

Mensa-App

Applikation für Mitteilung und kollektiven Austausch von
Speiseplaninformationen in universitären Gastronomieeinrichtungen

PSE: Entwurf

Praxis der Softwareentwicklung
Sommersemester 2023

Alexander Albers, Peer Booken, Elena Häußler,
Alexander Kutschera, Jonatan Ziegler

23. Juni 2023

Inhaltsverzeichnis

1	Einleitung	5
1.1	Änderungen zum Pflichtenheft	6
1.2	Erklärung der Notation	7
2	Grobentwurf	8
2.1	Aufteilung in Client und Server	9
2.1.1	Verteilungsdiagramm	9
2.2	Grobentwurf Frontend	10
2.2.1	Erklärung der Komponenten	12
2.2.2	Erklärung der Schnittstellen	13
2.2.3	Paketstruktur	14
2.3	Grobentwurf Backend	15
2.3.1	Erklärung der Komponenten	16
2.3.2	Erklärung der Schnittstellen	17
2.3.3	Paketstruktur	18
3	Feinentwurf	19
3.1	Feinentwurf Frontend	20
3.1.1	Starten der App	20
3.1.2	View	21
3.1.3	Paket <code>core</code>	22
3.1.4	Paket <code>detail-view</code>	30
3.1.5	Paket <code>favorites</code>	33
3.1.6	Paket <code>filter</code>	34
3.1.7	Paket <code>images</code>	37
3.1.8	Paket <code>mealplan</code>	39
3.1.9	Paket <code>settings</code>	41
3.1.10	Paket <code>logic</code> - Interfaces zur View	43
3.1.11	Paket <code>repository</code>	50
3.1.12	Paket <code>data-types</code>	50
3.1.13	Paket <code>error-handling</code>	57
3.1.14	Paket <code>interface</code> - Interfaces zum Model	60

3.1.15	Paket <code>model</code>	62
3.2	Feinentwurf Backend	65
3.2.1	Paket <code>startup</code>	65
3.2.2	Komponente <code>Scheduler</code>	69
3.2.3	Komponente <code>GraphQL</code>	72
3.2.4	Komponente <code>Command</code>	81
3.2.5	Komponente <code>MealplanManagement</code>	86
3.2.6	Komponente <code>ImageReview</code>	90
3.2.7	Komponente <code>Database</code>	93
3.2.8	Komponente <code>SwKaParser</code>	103
3.2.9	Komponente <code>FlickrApi</code>	107
3.2.10	Komponente <code>Mail</code>	110
3.2.11	Paket <code>util</code>	113
4	Schemata	114
4.1	Datenbankschema Server	115
4.1.1	Entity-Relationship-Model	115
4.1.2	Relationenschema	115
4.1.3	Domains	116
4.1.4	Entitäten	116
4.2	Datenbankschema Client	121
4.2.1	Entity-Relationship-Model	121
4.2.2	Relationenschema	122
4.2.3	Entitäten	122
4.3	API-Schema	128
4.3.1	Authentifizierung	128
4.3.2	Typen	128
4.3.3	Querys	131
4.3.4	Mutations	132
5	Glossar	135

Abbildungsverzeichnis

2.1	Verteilungsdiagramm der Mensa-App	9
2.2	Komponenten des Frontends	11
2.3	Komponenten des Backends	15
3.1	Sequenzdiagramm der <code>main</code> -Methode	20
3.2	Navigationsdiagramm des Frontends	21
3.3	Übersicht der selbstgeschriebenen Widgets der View	22
3.4	Klassendiagramm des Paket <code>detailview</code>	30
3.5	Sequenzdiagramm zum Abgeben einer Bewertung	32
3.6	Sequenzdiagramm zum Verlinken eines Bildes	32
3.7	Klassendiagramm des Paket <code>favorites</code>	33
3.8	Klassendiagramm des Paket <code>filter</code>	34
3.9	Sequenzdiagramm zum Speichern einer Filterkonfiguration	36
3.10	Sequenzdiagramm zum Zurücksetzen einer Filterkonfiguration	36
3.11	Klassendiagramm des Paket <code>images</code>	37
3.12	Sequenzdiagramm zum Upvoten eines Bildes	38
3.13	Sequenzdiagramm zum Melden eines Bildes	38
3.14	Klassendiagramm des <code>mealplan</code> -Pakets	39
3.15	Sequenzdiagramm Mensa-Wechsel	41
3.16	Klassendiagramm des Paket <code>settings</code>	41
3.17	Sequenzdiagramm zum Wechseln zwischen Light und Dark-Mode	42
3.18	Klassendiagramm der Interfaces zwischen View und View-Model	43
3.19	Sequenzdiagramm Mensa-Wechsel	45
3.20	Sequenzdiagramm Hinzufügen eines Favoriten	46
3.21	Sequenzdiagramm Anfragen der Preiskategorie	47
3.22	Sequenzdiagramm Verlinkung eines Bildes	49
3.23	Klassendiagramm für Speiseplandaten	50
3.24	Klassendiagramm Error Handling	57
3.25	Übersicht Interfacestruktur	60
3.26	Klassendiagramm Model	63
3.27	Klassendiagramm des <code>startup</code> -Pakets	65
3.28	Sequenzdiagramm des <code>startup</code> -Pakets	68

3.29	Klassendiagramm der Scheduling-Komponente	69
3.30	Sequenzdiagramm der Scheduling-Komponente	70
3.31	Klassendiagramm der Test-Klassen der Scheduling-Komponente	71
3.32	Klassendiagramm der GraphQL-Komponente	72
3.33	Sequenzdiagramm der GraphQL-Komponente	79
3.34	Klassendiagramm der GraphQL-Test-Komponente	80
3.35	Klassendiagramm der Command-Komponente	81
3.36	Sequenzdiagramm der Command-Komponente	84
3.37	Klassendiagramm der Test-Klassen der Command-Komponente	85
3.38	Klassendiagramm der MealplanManager-Komponente	86
3.39	Sequenzdiagramm der MealplanManager-Komponente	88
3.40	Klassendiagramm der Test-Klassen der MealplanManager-Komponente	89
3.41	Klassendiagramm der ImageReview-Komponente	90
3.42	Sequenzdiagramm der ImageReview-Komponente	91
3.43	Klassendiagramm der Testklassen der ImageReview-Komponente	92
3.44	Klassendiagramm der Database-Komponente	93
3.45	Sequenzdiagramm für eine Datenbankanfrage über die RequestDataAccess- Schnittstelle der Database-Komponente	102
3.46	Klassendiagramm der SwKaParser-Komponente	103
3.47	Sequenzdiagramm einer Parse-Abfolge in der SwKaParser-Komponente	106
3.48	Klassendiagramm der FlickrApi-Komponente	107
3.49	Sequenzdiagramm der Validierung einer übergebenen URL in der FlickrApi- Komponente	109
3.50	Sequenzdiagramm der Überprüfen der Existenz eines Bildes in der FlickrApi- Komponente	109
3.51	Klassendiagramm der Mail-Komponente	110
3.52	Sequenzdiagramm der Mail-Komponente	112
3.53	Klassendiagramm des util-Pakets	113
4.1	ER-Model Datenbank Backend	115
4.2	ER-Model Datenbank Frontend	121
4.3	Schema der GraphQL-API	134

Kapitel 1

Einleitung

Wie schon im zugehörigen Pflichtenheft erklärt, ist das Ziel dieses Projekts eine App zum Anzeigen von Speiseplänen der Mensen des Studierendenwerks Karlsruhe und zum Bewerten der enthaltenen Gerichte zu erstellen.

Um die im Pflichtenheft spezifizierten Funktionalitäten zu erreichen, werden zwei Bestandteile entworfen: Eine App als Frontend und eine Serveranwendung als Backend. Inhalt dieses Entwurfsdokuments ist die genaue Beschreibung der Struktur dieser beiden Bestandteile.

Zusätzlich werden einige Schemata definiert, die die Speicherung oder den Transport von Daten beschreiben.

1.1 Änderungen zum Pflichtenheft

Einige im Pflichtenheft getroffenen Spezifikationen wurden abgewandelt oder entfernt, da diese als nicht sinnvoll oder machbar erachtet werden. Diese beinhalten:

- [M1] Kannkriterium [KK06] „Benachrichtigungen für favorisierte Gerichte“ inklusive allen seinen Funktionen ([F7XX]) wird nicht umgesetzt.
- [M2] Kannkriterium [KK07] „Favoriten und Einstellungen im- und exportieren“ inklusive allen seinen Funktionen ([F604], [F605]) wird nicht umgesetzt.
- [M3] Es wurde eine weitere Ansicht ähnlich zu [UI14] hinzugefügt, die dem Nutzer mitteilt, dass zum aktuellen Zeitpunkt noch keine Speiseplandaten für den gewählten Tag und die gewählte Mensa zur Verfügung stehen.
- [M4] Die Mindestanforderungen für den ordnungsgemäßen Betrieb des Servers wurden abgeschwächt: Es sind nur noch CPU-Kerne mit mindestens 1 GHz anstatt 3 GHz Taktrate vonnöten.
- [M5] Die Formel für den Bilrang, ein Maß für die Beliebtheit eines Bildes, wurde festgelegt:

$$\text{Bilrang} := (1 - \alpha)s + \alpha \in [0, 1]$$

$$s := \frac{\# \text{Aufwertungen}}{\# \text{Aufwertungen} + \# \text{Abwertungen} + 10 \cdot \# \text{Meldeanträge} \cdot (\text{Bild nicht verifiziert}) + 1}$$

$$\alpha := \max\left\{0, \frac{1}{2} - \frac{1}{60} \cdot (\text{Alter des Bildes in Tagen})\right\}$$

Dabei werden neue Bilder stärker bevorzugt und gemeldete Bilder benachteiligt.

- [M6] Die Formel für die Meldeschranke, die Grenze, ab der ein Bild bei einem Meldeantrag durch das automatische System ausgeblendet wird, wurde festgelegt:

$$\text{Bild wird ausgeblendet} : \iff t \leq 30 \wedge \# \text{Meldeanträge} > \text{floor}\left(\frac{1}{35}t^2 + 5\right)$$

$$t := \text{Alter des Bildes in Tagen}$$

1.2 Erklärung der Notation

In diesem Dokument wird stellenweise eine vom UML-Standard¹ abweichende Notation verwendet. Im Folgenden werden die wichtigsten dieser Abweichungen erläutert und sonstige erwähnenswerte Notationen erklärt:

- Gestrichelte Pfeile mit offener Spitze stellen dar, dass eine Klasse eine andere verwendet.
- Die Angabe `Typ1 | Typ2` für einen Typen bedeutet, dass das zugehörige Attribut oder der zugehörige Parameter entweder vom Typ `Typ1` oder `Typ2` ist.
- Tupel mit mehreren Werten werden mit der Typenbezeichnung (`Typ1`, `Typ2`) dargestellt.
- Ein `struct` unterscheidet sich von einer `class` dadurch, dass ersteres lediglich eine Gruppierung von Daten ohne jegliche Funktion bereitstellt. Die Attribute können einmal bei der Erzeugung übergeben, und danach direkt abgerufen werden.
- Aktoren in Sequenzdiagrammen können auch außenstehende Komponenten darstellen, die Abläufe in einer Komponente auslösen.
- Wenn nicht anders angegeben sind alle in Klassendiagrammen dargestellten Attribute und Methoden als `public` zu betrachten.
- Der Typ `Void` stellt einen Typ mit nur einer möglichen Instanz dar und markiert somit, dass z.B. „nichts“ zurückgegeben wird.
- Klassen werden in Klassendiagrammen mit einem „C“, Schnittstellen mit einem „I“, Strukturen mit einem „S“ und Enumerationstypen mit einem „E“ in einem Kreis dargestellt.
- Statische Attribute und Methoden sind unterstrichen.
- Im Frontend werden Bezeichner in `camelCase`, im Backend hingegen in `snake_case` benannt. Dies ist Ergebnis der unterschiedlichen Konventionen der verwendeten Programmiersprachen im Front- und Backend und dem Wunsch, Bezeichner möglichst implementierungsgetreu.
- Des Öfteren werden die Typen `Uuid` und `Date` verwendet, die eine `UUID`² oder ein Datum darstellen.

¹<https://www.omg.org/spec/UML/2.5.1/PDF>

²<https://datatracker.ietf.org/doc/html/rfc4122>

Kapitel 2

Grobentwurf

In diesem Kapitel wird zunächst einen Überblick über den Aufbau der Architektur gegeben und anschließend die Struktur des Front- und Backends vorgestellt. Eine detailliertere Beschreibung folgt in Kapitel 3 Feinentwurf.

2.1 Aufteilung in Client und Server

Die Mensa-App ist im Wesentlichen in eine Client-App für die Interaktion mit dem Nutzer und einen Server zur Synchronisierung und Bereitstellung der Speiseplandaten aufgeteilt. Im Folgenden soll die Kommunikation zwischen Client und Server genauer erläutert werden.

2.1.1 Verteilungsdiagramm

Das folgende Verteilungsdiagramm zeigt die Aufteilung in Client und Server sowie deren Interaktion und die Zugriffe auf weitere externe Schnittstellen.

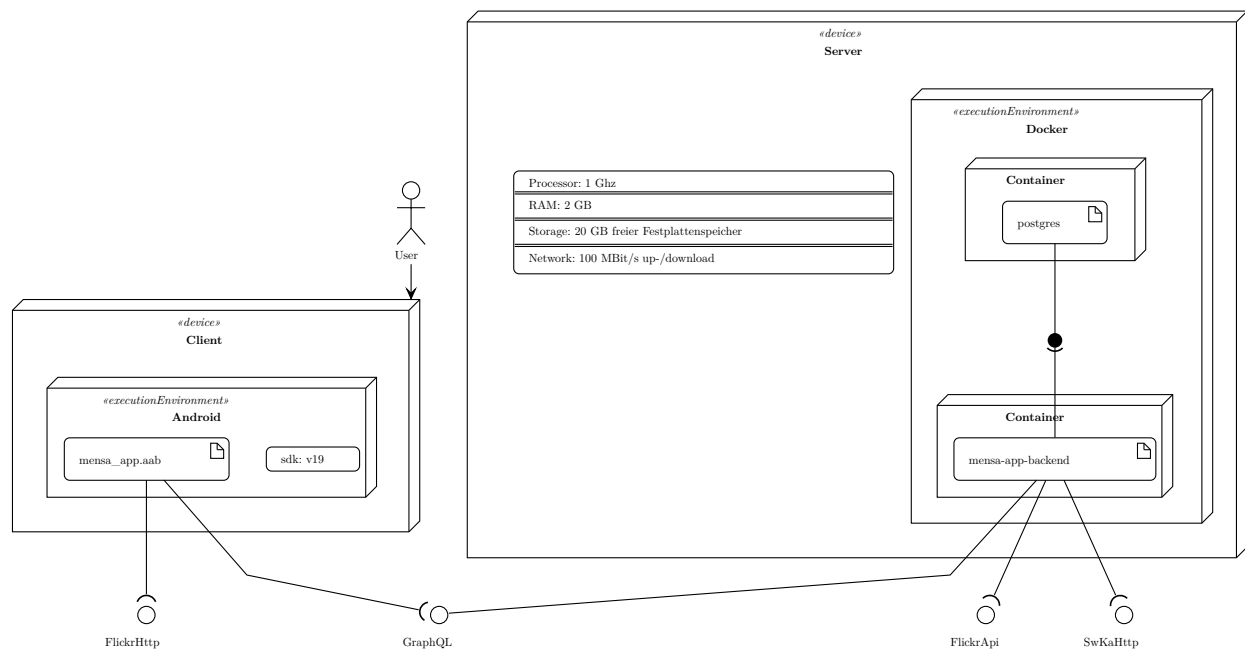


Abbildung 2.1: Verteilungsdiagramm der Mensa-App

Der Nutzer interagiert mit der App in einem Android-Betriebssystem, das auf einem Endgerät ausgeführt wird. Der Server lagert und verwaltet Anwendungen in Docker-Containern. Der Rust-Server ist gestartet und aktiv, sobald die ausführbare Datei erfolgreich in einem Container gestartet wurde. Der Server bietet in aktivem Zustand eine Kommunikationsschnittstelle nach außen an.

Basierend auf HTTP verwendet der Server eine GraphQL-API für die Kommunikation nach außen. Möchte ein Client mit dem Server kommunizieren, so muss er sich an die Kommunikation der API anpassen. Deshalb benutzt die App diese API, sodass eine erfolgreiche Kommunikation möglich ist. Der Server bietet keine andere Schnittstelle für einen Client an.

Weitere Informationen zur GraphQL-API finden sich in 4.3 API-Schema.

