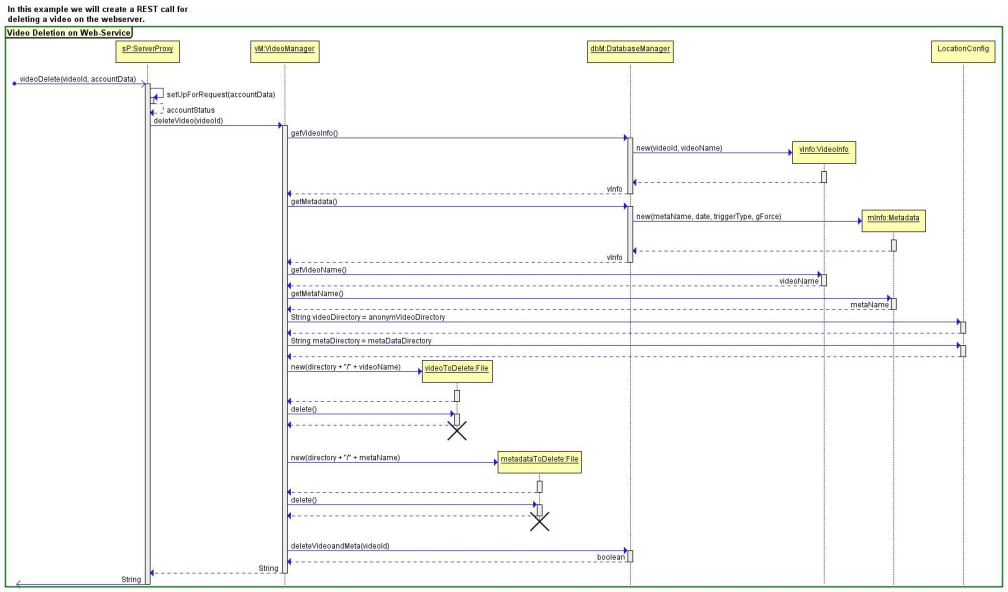

Abbildung 5.9: Video auf dem Dienst löschen