2.2 Grobentwurf Frontend

Für die Implementierung des Frontends wird das Model View View-Model (MVVM) verwendet. Das MVVM-Muster enthält folgende drei Komponenten:

- **View:** Hier liegen alle Komponenten, die für das Anzeigen der Benutzeroberfläche benötigt werden.
- **View-Model:** Hier befinden sich das Statemanagement, die Logik für die einzelnen Features und die Interfaces, die vom Model implementiert werden, inkl. der dazugehörigen Klassen, durch die die Daten übergeben werden.
- **Model:** Hier befinden sich Komponenten, die zur Abfrage und Bereitstellung in- und externer Daten notwendig sind.

Die folgende Grafik gibt eine Übersicht über die Komponenten in den jeweiligen Schichten:

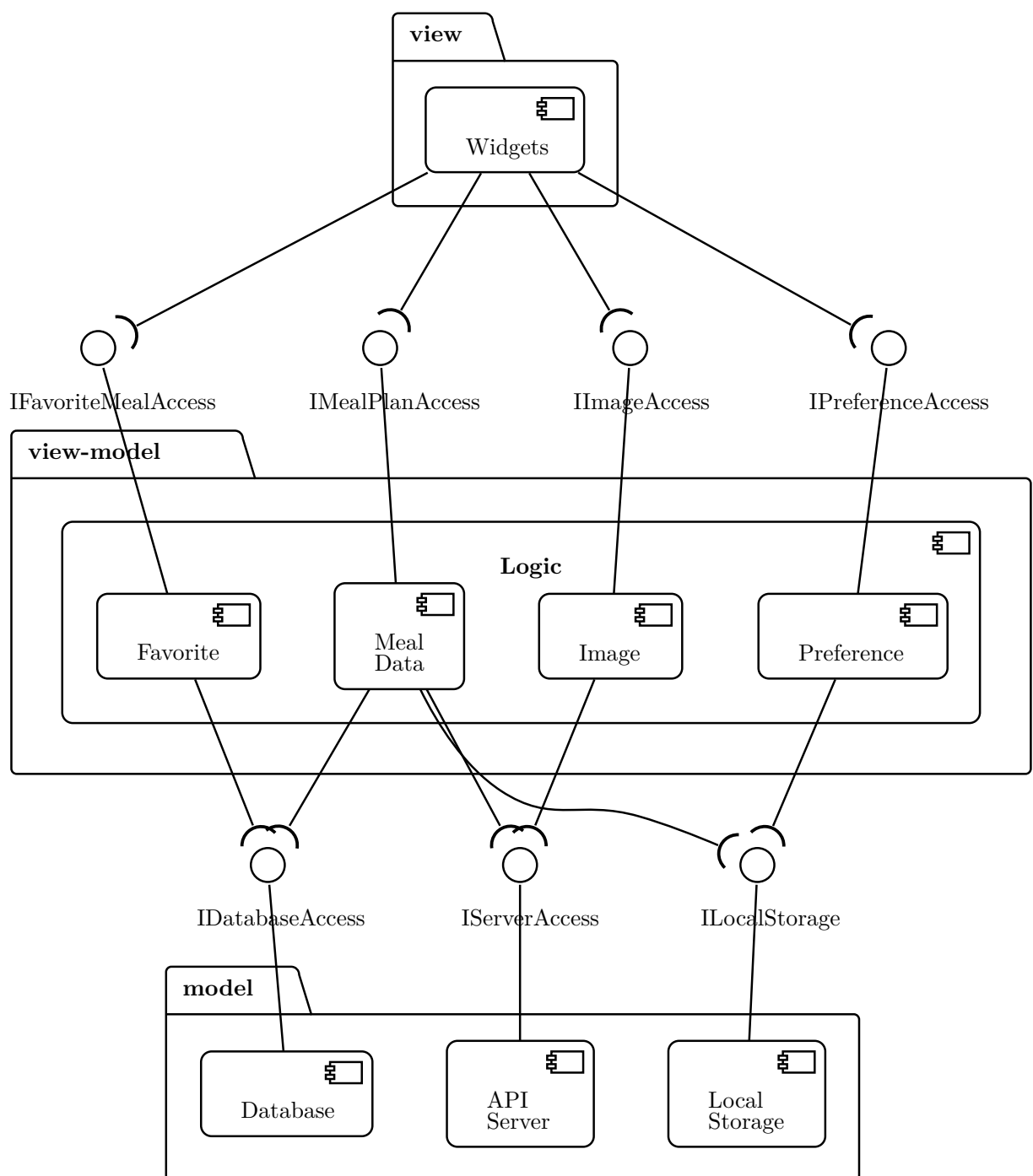


Abbildung 2.2: Komponenten des Frontends

2.2.1 Erklärung der Komponenten

Die Komponenten haben folgende Aufgabenbereiche:

Widgets

Diese Komponente enthält sämtliche Ressourcen, die für die Darstellung der Benutzeroberfläche benötigt werden. Für die genaue Beschreibung dieser Komponente siehe 3.1.3 Paket `core`.

Logic

Diese Komponente enthält die Logik, die für die Benutzeroberfläche benötigt wird. Für die genaue Beschreibung dieser Komponente siehe 3.1.10 Paket `logic` - Interfaces zur View.

Image

Diese Komponente enthält die Logik, die für die Darstellung von Bildern benötigt wird. Für die genaue Beschreibung dieser Komponente siehe 3.1.7 Paket `images`.

Meal Data

Diese Komponente enthält die Logik, die für die Darstellung von Gerichten und dem Speiseplan benötigt wird. Für die genaue Beschreibung dieser Komponente siehe 3.1.10 Paket `logic` - Interfaces zur View.

Favorite

Diese Komponente enthält die Logik, die für die Darstellung von Favoriten benötigt wird. Für die genaue Beschreibung dieser Komponente siehe 3.1.5 Paket `favorites`.

Preference

Diese Komponente enthält die Logik, die für die Darstellung von Einstellungen oder den Zugriff auf andere Informationen zu den Präferenzen des Nutzers benötigt wird. Für die genaue Beschreibung dieser Komponente siehe 3.1.9 Paket `settings`.

API Server

Diese Komponente ist für die Kommunikation mit dem Server zuständig. Für die genaue Beschreibung dieser Komponente siehe 3.1.15 Paket `model`.

Database

Diese Komponente ist für Zugriffe auf die Datenbank zuständig. Für die genaue Beschreibung dieser Komponente siehe 3.1.15 Paket `model`.

Local Storage

Diese Komponente ist für Zugriff auf den lokalen Speicher zuständig. Für die genaue Beschreibung dieser Komponente siehe 3.1.15 Paket `model`.

2.2.2 Erklärung der Schnittstellen

Für die Kommunikation zwischen den Komponenten existieren die folgenden Schnittstellen. Diese werden im Folgenden genauer beschrieben.

IFavoriteMealAccess

Diese Schnittstelle erlaubt das Laden und Ändern der Favoriten des Nutzers. Für die genaue Beschreibung dieser Schnittstelle siehe 3.1.10 Paket `logic` - Interfaces zur View.

IImageAccess

Diese Schnittstelle erlaubt das Verlinken, Bewerten und Melden von Bildern. Für die genaue Beschreibung dieser Schnittstelle siehe 3.1.10 Paket `logic` - Interfaces zur View.

IMealPlanAccess

Diese Schnittstelle erlaubt das Laden des Speiseplans oder einzelner Gerichte und Änderungen an Gerichten vorzunehmen. Für die genaue Beschreibung dieser Schnittstelle siehe 3.1.10 Paket `logic` - Interfaces zur View.

IPreferenceAccess

Diese Schnittstelle erlaubt das Laden und Ändern von verschiedenen Einstellungen und Client-Informationen. Für die genaue Beschreibung dieser Schnittstelle siehe 3.1.10 Paket `logic` - Interfaces zur View.

IDatabaseAccess

Diese Schnittstelle ermöglicht das Ändern von Daten in der Datenbank und das Laden von Daten aus der Datenbank. Für die genaue Beschreibung dieser Schnittstelle siehe 3.1.14 Paket `interface` - Interfaces zum Model.

ILocalStorage

Diese Schnittstelle ermöglicht das Speichern und Laden von Daten mithilfe lokaler Speichermöglichkeiten außerhalb der Datenbank dar. Für die genaue Beschreibung dieser Schnittstelle siehe 3.1.14 Paket `interface` - Interfaces zum Model.

IServerAccess

Diese Schnittstelle ist eine interne Schnittstelle zur Kommunikation mit dem Server zum Datenaustausch. Für die genaue Beschreibung dieser Schnittstelle siehe 3.1.14 Paket `interface` - Interfaces zum Model.

2.2.3 Paketstruktur

Dabei wird folgende Paketstruktur verwendet:

```
└─ view
  └─ core
    └─ buttons
    └─ meal_view_format
    └─ selection_components
    └─ information_display
    └─ input_components
    └─ dialogs
  └─ detail-view
  └─ favorites
  └─ filter
  └─ mealplan
  └─ images
  └─ settings
└─ view-model
  └─ logic
    └─ favorite
    └─ image
    └─ meal
    └─ preference
  └─ repository
    └─ data-classes
      └─ filter
      └─ meal
      └─ mealplan
      └─ settings
    └─ error-handling
    └─ interface
└─ model
  └─ api-server
  └─ database
  └─ local-storage
```

2.3 Grobentwurf Backend

Für die Implementierung des Backends wird ein transparentes Drei-Schichten-Modell verwendet. Jede Komponente ist einer der drei folgenden Schichten zugeordnet:

- **Trigger:** Hier werden alle zu behandelnden Ereignisse wie API-Anfragen ausgelöst.
- **Logic:** Hier befindet sich jegliche anwendungsspezifische Logik. Diese ist dabei komplett von den Implementierungen der anderen Schichten unabhängig und interagiert selbst nicht mit der Außenwelt.
- **Data:** Hier finden alle Interaktionen mit der Außenwelt und die dazu benötigten Konvertierungen von Daten statt.

An den Schichtgrenzen werden Schnittstellen definiert, über die Komponenten schichtübergreifend zusammenarbeiten können.

Die folgende Grafik gibt einen Überblick über die Komponenten in den jeweiligen Schichten und deren Schnittstellen:

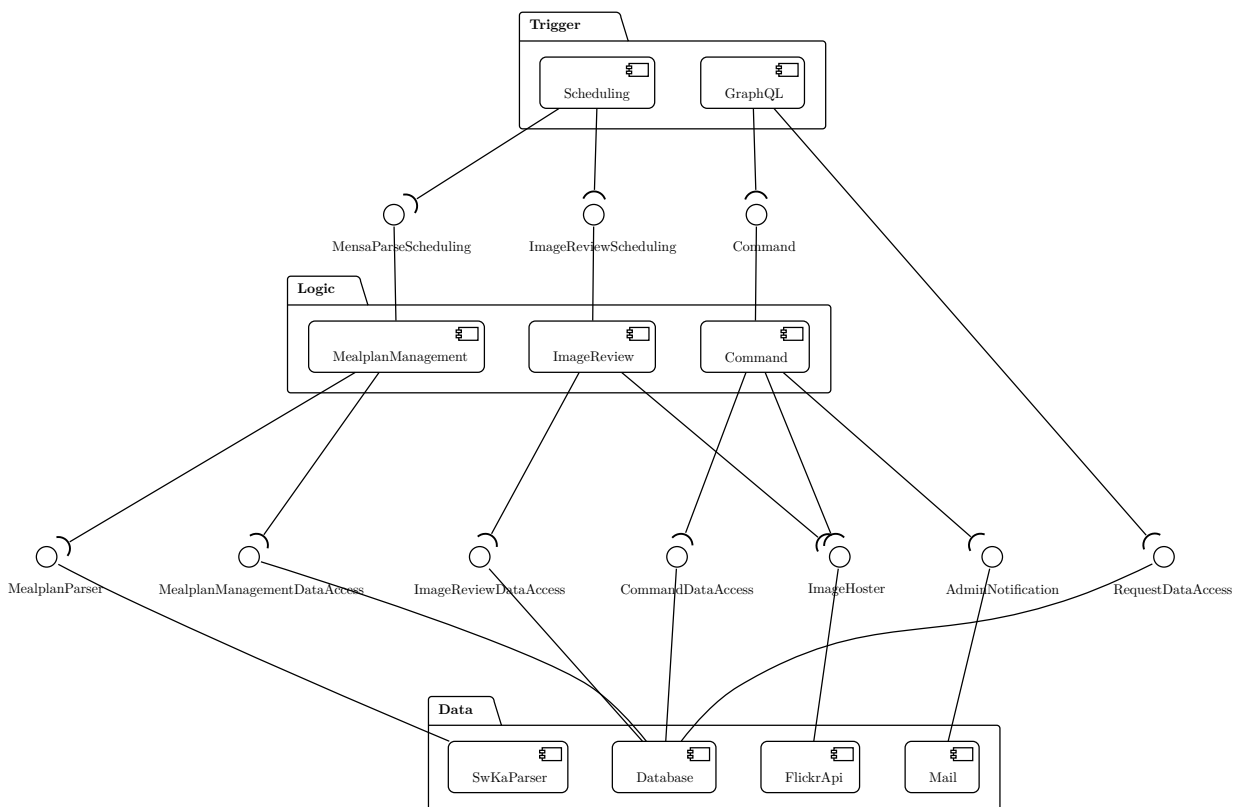


Abbildung 2.3: Komponenten des Backends

2.3.1 Erklärung der Komponenten

Die Komponenten haben folgende Aufgabenbereiche:

Scheduling

Diese Komponente löst regelmäßige Ereignisse wie z.B. das Abfragen des aktuellen Speiseplans aus. Für die genaue Beschreibung dieser Komponente siehe 3.2.2 Komponente **Scheduler**.

GraphQL

Diese Komponente enthält den Webserver, der API-Anfragen ermöglicht und den Einstiegspunkt für diese darstellt. Für die genaue Beschreibung dieser Komponente siehe 3.2.3 Komponente **GraphQL**.

Command

Diese Komponente enthält all die Logik, die bei API-Anfragen benötigt wird, die mehr als nur Daten abfragen. Für die genaue Beschreibung dieser Komponente siehe 3.2.4 Komponente **Command**.

MealplanManagement

Diese Komponente steuert die Aktualisierung der Speisepläne und transformiert die erhaltenen Speiseplandaten in ein für die Datenbank kompatibles Format. Für die genaue Beschreibung dieser Komponente siehe 3.2.5 Komponente **MealplanManagement**.

ImageReview

Diese Komponente prüft, ob verlinkte Bilder immer noch beim Bildhoster vorhanden sind, und entfernt sie gegebenenfalls aus der Datenbank. Für die genaue Beschreibung dieser Komponente siehe 3.2.6 Komponente **ImageReview**.

Database

Diese Komponente ist für Zugriffe auf die Datenbank zuständig. Für die genaue Beschreibung dieser Komponente siehe 3.2.7 Komponente **Database**.

SwKaParser

Diese Komponente ist für die Abfrage der aktuellen Speisepläne von den Webseiten der Mensen zuständig. Für die genaue Beschreibung dieser Komponente siehe 3.2.8 Komponente **SwKaParser**.

FlickrApi

Diese Komponente ist für die Kommunikation mit dem Bildhoster „Flickr“ zuständig. Für die genaue Beschreibung dieser Komponente siehe 3.2.9 Komponente **FlickrApi**.

Mail

Diese Komponente ist für das Senden von E-Mails an den Administrator zuständig. Für die genaue Beschreibung dieser Komponente siehe 3.2.10 Komponente **Mail**.

2.3.2 Erklärung der Schnittstellen

Für die Kommunikation zwischen den Komponenten existieren die folgenden Schnittstellen. Diese werden bei der implementierenden Komponente genauer beschreiben.

MensaParseScheduling

Diese Schnittstelle erlaubt das Starten der Vorgänge zum Aktualisieren des Speiseplans von der Mensa-Webseite. Für die genaue Beschreibung dieser Schnittstelle siehe 3.2.5 Komponente `MealplanManagement`.

ImageReviewScheduling

Diese Schnittstelle erlaubt das Starten des Vorgangs zum Überprüfen der Existenz der zu Gerichten verlinkten Bilder. Für die genaue Beschreibung dieser Schnittstelle siehe 3.2.6 Komponente `ImageReview`.

Command

Diese Schnittstelle erlaubt das Ausführen von API-Befehlen. Für die genaue Beschreibung dieser Schnittstelle siehe 3.2.4 Komponente `Command`.

MealplanParser

Diese Schnittstelle erlaubt das Auslesen von Speiseplänen aus einer externen Quelle. Für die genaue Beschreibung dieser Schnittstelle siehe 3.2.8 Komponente `SwKaParser`.

AdminNotification

Diese Schnittstelle erlaubt die Benachrichtigung von Administratoren bei Meldeanträgen. Für die genaue Beschreibung dieser Schnittstelle siehe 3.2.10 Komponente `Mail`.

ImageHoster

Diese Schnittstelle erlaubt die Kommunikation mit dem Bildhoster. Für die genaue Beschreibung dieser Schnittstelle siehe 3.2.9 Komponente `FlickrApi`.

RequestDataAccess

Diese Schnittstelle erlaubt die Abfrage von Speiseplandaten aus dem Datenspeicher für API-Anfragen. Für die genaue Beschreibung dieser Schnittstelle siehe 3.2.7 Komponente `Database`.

CommandDataAccess

Diese Schnittstelle erlaubt die Abfrage und Manipulation von Daten aus dem Datenspeicher, die für API-Befehle nötig sind. Für die genaue Beschreibung dieser Schnittstelle siehe 3.2.7 Komponente `Database`.

ImageReviewDataAccess

Diese Schnittstelle erlaubt die Abfrage und Manipulation von Daten aus dem Datenspeicher, die für das Überprüfen der zu Gerichten verlinkten Bilder nötig sind. Für die genaue Beschreibung dieser Schnittstelle siehe 3.2.7 Komponente `Database`.

MealplanManagementDataAccess

Diese Schnittstelle erlaubt die Abfrage und Manipulation von Daten aus dem Datenspeicher, die für das Aktualisieren der Speisepläne nötig sind. Für die genaue Beschreibung dieser Schnittstelle siehe 3.2.7 Komponente Database.

2.3.3 Paketstruktur

Dabei wird folgende Paketstruktur verwendet (Test-Pakete werden nicht dargestellt):

```
└─ util
└─ startup
    └─ config
    └─ logging
└─ layer
    └─ trigger
        └─ graphql
        └─ scheduling
    └─ logic
        └─ api_command
            └─ auth
        └─ image_review
        └─ mealplan_management
    └─ data
        └─ mail
        └─ flickr_api
        └─ database
        └─ swka_parser
└─ interface
    └─ mealplan_management
    └─ image_review
    └─ api_command
    └─ persistent_data
        └─ model
    └─ admin_notification
    └─ image_hoster
    └─ mensa_parser
```

Kapitel 3

Feinentwurf

In diesem Kapitel wird die zuvor definierte Struktur für das Front- und Backend weiter ausgeführt, indem die einzelnen Klassen und Schnittstellen der Komponenten detailliert beschreiben werden. Zusätzlich werden beispielhaft einige typische Abläufe in Sequenzdiagrammen dargestellt.

3.1 Feinentwurf Frontend

Im Folgenden wird das Frontend und dessen Komponenten genauer beschrieben.

3.1.1 Starten der App

Durch die `main`-Methode wird die App gestartet. Dabei wird je eine Instanz für Klassen erstellt, die im Model liegen. Anschließend werden durch ein `MultiProvider` die benötigten `ChangeNotifier` erstellt, die Instanzen der Klassen des Models entgegen nehmen.

Anschließend wird `runApp` aufgerufen, die die grafische Benutzeroberfläche der App baut.

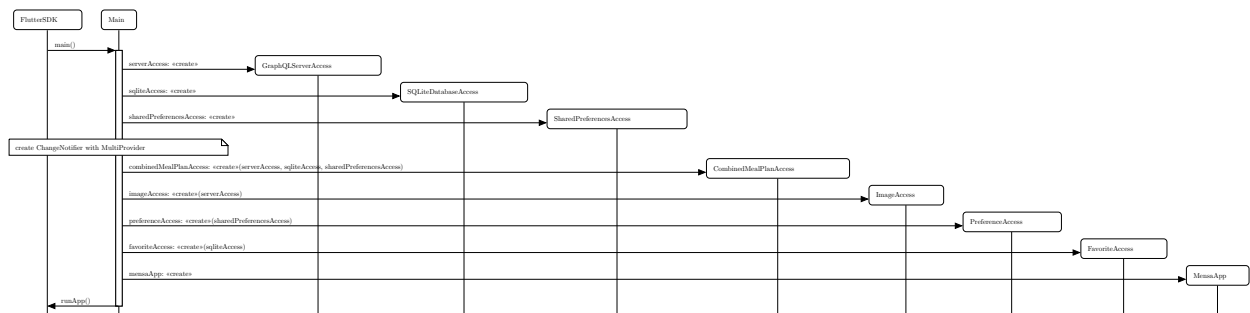


Abbildung 3.1: Sequenzdiagramm der `main`-Methode

3.1.2 View

Zuerst wird ausführlicher auf die View eingegangen, die für das Anzeigen der Benutzeroberfläche zuständig ist. Dabei verdeutlicht das folgende Diagramm, wie durch die App navigiert werden kann.

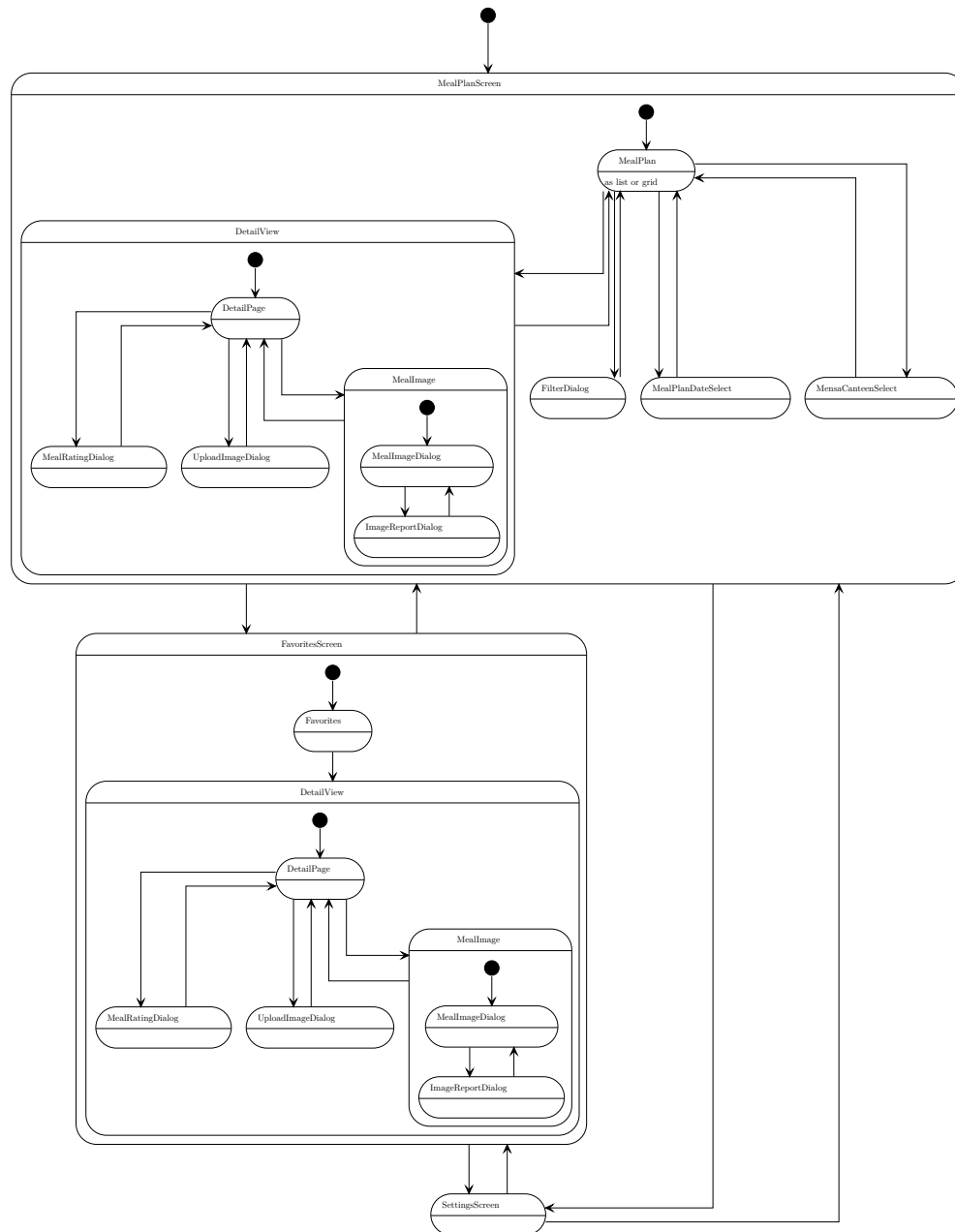


Abbildung 3.2: Navigationsdiagramm des Frontends

Insgesamt werden einige Widgets benutzt, die ein einheitliches Design oder ähnliches haben sollen und deshalb explizit hier erwähnt werden und Teil des Entwurfs sind. Im folgenden Diagramm sind diese Widgets und ihre Zugriffe aufeinander verdeutlicht. Dabei werden keine von Flutter zur Verfügung gestellte Widgets, die benutzt werden, modelliert.

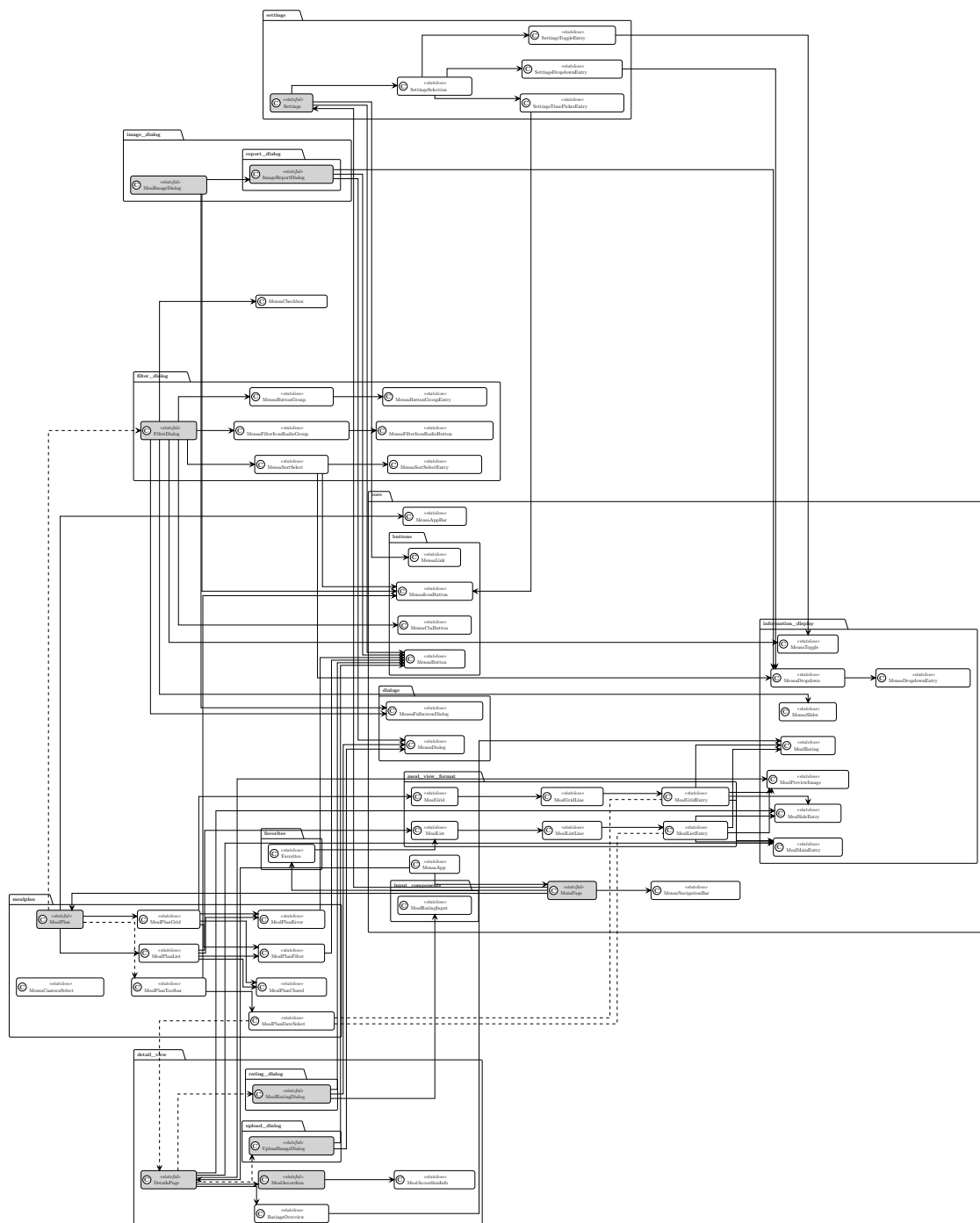


Abbildung 3.3: Übersicht der selbstgeschriebenen Widgets der View

Alle diese Widgets erben entweder von `StatelessWidget` oder von `StatefulWidget`, die eine private State-Klasse haben.

3.1.3 Paket core

Im Folgenden werden verschiedene Widgets beschrieben, die von unterschiedlichen Screens benötigt werden und deshalb zentral im `core`-Paket zur Verfügung gestellt werden. Dabei werden Nachrich-

ten von unterschiedlichen Sprachen und unterschiedlichen Farbschemata ebenfalls in Dateien in der View abgespeichert. Im Folgenden sind zu einigen Widgets zur Verdeutlichung Bilder von der Benutzeroberfläche angegeben. Diese sind hier nur im Lightmode angezeigt, obwohl sie auch im Darkmode existieren. Auf die Darstellung im Darkmode wurde im Pflichtenheft näher eingegangen.

MensaApp

Typ: class extends StatefulWidget

Paket: view/core

Beschreibung: Dieses Widget zeigt die gesamte Mensa-App an.

MainPage

Typ: class extends StatefulWidget

Paket: view/core

Beschreibung: Dieses Widget zeigt den Rahmen für die Hauptansichten an.

MensaNavigationBar

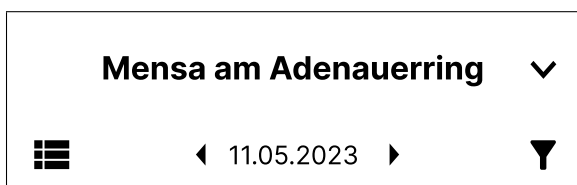


Typ: class extends StatelessWidget

Paket: view/core

Beschreibung: Dieses Widget zeigt die Menüleiste in der MainPage an.

MensaAppBar

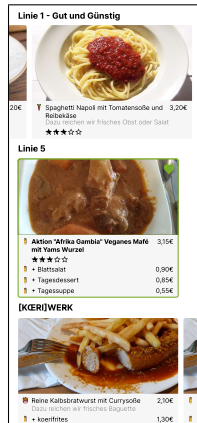


Typ: class extends StatelessWidget

Paket: view/core

Beschreibung: Dieses Widget zeigt die obere App-Leiste mit der aktuell gewählten Mensa an.

MealGrid



Typ: class extends StatelessWidget

Paket: view/core/meal-view-format

Beschreibung: Dieses Widget zeigt Gerichte im Galerie-Modus an.

MealGridLine



Typ: class extends StatelessWidget

Paket: view/core/meal-view-format

Beschreibung: Dieses Widget zeigt eine Mensa-Linie mit ihren Gerichten des MealGrid an.

MealGridEntry

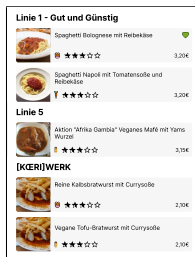


Typ: class extends StatelessWidget

Paket: view/core/meal-view-format

Beschreibung: Dieses Widget zeigt ein Gericht in einer MealGridLine an.

MealList

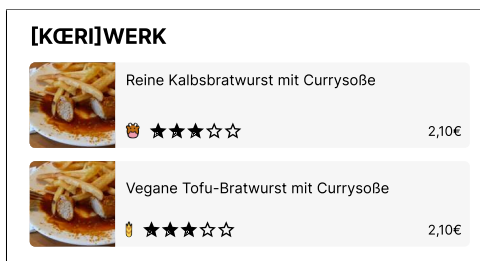


Typ: class extends StatelessWidget

Paket: view/core/meal-view-format

Beschreibung: Dieses Widget zeigt Gerichte im Listen-Modus an.

MealListLine

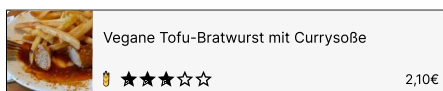


Typ: class extends StatelessWidget

Paket: view/core/meal-view-format

Beschreibung: Dieses Widget zeigt eine Mensa-Linie mit ihren Gerichten der MealList an.

MealListEntry



Typ: class extends StatelessWidget

Paket: view/core/meal-view-format

Beschreibung: Dieses Widget zeigt ein Gericht in einer MealListLine an.

MealPreviewImage

Typ: class extends StatelessWidget

Paket: view/core/information-display

Beschreibung: Dieses Widget zeigt ein Vorschaubild eines Gerichts für MealGridEntry oder MealListEntry an.

MealRating




Typ: class extends StatelessWidget

Paket: view/core/information-display

Beschreibung: Dieses Widget zeigt eine Sterne-Bewertung eines Gerichts an.

MealMainEntry


 Aktion "Afrika Gambia" Veganes Mafé mit Yams Wurzel	3,15€
---------------------------------------------------------------------------------------------------------------------------------------	-------

Typ: class extends StatelessWidget

Paket: view/core/information-display

Beschreibung: Dieses Widget zeigt eine Beschreibung eines Gerichts inklusive Name, Gerichtstyp und Preis an.

MealSideEntry

 + Tagessuppe	0,55€
-------------------------------------------------------------------------------------------------	-------

Typ: class extends StatelessWidget

Paket: view/core/information-display

Beschreibung: Dieses Widget zeigt eine Beschreibung einer Beilage inklusive Name, Gerichtstyp und Preis an.

MensaRatingInput



Typ: class extends StatelessWidget

Paket: view/core/input-components

Beschreibung: Dieses Widget zeigt Schaltflächen zum Wählen einer Sternebewertung an.

MensaCheckbox

☐ Scheinefleisch

Typ: class extends StatelessWidget

Paket: view/core/selection-components

Beschreibung: Dieses Widget zeigt eine Checkbox an.

MensaSlider



Typ: class extends StatelessWidget

Paket: `view/core/selection-components`

Beschreibung: Dieses Widget zeigt einen Slider an.

MensaToggle

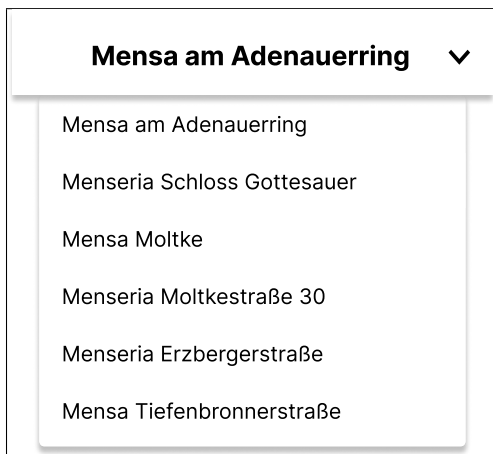


Typ: `class` extends `StatelessWidget`

Paket: `view/core/selection-components`

Beschreibung: Dieses Widget zeigt einen Schalter an.

MensaDropdown



Typ: `class` extends `StatelessWidget`

Paket: `view/core/selection-components`

Beschreibung: Dieses Widget zeigt eine Dropdown-Liste an.

MensaDropdownEntry



Typ: `class` extends `StatelessWidget`

Paket: `view/core/selection-components`

Beschreibung: Dieses Widget zeigt einen Eintrag einer Dropdown-Liste an.

MensalIconButton



Typ: `class` extends `StatelessWidget`

Paket: `view/core/buttons`

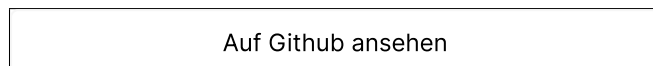
Beschreibung: Dieses Widget zeigt eine Schaltfläche mit Icon an.

MensaButton

Typ: class extends StatelessWidget

Paket: view/core/buttons

Beschreibung: Diess Widget zeigt eine Schaltfläche an.

MensaLink

Typ: class extends StatelessWidget

Paket: view/core/buttons

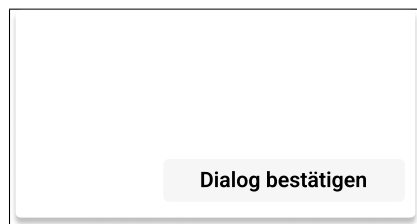
Beschreibung: Dieses Widget zeigt eine Schaltfläche mit einem Link zu einer Webseite an.

MensaCtaButton

Typ: class extends StatelessWidget

Paket: view/core/buttons

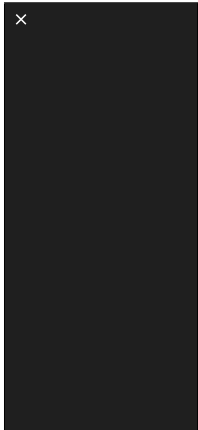
Beschreibung: Dieses Widget zeigt eine Schaltfläche für eine Aktion an.

MensaDialog

Typ: class extends StatelessWidget

Paket: view/core/dialogs

Beschreibung: Dieses Widget zeigt einen allgemeinen Dialog an, der nicht den gesamten Bildschirm einnimmt.

MensaFullscreenDialog

Typ: `class` extends `StatelessWidget`

Paket: `view/core/dialogs`

Beschreibung: Dieses Widget zeigt einen allgemeinen Dialog an, der den gesamten Bildschirm einnimmt.

3.1.4 Paket detail-view

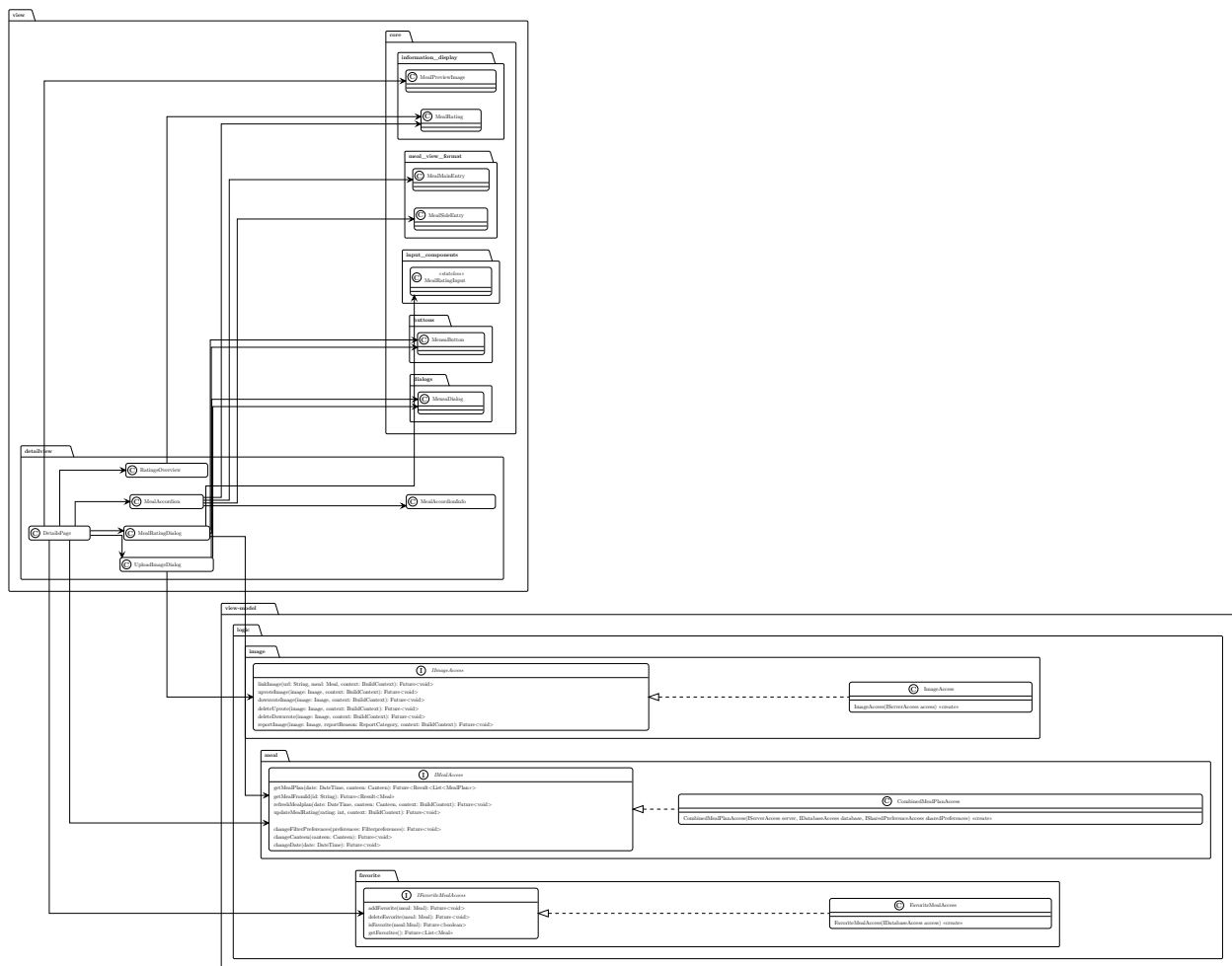


Abbildung 3.4: Klassendiagramm des Paket `detailview`

DetailsPage

Typ: class extends StatefulWidget

Paket: view/detail-view

Beschreibung: Dieses Widget zeigt die Detailansicht eines Gerichts.

UploadImageDialog

Typ: class extends StatefulWidget

Paket: view/detail-view

Beschreibung: Dieses Widget zeigt einen Dialog zum Verlinken eines Bildes.

MealAccordion

Typ: class extends StatefulWidget

Paket: view/detail-view

Beschreibung: Dieses Widget zeigt Informationen zu einem Gericht und seinen Beilagen, sowie Bezeichnung, Gerichtstyp und Preis. Dieses Widget kann ausgeklappt werden, um zusätzlich MealAccordionInfo anzuzeigen.

MealAccordionInfo

Typ: class extends StatelessWidget

Paket: view/detail-view

Beschreibung: Dieses Widget zeigt Allergene und Zusatzstoffe zu einem Gericht an.

RatingsOverview

Typ: class extends StatelessWidget

Paket: view/detail-view

Beschreibung: Dieses Widget zeigt eine Übersicht über die Bewertungen eines Gerichts an. Hierzu zählt die durchschnittlich Abgegebene Bewertung, sowie die Gesamtzahl aller abgegebenen Bewertungen.

MealRatingDialog

Typ: class extends StatefulWidget

Paket: view/detail-view

Beschreibung: Dieses Widget zeigt einen Dialog zum Abgeben einer Bewertung für ein Gericht an.

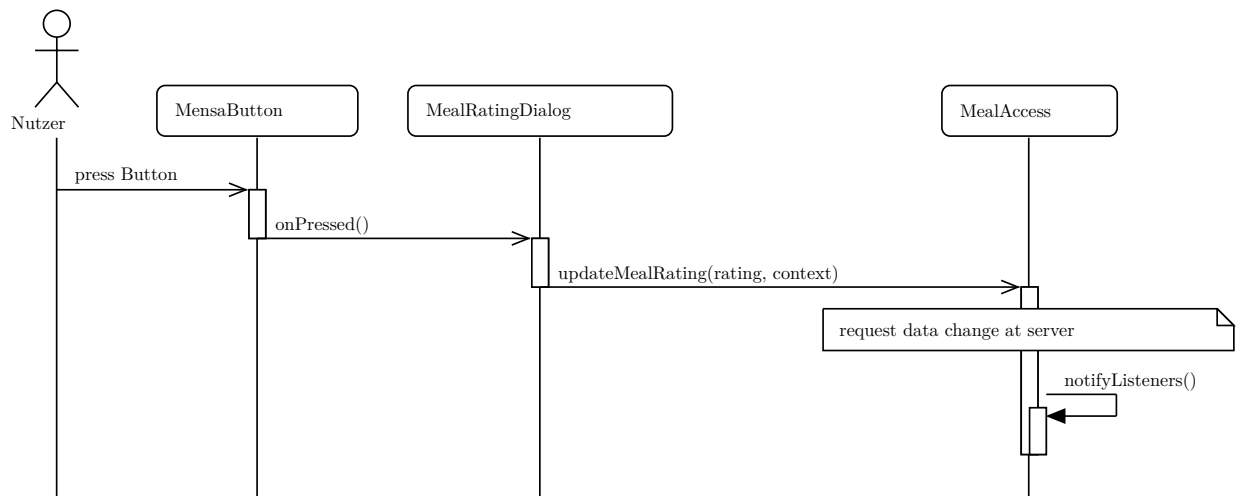
Beispielhafte Sequenzdiagramme

Abbildung 3.5: Sequenzdiagramm zum Abgeben einer Bewertung

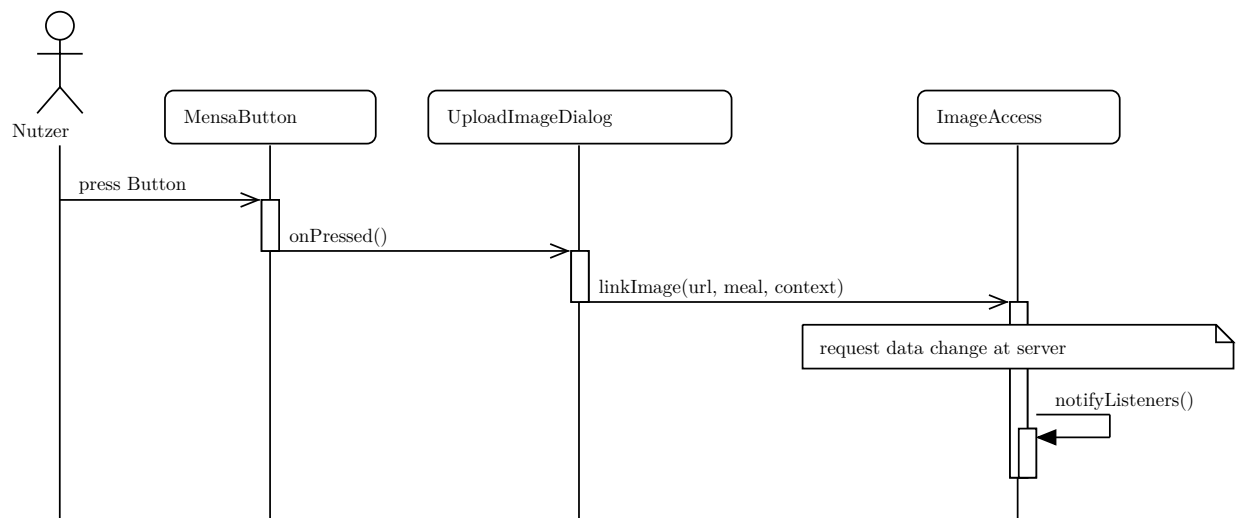


Abbildung 3.6: Sequenzdiagramm zum Verlinken eines Bildes

3.1.5 Paket favorites

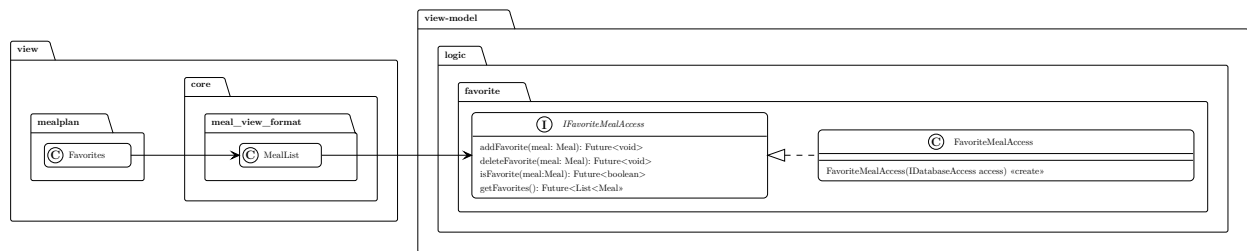


Abbildung 3.7: Klassendiagramm des Paket favorites

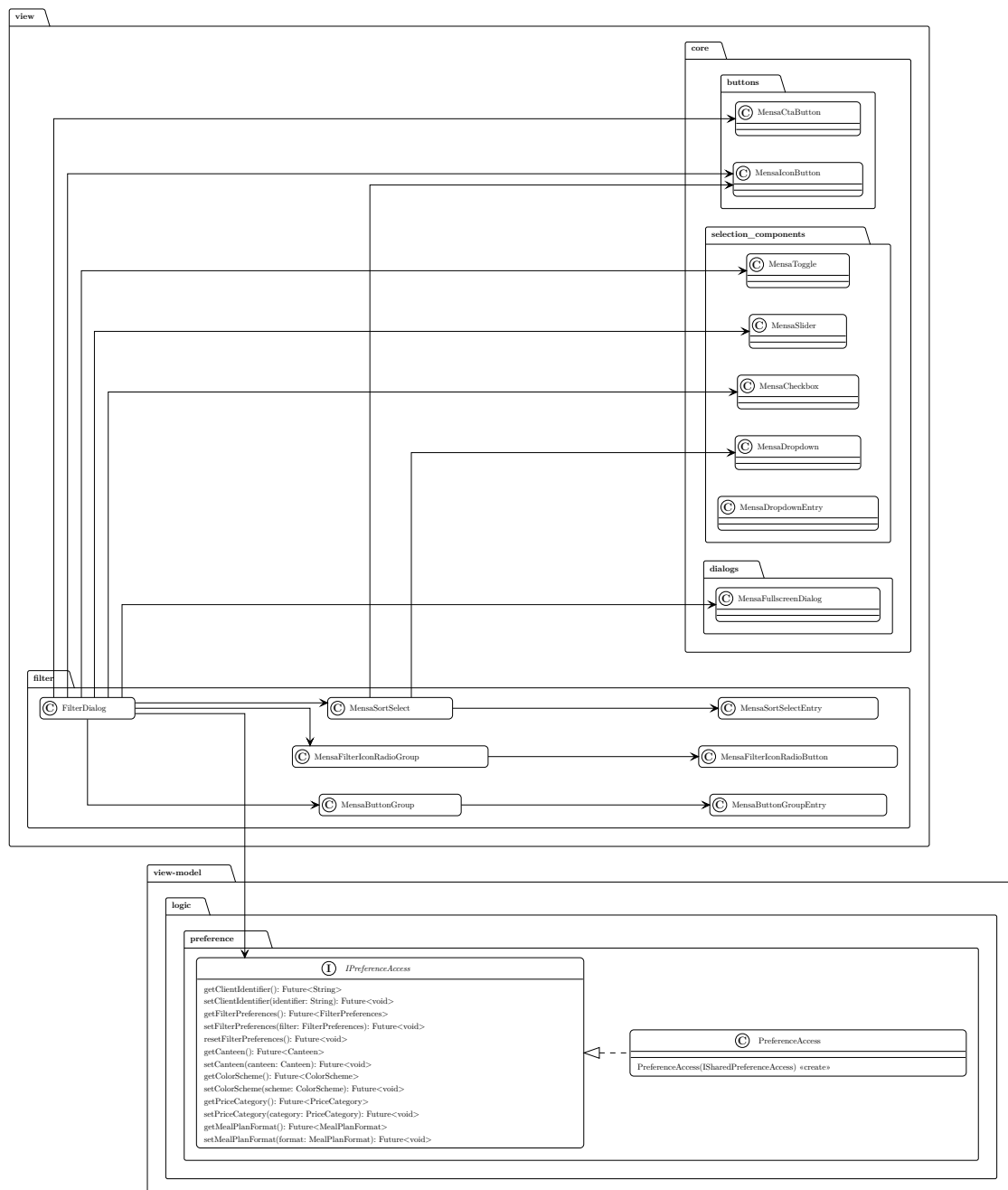
Favorites

Typ: class extends StatelessWidget

Paket: view/mealplan

Beschreibung: Dieses Widget zeigt die Favoritenansicht mit einer Liste der favorisierten Gerichte an.

3.1.6 Paket filter

Abbildung 3.8: Klassendiagramm des Paket **filter**

FilterDialog

Typ: class extends `StatefulWidget`

Paket: view/filter

Beschreibung: Dieses Widget zeigt einen Dialog zum Wählen der Filtereinstellungen an.

MensaButtonGroup

Typ: class extends StatelessWidget

Paket: view/filter

Beschreibung: Dieses Widget zeigt eine Gruppe von Schaltflächen zum Wählen von Optionen an.

MensaButtonGroupEntry

Typ: class extends StatelessWidget

Paket: view/filter

Beschreibung: Dieses Widget zeigt eine Schaltfläche in `MensaButtonGroup` an.

MensaFilterIconCheckboxGroup

Typ: class extends StatelessWidget

Paket: view/filter

Beschreibung: Diese Widget zeigt eine Gruppe von wählbaren Optionen an. Diese Optionen werden über ein Icon dargestellt.

MensaFilterIconCheckbox

Typ: class extends StatelessWidget

Paket: view/filter

Beschreibung: Dieses Widget zeigt eine Schaltfläche in `??` an. Die Schaltfläche zeigt ein Icon sowie einen Text zur Option an.

MensaSortSelect

Typ: class extends StatelessWidget

Paket: view/filter

Beschreibung: Dieses Widget zeigt Optionen zum Wählen einer Sortierung an.

MensaSortSelectEntry

Typ: class extends StatelessWidget

Paket: view/filter

Beschreibung: Dieses Widget zeigt eine Sortieroption in `MensaSortSelect` an.

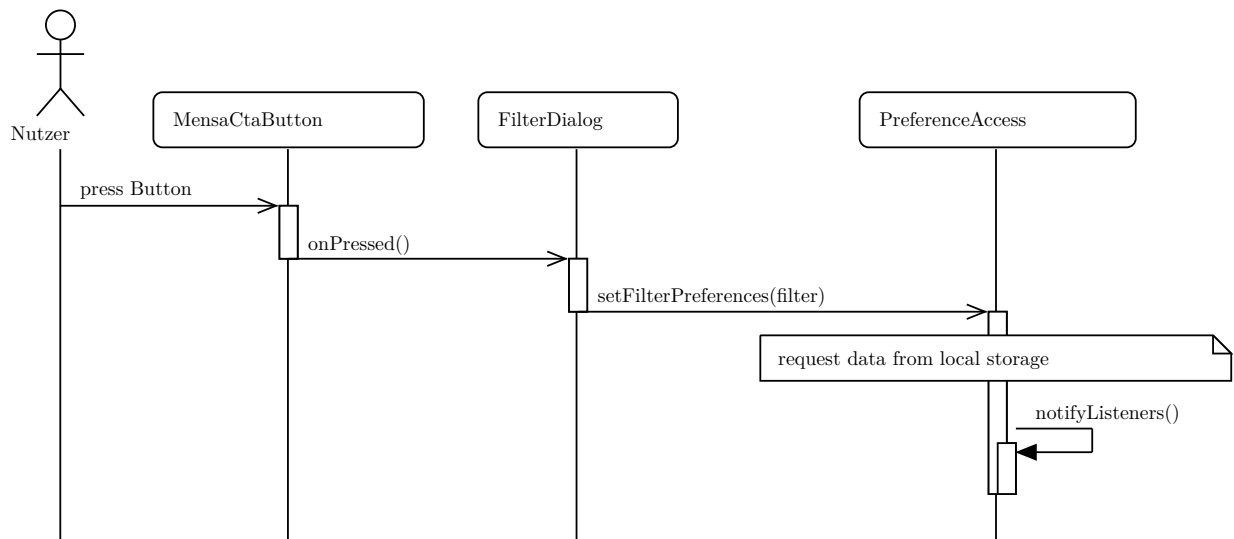
Beispielhafte Sequenzdiagramme

Abbildung 3.9: Sequenzdiagramm zum Speichern einer Filterkonfiguration

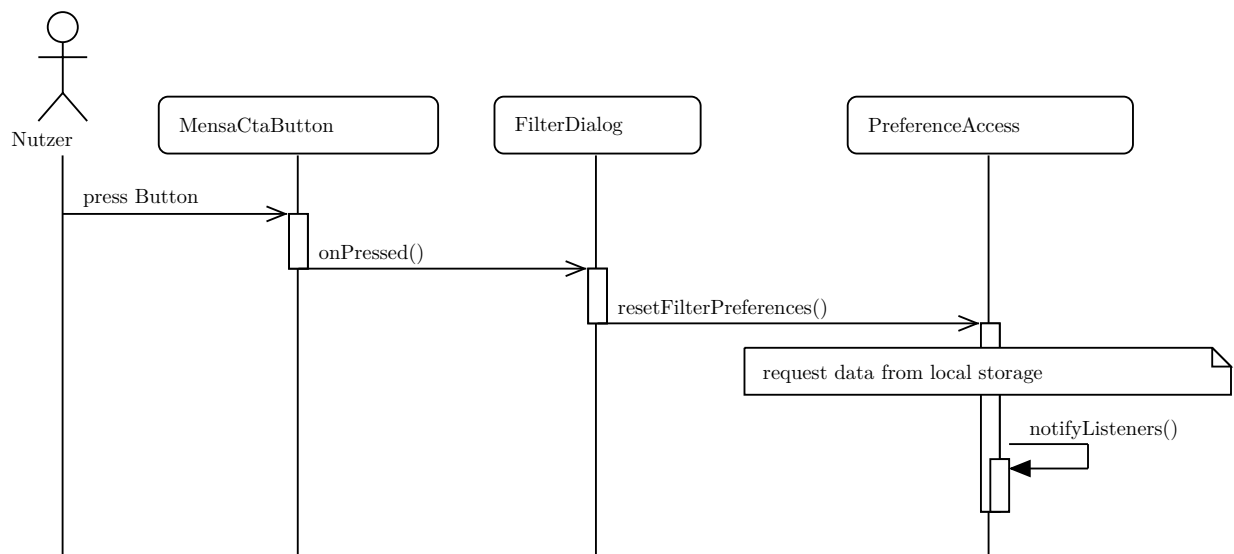


Abbildung 3.10: Sequenzdiagramm zum Zurücksetzen einer Filterkonfiguration

3.1.7 Paket images

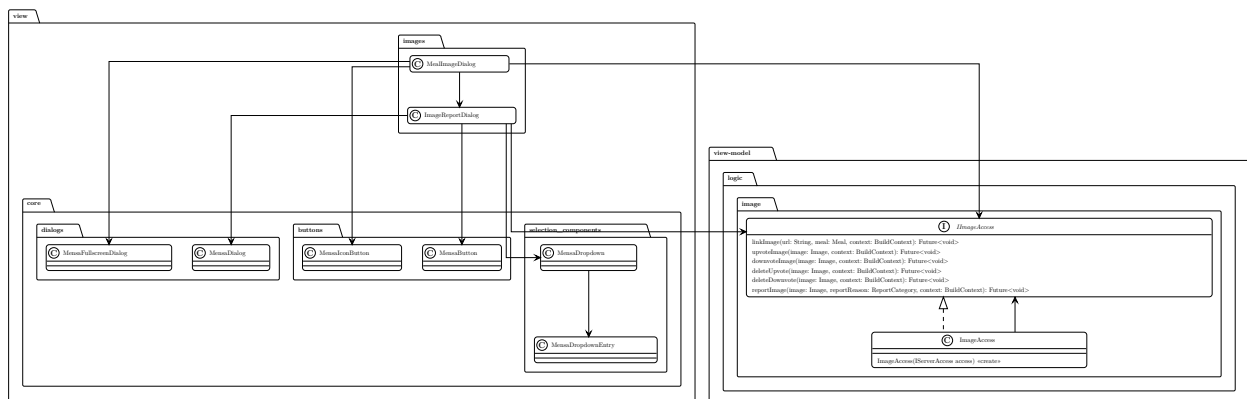


Abbildung 3.11: Klassendiagramm des Paket `images`

MealImageDialog

Typ: class implements `StatefulWidget`

Paket: `view/images`

Beschreibung: Dieses Widget ist ein Dialog und öffnet sich durch einen Klick auf das Bild in der `DetailsPage`. In dieser Ansicht kann man durch die angezeigten Bilder wechseln, Bilder bewerten und Bilder melden.

ImageReportDialog

Typ: class implements `StatefulWidget`

Paket: `view/images`

Beschreibung: Dieses Widget ist ein Dialog und öffnet sich durch das Melden eines Bildes im `MealImageDialog`. Nach der Bestätigung des `ImageReportDialogs`, befindet sich der Nutzer wieder im `MealImageDialog`. Dieser Dialog besteht aus mehreren Widgets des Cores.

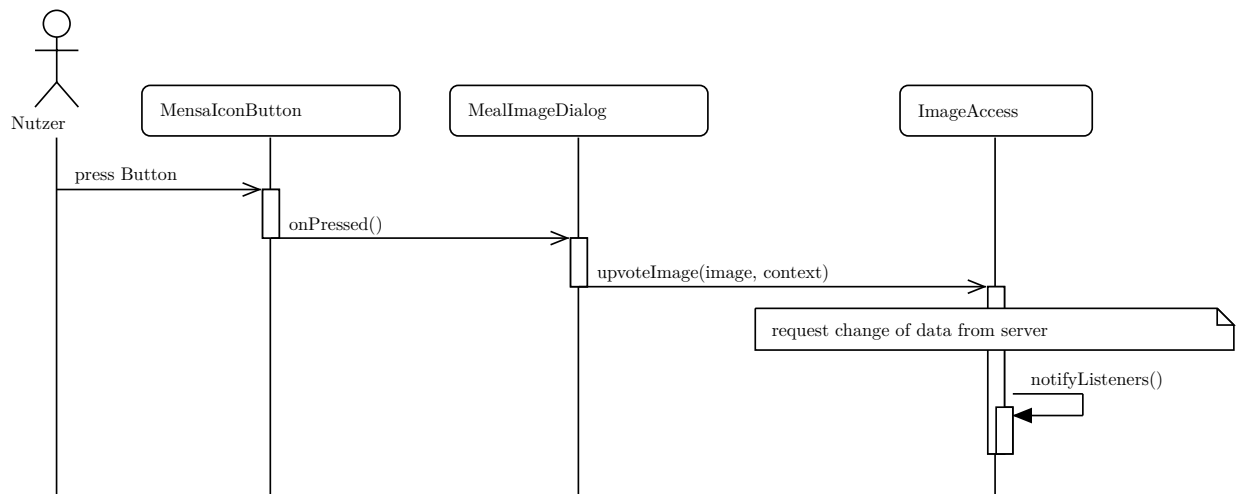
Beispielhafte Sequenzdiagramme

Abbildung 3.12: Sequenzdiagramm zum Upvoten eines Bildes

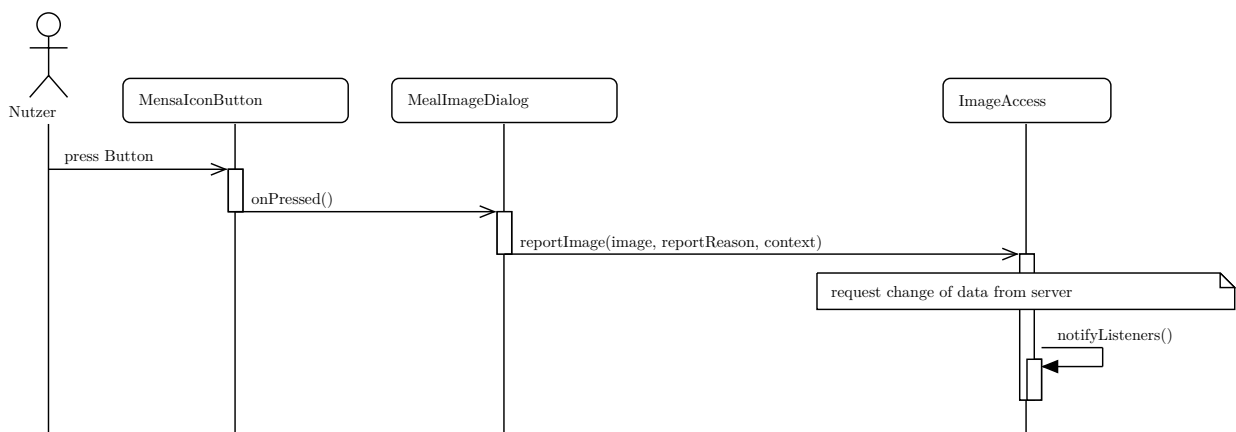


Abbildung 3.13: Sequenzdiagramm zum Melden eines Bildes

3.1.8 Paket mealplan

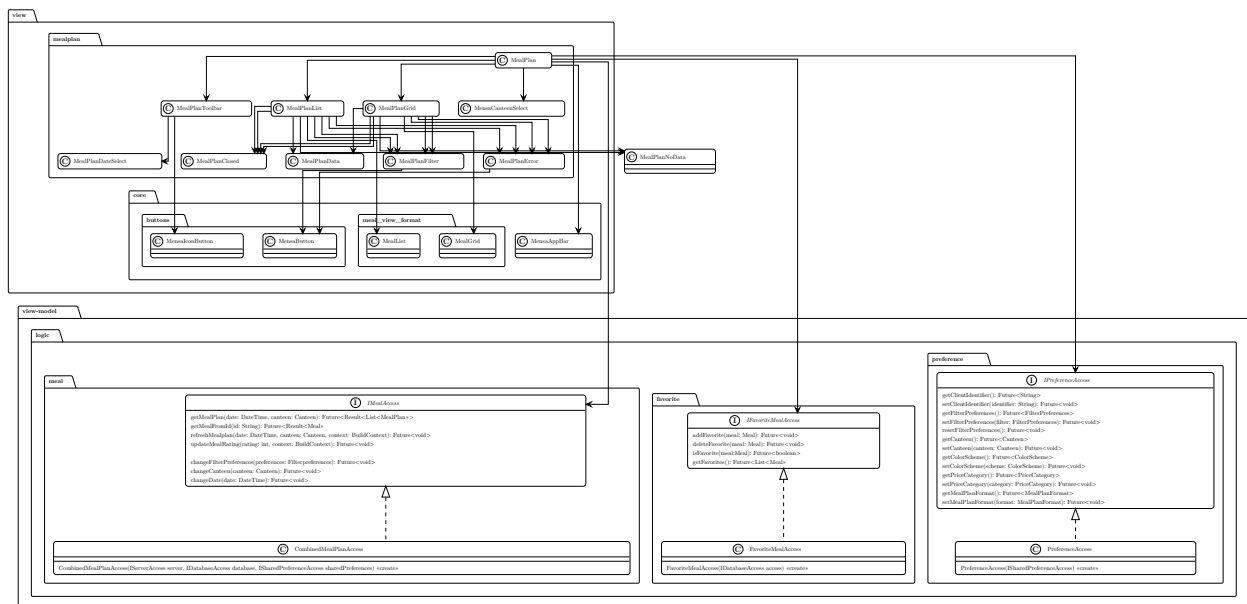


Abbildung 3.14: Klassendiagramm des mealplan-Pakets

MealPlan

Typ: class extends StatefulWidget

Paket: view/mealplan

Beschreibung: Dieses Widget zeigt die Speiseplanansicht an.

MealPlanList

Typ: class extends StatelessWidget

Paket: view/mealplan

Beschreibung: Dieses Widget zeigt den Speiseplan als Liste an.

MealPlanGrid

Typ: class extends StatelessWidget

Paket: view/mealplan

Beschreibung: Dieses Widget zeigt den Speiseplan als Galerie an.

MealPlanToolbar

Typ: class extends StatelessWidget

Paket: view/mealplan

Beschreibung: Dieses Widget zeigt die Toolbar zum Wechseln des Anzeigeformats und des Datums an.

MealPlanDateSelect

Typ: `class extends StatelessWidget`

Paket: `view/mealplan`

Beschreibung: Dieses Widget zeigt einen Dialog zur Auswahl des Datums an.

MealPlanError

Typ: `class extends StatelessWidget`

Paket: `view/mealplan`

Beschreibung: Dieses Widget zeigt an, dass keine Speiseplandaten aufgrund von Verbindungsfehlern vorhanden sind.

MealPlanClosed

Typ: `class extends StatelessWidget`

Paket: `view/mealplan`

Beschreibung: Dieses Widget zeigt an, dass die Mensa geschlossen hat.

MealPlanNoData

Typ: `class extends StatelessWidget`

Paket: `view/mealplan`

Beschreibung: Dieses Widget zeigt an, dass die App (noch) keine Speiseplandaten der Mensa hat.

MealPlanFilter

Typ: `class extends StatelessWidget`

Paket: `view/mealplan`

Beschreibung: Dieses Widget zeigt an, dass kein Gericht des Speiseplans den Filtern entspricht.

MensaCanteenSelect

Typ: `class extends StatelessWidget`

Paket: `view/mealplan`

Beschreibung: Dieses Widget zeigt ein Dropdown-Menü verschiedener Mensen an.

Beispielhaftes Sequenzdiagramm

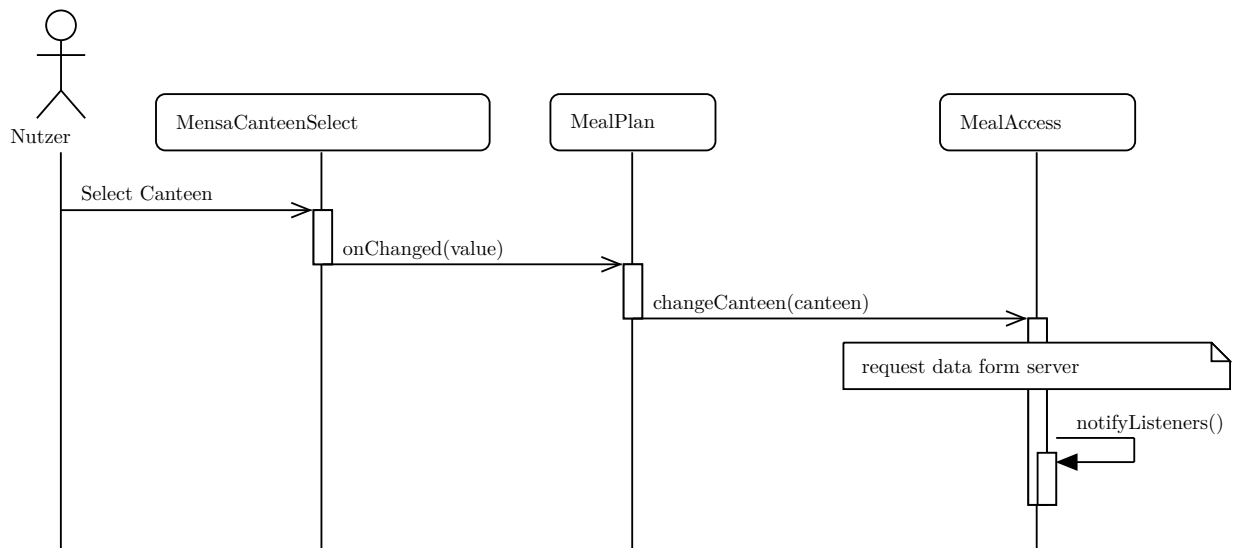


Abbildung 3.15: Sequenzdiagramm Mensa-Wechsel

3.1.9 Paket settings

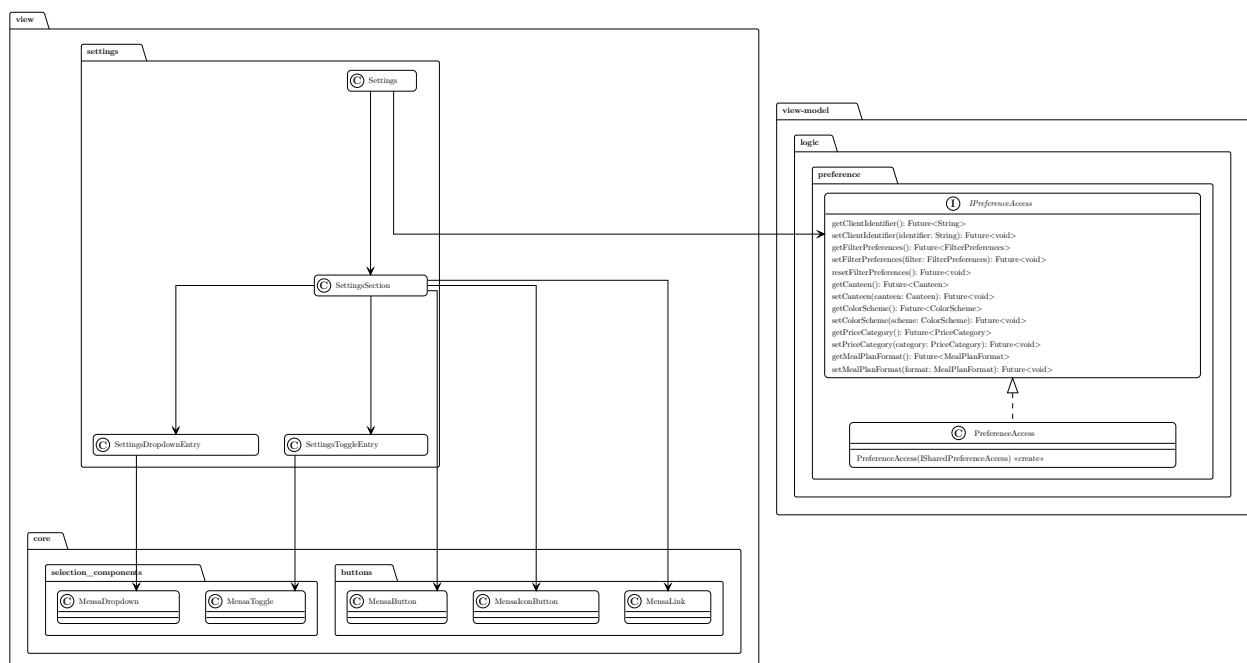


Abbildung 3.16: Klassendiagramm des Paket settings

Settings

Typ: class implements `StatefulWidget`

Paket: `view/settings`

Beschreibung: Zeigt die Einstellungen an. Diese Oberfläche besteht aus (mehreren) `SettingsSections` und anderen Widgets.

SettingsSection

Typ: class implements `StatelessWidget`

Paket: `view/settings`

Beschreibung: Dieses Widget bildet einen Abschnitt in der `Settings`-Oberfläche.

SettingsDropDownEntry

Typ: class implements `StatelessWidget`

Paket: `view/settings`

Beschreibung: Dieses Widget ist ein Eintrag in einem „Dropdown“ auf der `Settings`-Oberfläche.

SettingsToggleEntry

Typ: class implements `StatelessWidget`

Paket: `view/settings`

Beschreibung: Dieses Widget ist ein „Toggle-Button“ der für die Anforderungen der `Settings`-Oberfläche zugeschnitten ist.

Beispielhaftes Sequenzdiagramm

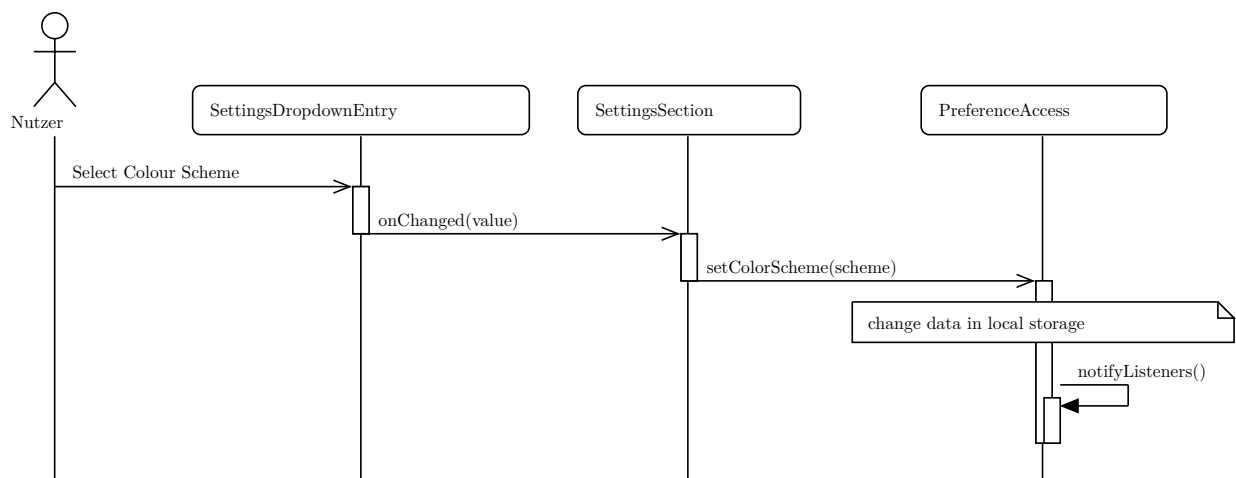


Abbildung 3.17: Sequenzdiagramm zum Wechseln zwischen Light und Dark-Mode

3.1.10 Paket logic - Interfaces zur View

Dieses Paket enthält die Logik, die für die Steuerung der Benutzeroberfläche benötigt wird.

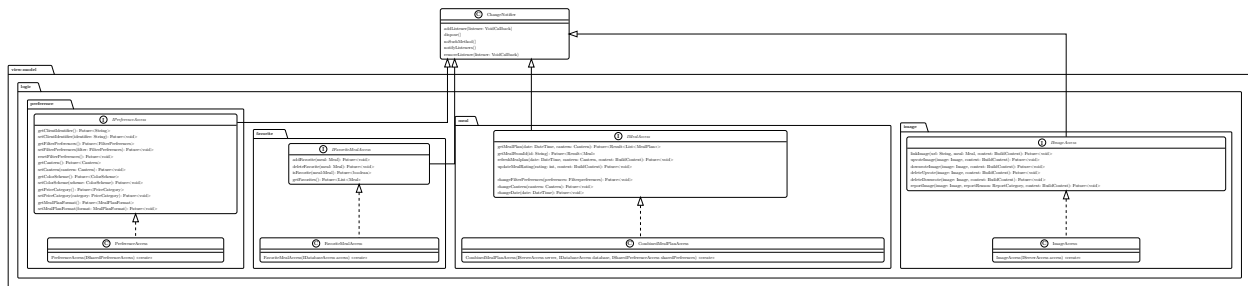


Abbildung 3.18: Klassendiagramm der Interfaces zwischen View und View-Model

Der Zugriff auf die Daten, wird asynchron durchgeführt, weshalb hier der Future-Datentyp genutzt wird.

ChangeNotifier

Typ: class

Paket: -

Beschreibung: Der ChangeNotifier ist eine aus der Provider-Library von Dart bereitgestellte Klasse, die ihre Listener über Veränderungen von Daten informieren kann. Der ChangeNotifier ist eine Form des Observable-Entwurfsmusters.

IMealAccess

Typ: interface extends ChangeNotifier

Paket: view-model/logic/meal

Beschreibung: Dieses Interface erlaubt das Laden des Speiseplans oder einzelner Gerichte und Änderungen an Gerichten.

Methoden:

- + `getMealPlan(DateTime date, Canteen canteen): Future<Result<List<MealPlan>>>`
Eine Liste der verschiedenen Speisepläne der Linien werden zurückgegeben oder einen Fehler, falls diese nicht vorhanden sind bzw. nicht geladen werden konnten.
- + `getMealFromId(String id): Future<Result<Meal>>`
Das Gericht mit der übergebenen ID wird zurückgegeben oder ein Fehler, falls dieses nicht vorhanden ist oder nicht geladen werden konnte.
- + `refreshMealplan(date: DateTime, canteen: Canteen, context: BuildContext): Future<void>`
Der gesamte Speiseplan wird beim Server neu angefragt und in der Datenbank und

der Benutzeroberfläche aktualisiert. Falls dies nicht möglich ist, so wird eine temporäre Fehlermeldung angezeigt, wozu `context` benötigt wird.

+ `updateMealRating(rating: int, context: BuildContext): Future<void>`

Die Bewertung eines Nutzers zu einem bestimmten Gericht wird mit dem Server synchronisiert. Falls dies nicht möglich ist, so wird eine temporäre Fehlermeldung angezeigt, wozu `context` benötigt wird.

+ `changeFilterPreferences(preferences: FilterPreferences): Future<void>`

Die übergebenen `FilterPreferences` werden gespeichert und die Widgets über die Änderung informiert.

+ `changeCanteen(canteen: Canteen): Future<void>`

Der Speiseplan der übergebenen Mensa wird angezeigt.

+ `changeDate(date: Date): Future<void>`

Der Speiseplan für den übergebenen Tag wird angezeigt.

CombinedMealPlanAccess

Typ: class implements `IMealAccess`

Paket: `view-model/logic/meal`

Beschreibung: Weiterleitung der Anfragen von Daten zu Gerichten oder Speiseplänen an die Datenbank oder an den Server, wobei bei Fehlern eine temporäre Fehlermeldung ausgegeben wird. Die Listener des `ChangeNotifiers` werden über Änderungen in den Daten benachrichtigt.

Methoden:

+ `CombinedMealAccess(IServerAccess, IDatabaseAccess, ISharedPreferenceAccess):`

`<<create>>`

Konstruktor der einen kombinierten Zugriffspunkt auf den Speiseplan ermöglicht.

Beispielhaftes Sequenzdiagramm

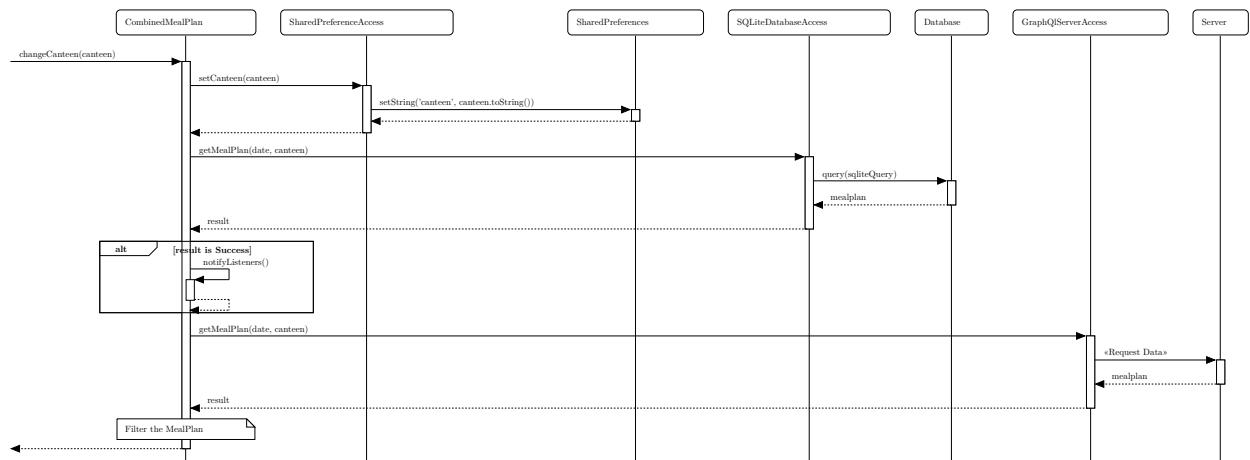


Abbildung 3.19: Sequenzdiagramm Mensa-Wechsel

IFavoriteMealAccess

Typ: interface extends `ChangeNotifier`

Paket: `view-model/logic/favorite`

Beschreibung: Dieses Interface erlaubt das Laden und Ändern der Favoriten des Nutzers.

Methoden:

- + `addFavorite(meal: Meal): Future<void>`
Das übergebene Gericht wird zu den Favoriten des Nutzers hinzugefügt.
- + `deleteFavorite(meal: Meal): Future<void>`
Das übergebene Gericht wird aus den Favoriten des Nutzers entfernt.
- + `isFavorite(meal: Meal): Future<boolean>`
Es wird `true` zurückgegeben, falls das übergebene Gericht ein Favorit ist.
- + `getFavorites(): Future<List<Meal>>`
Eine Liste aller vom Nutzer favorisierten Gerichte wird zurückgegeben.

FavoriteMealAccess

Typ: class implements `IFavoriteMealAccess`

Paket: `view-model/logic/favorite`

Beschreibung: Weiterleitung der Anfragen zu Favoriten an die Datenbank. Die Listener des `ChangeNotifier` werden über Änderungen in den Daten benachrichtigt.

Methoden:

+ FavoriteMealAccess(IDatabaseAccess access): <<create>>

Konstruktor welcher einen Zugriffspunkt auf die in einer lokal gespeicherten Datenbank erstellt.

Beispielhaftes Sequenzdiagramm

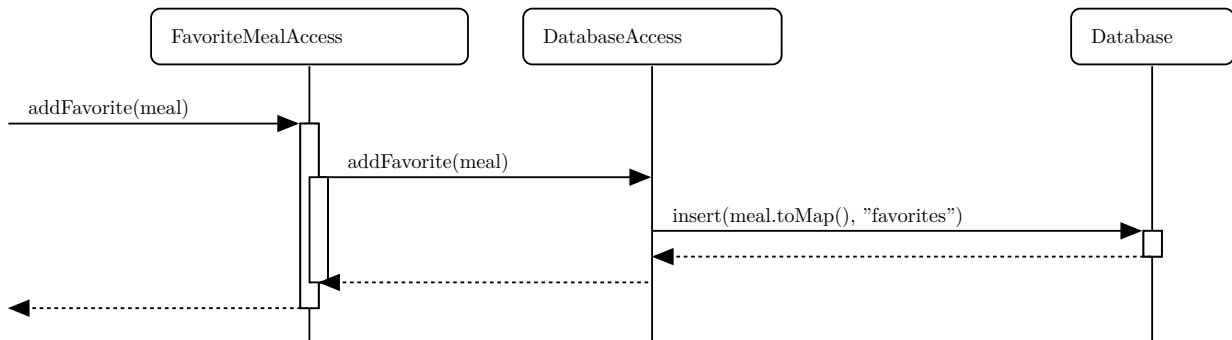


Abbildung 3.20: Sequenzdiagramm Hinzufügen eines Favoriten

IPreferenceAccess

Typ: interface extends ChangeNotifier

Paket: view-model/logic/preference

Beschreibung: Dieses Interface erlaubt das Laden und Ändern von verschiedenen Einstellungen und Clientinformationen.

Methoden:

- + getClientIdentifier(): Future<String>
Der Geräteidentifikator wird zurückgegeben.
- + setClientIdentifier(identifizier: String): Future<void>
Der übergebene Identifikator wird als Geräteidentifikator gespeichert.
- + getFilterPreferences(): Future<FilterPreferences>
Gibt die vom Nutzer gespeicherten Filtereinstellungen zurück.
- + setFilterPreferences(FilterPreference filter): Future<void>
Die übergebene Filterkonfiguration wird gespeichert.
- + resetFilterPreferences(FilterPreference filter): Future<void>
Die Filterkonfiguration wird auf Standart-Werte zurückgesetzt, sodass alle Gerichte angezeigt werden.
- + getCanteen(): Future<Canteen>
Die zuletzt angezeigte Mensa wird zurückgegeben.
- + setCanteen(Canteen canteen): Future<void>
Die übergebene Mensa wird gespeichert.
- + getColorScheme(): Future<ColorScheme>
Das gespeicherte Farbschema wird zurückgegeben.

- + `setColorScheme(ColorScheme scheme): Future<void>`
Das übergebene Farbschema wird gespeichert.
- + `getPriceCategory(): Future<PriceCategory>`
Die eingestellte Preisklasse wird zurückgegeben.
- + `setPriceCategory(PriceCategory category): Future<void>`
Die übergebene Preisklasse wird gespeichert.
- + `getMealPlanFormat(): Future<MealPlanFormat>`
Die zuletzt gewählte Darstellungsform der Speiseplanansicht wird zurückgegeben.
- + `setMealPlanFormat(MealPlanFormat format): Future<void>`
Die übergebene Darstellungsform wird gespeichert.

PreferenceAccess

Typ: class implements IPreferenceAccess

Paket: view-model/logic/preference

Beschreibung: Weiterleitung der Anfragen zu Einstellungen oder Nutzerpräferenzen an den lokalen Speicher, wobei bei Fehlern auf voreingestellte Werte zurückgegriffen wird, welche dann im local Storage gespeichert werden. Die Listener des ChangeNotifiers werden über Änderungen in den Daten benachrichtigt.

Methoden:

- + `PreferenceAccess(ISharedPreferenceAccess access): <<create>>`
Konstruktor welcher einen Zugriffspunkt auf die lokal gespeicherten Einstellungsdaten erstellt.

Beispielhaftes Sequenzdiagramm

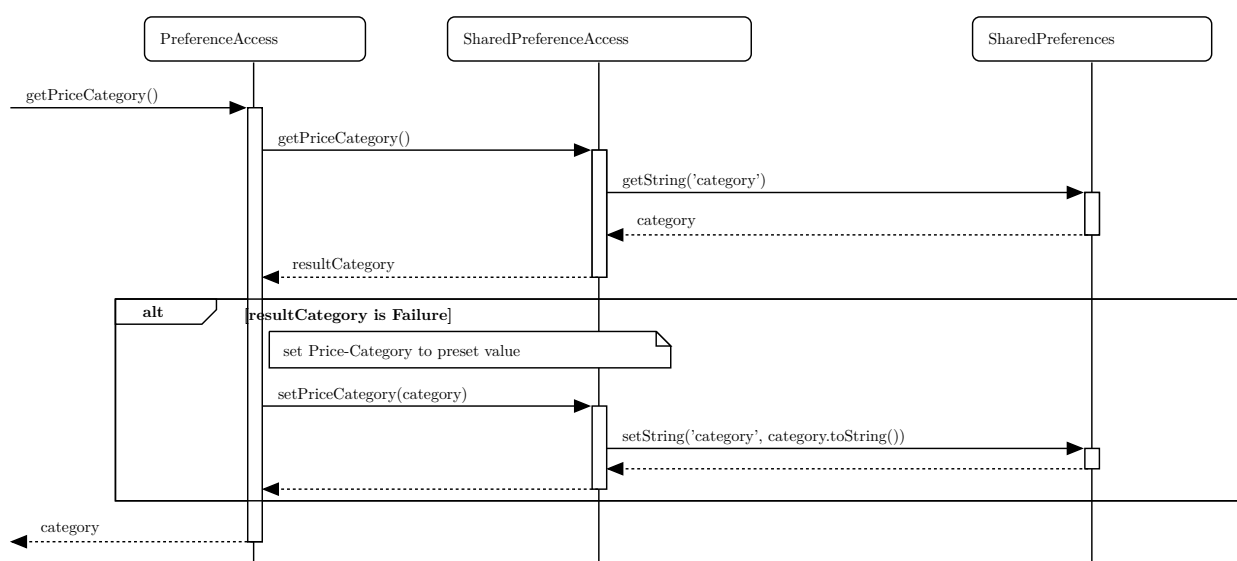


Abbildung 3.21: Sequenzdiagramm Anfragen der Preiskategorie

IIImageAccess

Typ: interface extends `ChangeNotifier`

Paket: `view-model/logic/image`

Beschreibung: Dieses Interface erlaubt das Verlinken, Bewerten und Melden von Bildern.

Methoden:

- + `linkImage(url:String, meal: Meal, context: BuildContext): Future<void>`
Die übergebene URL wird serverseitig mit dem übergebenen Bild verknüpft. Falls dies nicht möglich ist, so wird eine temporäre Fehlermeldung angezeigt, wozu `context` benötigt wird.
- + `upvoteImage(image: Image, context: BuildContext): Future<void>`
Die Bewertung des übergebenen Bildes wird dem Server übermittelt. Falls dies nicht möglich ist, so wird eine temporäre Fehlermeldung angezeigt, wozu `context` benötigt wird.
- + `downvoteImage(image: Image, context: BuildContext): Future<void>`
Die Bewertung des übergebenen Bildes wird dem Server übermittelt. Falls dies nicht möglich ist, so wird eine temporäre Fehlermeldung angezeigt, wozu `context` benötigt wird.
- + `deleteUpvote(image: Image, context: BuildContext): Future<void>`
Die Entfernung der positiven Bewertung des übergebenen Bildes wird dem Server übermittelt. Falls dies nicht möglich ist, so wird eine temporäre Fehlermeldung angezeigt, wozu `context` benötigt wird.
- + `deleteDownvote(image: Image, context: BuildContext): Future<void>`
Die Entfernung der negativen Bewertung des übergebenen Bildes wird dem Server übermittelt. Falls dies nicht möglich ist, so wird eine temporäre Fehlermeldung angezeigt, wozu `context` benötigt wird.
- + `reportImage(image: Image, reportReason: ReportCategory, context: BuildContext): Future<void>`
Die Meldung des übergebenen Bildes wird dem Server übermittelt. Falls dies nicht möglich ist, so wird eine temporäre Fehlermeldung angezeigt, wozu `context` benötigt wird.

ImageAccess

Typ: class implements `IIImageAccess`

Paket: `view-model/logic/image`

Beschreibung: Weiterleitung der Anfragen zu Bildern an den Server, wobei bei Fehlern eine temporäre Fehlermeldung ausgegeben wird. Die Listener des `ChangeNotifiers` werden über Änderungen in den Daten benachrichtigt.

Methoden:

- + `ImageAccess(IServerAccess access): <<create>>`
Konstruktor welcher einen Zugriffspunkt auf die Serverfunktionalitäten für Bilder erstellt.

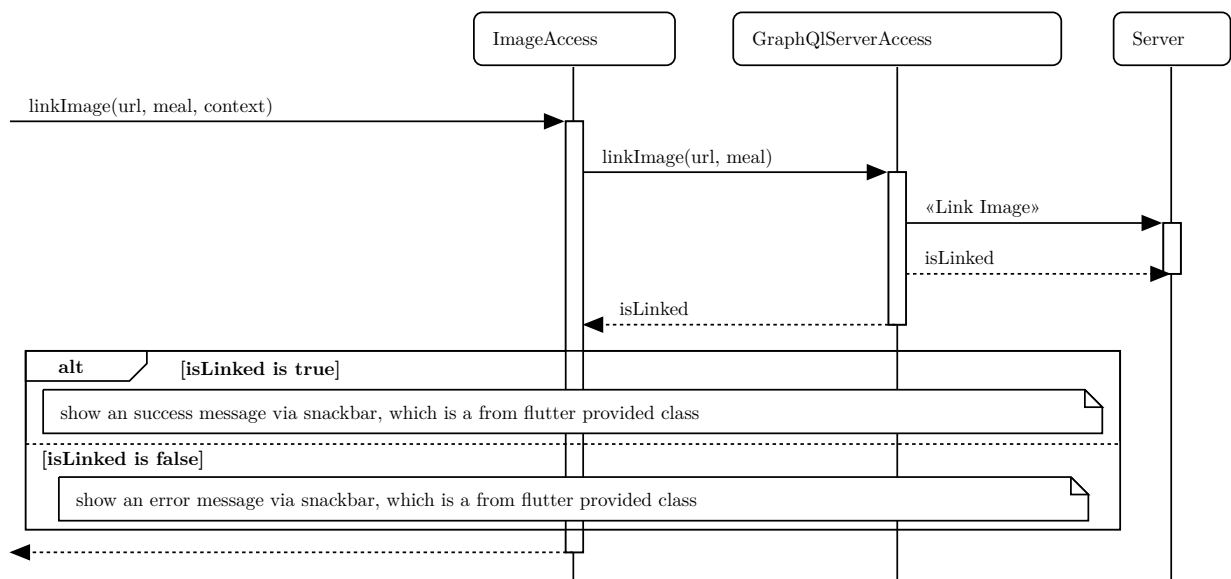
Beispielhaftes Sequenzdiagramm

Abbildung 3.22: Sequenzdiagramm Verlinkung eines Bildes

3.1.11 Paket repository

Dieses Paket enthält die Interfaces, die vom Model implementiert werden, und die dafür benötigten Klassen für Daten und Error-Handling. Das View-Model soll hierbei die Interfaces bereitstellen, weshalb diese Interfaces im View-Model enthalten sind.

3.1.12 Paket data-types

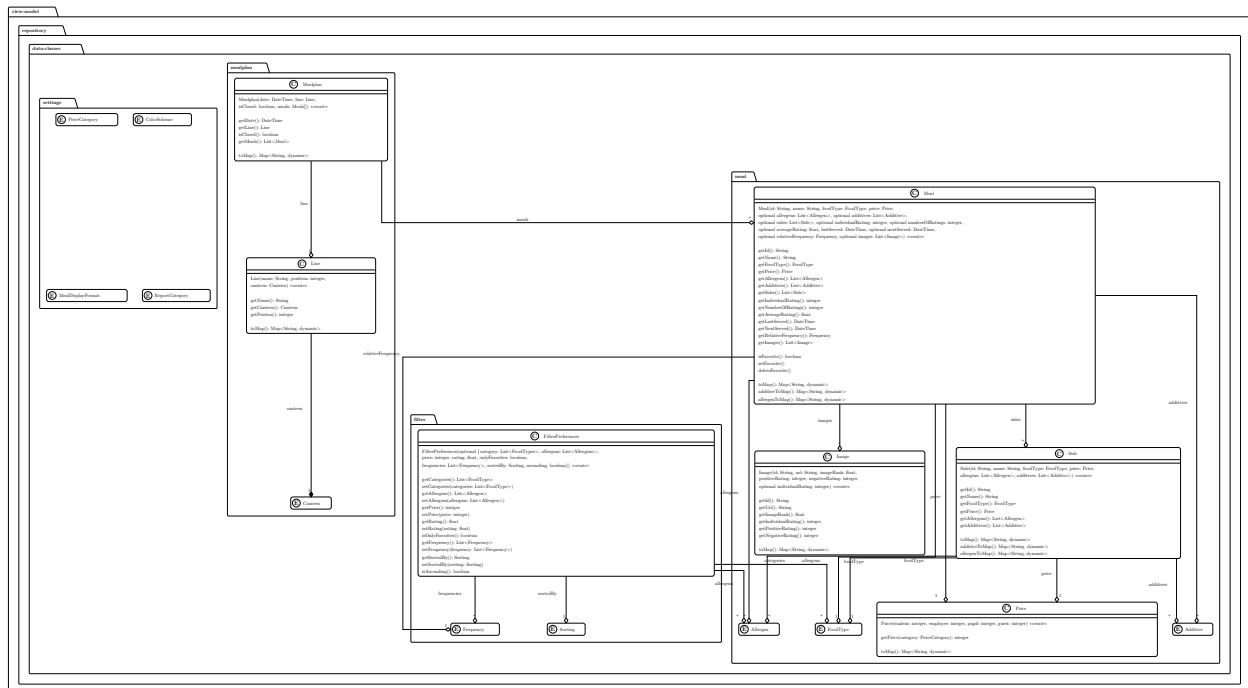


Abbildung 3.23: Klassendiagramm für Speiseplandaten

Meal

Typ: class

Paket: view-model/repository/data-classes/meal

Beschreibung: Klasse, die ein Gericht modelliert.

Methoden:

```
+ Meal(id: String, name: String, foodType: FoodType, price: Price,  
    optional allergens: List<Allergen>, optional additives: List<Additive>,  
    optional sides: List<Side>, optional individualRating: integer, optional  
    numberOfRatings: integer, optional averageRating: float, lastServed:  
    DateTime, optional nextServed: DateTime, optional relativeFrequency:  
    Frequency, optional images: List<Image>, isFavorite: boolean) <<create>>
```

Konstruktor, der alle Attribute initialisiert, wobei nicht alle Attribute zwangsläufig benötigt

werden. Die Attribute, die nicht zwangsläufig benötigt werden, sind mit dem Wort `optional` gekennzeichnet.

- + `getId(): String`
Gibt den Identifikator des Gerichts zurück.
- + `getName(): String`
Gibt den Namen des Gerichts zurück.
- + `getFoodType(): FoodType`
Gibt die Art des Gerichts zurück.
- + `getPrice(): Price`
Gibt den Preis des Gerichts zurück.
- + `getAllergens(): List<Allergen>`
Gibt alle Allergene des Gerichts zurück.
- + `getAdditives(): List<Additive>`
Gibt alle Zusatzstoffe des Gerichts zurück.
- + `getSides(): List<Side>`
Gibt alle Beilagen des Gerichts zurück.
- + `getIndividualRating(): integer`
Gibt die Bewertung des Nutzers für das Gericht zurück.
- + `getNumberOfRatings(): integer`
Gibt die Gesamtanzahl der Bewertungen des Gerichts zurück.
- + `getAverageRating(): float`
Gibt die Durchschnittliche Bewertung des Gerichts zurück.
- + `getLastServed(): DateTime`
Gibt das Datum des Tages zurück, an dem das Gericht zuletzt serviert wurde.
- + `getNextServed(): DateTime`
Gibt das Datum des Tages zurück, an dem das Gericht das nächste Mal serviert wird.
- + `getRelativeFrequency(): Frequency`
Gibt die Kategorie der relativen Häufigkeit zurück, mit dem das Gericht angeboten wird.
- + `getImages(): List<Image>`
Gibt alle Bilder zurück, die zu dem Gericht gehören.
- + `isFavorite(): boolean`
Gibt zurück, ob das Gericht ein Favorit ist.
- + `setFavorite()`
Setzt den Wert `isFavorite` auf `true`.
- + `deleteFavorite()`
Setzt den Wert `isFavorite` auf `false`.
- + `Map<String, dynamic> toMap()`
Gibt eine Zuordnung von den Attributen als String zu den Werten zurück, die dafür genutzt wird, die Werte in der Datenbank zu speichern. Dabei werden alle Attribute in die Liste eingefügt, die initialisiert sind und in der Datenbank für Meal benötigt werden (siehe Datenbankschema).

+ Map<String, dynamic> additiveToMap()

Gibt alle Zusatzstoffe in einer Map zurück, die in der Datenbank gespeichert werden kann.

+ Map<String, dynamic> allergenToMap()

Gibt alle Allergene in einer Map zurück, die in der Datenbank gespeichert werden kann.

Side

Typ: class

Paket: view-model/repository/data-classes/meal

Beschreibung: Klasse, die eine Beilage modelliert.

Methoden:

+ Side(id: String, name: String, foodType: FoodType, price: Price, allergens: List<Allergen>, additives: List<Additive>) <<create>>

Konstruktor, der alle Attribute mit den übergebenen Werten initialisiert.

+ getId(): String

Gibt den Identifikator der Beilage zurück.

+ getName(): String

Gibt den Namen der Beilage zurück.

+ getFoodType(): FoodType

Gibt die Art der Beilage zurück.

+ getPrice(): Price

Gibt den Preis der Beilage zurück.

+ getAllergens(): List<Allergen>

Gibt alle Allergene der Beilage zurück.

+ getAdditives(): List<Additive>

Gibt alle Zusatzstoffe der Beilage zurück.

+ Map<String, dynamic> toMap()

Gibt eine Zuordnung von den Attributen als String zu den Werten zurück, die dafür genutzt wird, die Werte in der Datenbank zu speichern. Dabei werden alle Attribute in die Liste eingefügt, die in der Datenbank für Side benötigt werden (siehe Datenbankschema).

+ Map<String, dynamic> additiveToMap()

Gibt alle Zusatzstoffe in einer Map zurück, die in der Datenbank gespeichert werden kann.

+ Map<String, dynamic> allergenToMap()

Gibt alle Allergene in einer Map zurück, die in der Datenbank gespeichert werden kann.

Image

Typ: class

Paket: view-model/repository/data-classes/meal

Beschreibung: Klasse, die ein Bild modelliert

Methoden:

- + `Image(id: String, url: String, imageRank: float, positiveRating: integer, negativeRating: integer, optional individualRating: integer) <<create>>`
Konstruktor, der alle Attribute initialisiert, wobei nicht alle Attribute zwangsläufig benötigt werden. Die Attribute, die nicht zwangsläufig benötigt werden, sind mit dem Wort `optional` gekennzeichnet.
- + `getId(): String`
Gibt den Identifikator des Bildes zurück.
- + `getUrl(): String`
Gibt den Link zum Bild zurück.
- + `getImageRank(): float`
Gibt den Bilddrang des Bildes zurück.
- + `getIndividualRating(): integer`
Gibt die Bewertung des Nutzers für das Bild zurück.
- + `getPositiveRating(): integer`
Gibt die Anzahl an positiven Bewertungen des Bildes zurück.
- + `getPositiveRating(): integer`
Gibt die Anzahl an negativen Bewertungen des Bildes zurück.
- + `Map<String, dynamic> toMap()`
Gibt eine Zuordnung von den Attributen als String zu den Werten zurück, die dafür genutzt wird, die Werte in der Datenbank zu speichern. Dabei werden alle Attribute in die Liste eingefügt, die in der Datenbank für Image benötigt werden (siehe Datenbankschema).

Price

Typ: class

Paket: view-model/repository/data-classes/meal

Beschreibung: Klasse, die den Preis eines Gerichts oder einer Beilage enthält

Methoden:

- + `Price(student: integer, employee: integer, pupil: integer, guest: integer) <<create>>`
Konstruktor, der alle Attribute mit den übergebenen Werten initialisiert.
- + `getPrice(category: PriceCategory): integer`
Gibt den Preis in Abhängigkeit von der Preiskategorie zurück
- + `Map<String, dynamic> toMap()`
Gibt eine Zuordnung von den Attributen als String zu den Werten zurück, die dafür genutzt wird, die Werte in der Datenbank zu speichern. Dabei werden alle Attribute in die Liste eingefügt, die in der Datenbank für Price benötigt werden (siehe Datenbankschema).

Mealplan

Typ: class

Paket: view-model/repository/data-classes/mealplan

Beschreibung: Klasse, die den Speiseplan eines bestimmten Tages an einer bestimmten Linie enthält.

Methoden:

- + Mealplan(date: DateTime, line: Line, isClosed: boolean, meals: Meals[]) <<create>>
Konstruktor, der alle Attribute mit den übergebenen Werten initialisiert.
- + getDate(): DateTime
Gibt das Datum des Speiseplans zurück.
- + getLine(): Line
Gibt die Linie des Speiseplans zurück.
- + isClosed(): boolean
Gibt zurück, ob die ausgewählte Mensa am ausgewählten Tag geöffnet ist
- + getMeals(): List<Meal>
Gibt alle Gerichte zurück, die am ausgewählten Tag an der ausgewählten Linie serviert werden.
- + Map<String, dynamic> toMap()
Gibt eine Zuordnung von den Attributen als String zu den Werten zurück, die dafür genutzt wird, die Werte in der Datenbank zu speichern. Dabei werden alle Attribute in die Liste eingefügt, die in der Datenbank für MealPlan benötigt werden (siehe Datenbankschema).

Line

Typ: class

Paket: view-model/repository/data-classes/mealplan

Beschreibung: Klasse, die eine Line in einer Mensa modelliert.

Methoden:

- + Line(name: String, position: integer, canteen: Canteen) <<create>>
Konstruktor, der alle Attribute mit den übergebenen Werten initialisiert.
- + getName(): String
Gibt den Namen der Linie zurück.
- + getCanteen(): Canteen
Gibt die Mensa zurück, in der sich die Linie befindet.
- + getPosition(): integer
Gibt die Position der Linie zurück, an der die Linie auf der Webseite des Studierendenwerks angezeigt wurde.
- + Map<String, dynamic> toMap()
Gibt eine Zuordnung von den Attributen als String zu den Werten zurück, die dafür genutzt wird, die Werte in der Datenbank zu speichern. Dabei werden alle Attribute in die Liste eingefügt, die in der Datenbank für Line benötigt werden (siehe Datenbankschema).

FilterPreferences

Typ: class

Paket: view-model/repository/data-classes/filter

Beschreibung: Klasse, die eine Filterkonfiguration modelliert.

Methoden:

- + `FilterPreferences(optional {category: List<FoodTypes>, allergens: List<Allergens>, price: integer, rating: float, onlyFavorites: boolean, frequencies: List<Frequency>, sortBy: Sorting, ascending: boolean})`
`<<create>>`
Erstellt eine Filterkonfiguration anhand der übergebenen Parameter.
- + `getCategories(): List<FoodType>`
Gibt die Gerichtsarten zurück, nach denen gefiltert werden soll.
- + `setCategories(categories: List<FoodType>)`
Setzt die Gerichtsarten fest, nach denen gefiltert werden soll.
- + `getAllergens(): List<Allergen>`
Gibt die Allergene zurück, nach denen gefiltert werden soll.
- + `setAllergens(allergens: List<Allergen>)`
Setzt die Allergene fest, nach denen gefiltert werden soll.
- + `getPrice(): integer`
Gibt den Maximalpreis zurück, nach dem gefiltert werden soll.
- + `setPrice(price: integer)`
Setzt den Maximalpreis fest, nach dem gefiltert werden soll.
- + `getRating(): float`
Gibt die Mindestbewertung zurück, nach der gefiltert werden soll.
- + `setRating(rating: float)`
Setzt die Mindestbewertung fest, nach der gefiltert werden soll.
- + `getFrequency(): List<Frequency>`
Gibt an, nach welchen Häufigkeiten gefiltert werden soll.
- + `setFrequency(frequency: List<Frequency>)`
Setzt fest, nach welchen Häufigkeiten gefiltert werden soll.
- + `isOnlyFavorites(): boolean`
Gibt an, ob nach Favoriten gefiltert werden soll.
- + `getSortedBy(): Sorting`
Gibt die Sortierung an, nach der die Ergebnisse sortiert werden sollen.
- + `setSortedBy(sorting: Sorting)`
Setzt die Sortierung fest, nach der die Ergebnisse sortiert werden sollen.
- + `isAscending(): boolean`
Gibt an, ob die Ergebnisse auf- oder absteigend sortiert werden sollen.

PriceCategory

Typ: enum

Paket: view_model/repository/data_classes/settings

Beschreibung: Eine Menge alle Preiseinstufungen, die als Filtermöglichkeit verwendet wird.

ColorScheme

Typ: enum

Paket: view_model/repository/data_classes/settings

Beschreibung: Die Menge aller Farbschemen der Nutzeroberfläche.

MealDisplayFormat

Typ: enum

Paket: view_model/repository/data_classes/settings

Beschreibung: Eine Auswahl an Darstellungsmöglichkeiten der Gerichte in der Nutzeroberfläche.

ReportCategory

Typ: enum

Paket: view_model/repository/data_classes/settings

Beschreibung: Eine Menge aller Report-Kategorien, die für das Melden eines Bildes verwendet werden.

Canteen

Typ: enum

Paket: view_model/repository/data_classes/mealplan

Beschreibung: Eine Auswahl aller Kantinen des Studierendenwerks.

Frequency

Typ: enum

Paket: view_model/repository/data_classes/filter

Beschreibung: Menge aller Häufigkeitstypen.

Sorting

Typ: enum

Paket: view_model/repository/data_classes/filter

Beschreibung: Eine Auswahl aller Sortioptionen.

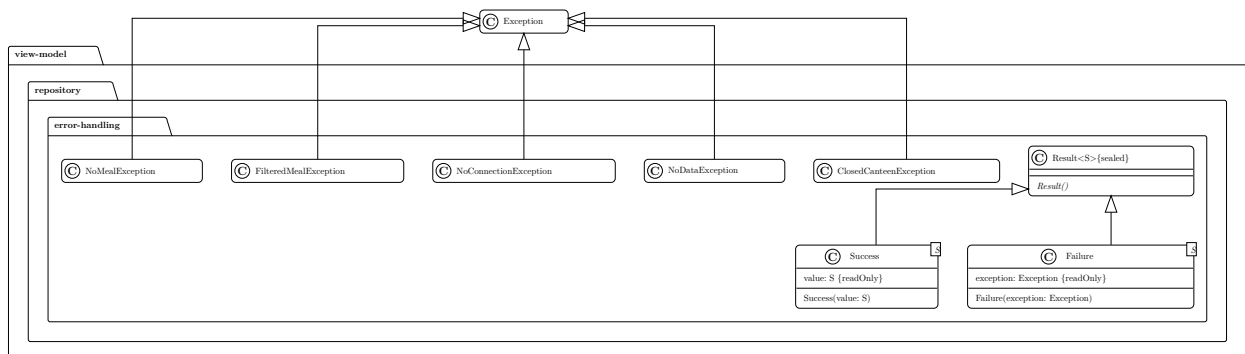
Allergen**Typ:** enum**Paket:** view_model/repository/data_classes/meal**Beschreibung:** Menge aller Allergene. Eine Auflistung aller Allergene ist unter Datenbankschema Client zu finden.**FoodType****Typ:** enum**Paket:** view_model/repository/data_classes/meal**Beschreibung:** Menge aller Gerichtstypen. Eine Auflistung aller Gerichtstypen ist unter Datenbankschema Client zu finden.**Additive****Typ:** enum**Paket:** view_model/repository/data_classes/meal**Beschreibung:** Menge aller Additive. Eine Auflistung aller Additive ist unter Datenbankschema Client zu finden.**3.1.13 Paket error-handling**

Abbildung 3.24: Klassendiagramm für die Klassen des Error Handlings, das für die Kommunikation zwischen den Schichten genutzt wird.

Result<S> sealed**Typ:** class**Paket:** view-model/repository/error-handling**Beschreibung:** Durch den Result-Typen lässt sich entweder ein Success oder ein Failure zurückgeben. Was genau zurückgegeben wird, lässt sich durch einen Switch-Ausdruck feststellen.**Methoden:**

```
+ {abstract} Result() <<create>>
```

Konstruktor für Result

Success<S>

Typ: class extends Result

Paket: view-model/repository/error-handling

Beschreibung: Diese Klasse ist der Rückgabety, wenn kein Fehler auftritt und ein Result im Rückgabewert enthalten ist.

Attribute:

```
+ value: S
```

Rückgabewert bei Erfolg der Funktion

Methoden:

```
+ Success(value: S) <<create>>
```

Konstruktor, der value auf den übergebenen Wert setzt.

Failure<S>

Typ: class extends Result

Paket: view-model/repository/error-handling

Beschreibung: Diese Klasse ist der Rückgabety, wenn ein Fehler auftritt und ein Result im Rückgabewert enthalten ist.

Attribute:

```
+ exception: Exception
```

Exception, die bei Misserfolg auftritt.

Methoden:

```
+ Failure(exception: Exception) <<create>>
```

Konstruktor, der exception auf die übergebene Exception setzt.

ClosedCanteenException

Typ: class extends Exception

Paket: view-model/repository/error-handling

Beschreibung: Diese Exception tritt auf, wenn die Mensa geschlossen ist.

NoDataException

Typ: class extends Exception

Paket: view-model/repository/error-handling

Beschreibung: Diese Exception tritt auf, wenn für das gewählte Datum keine Daten vorhanden sind, da das Datum vor den Aufzeichnungen der Speisepläne oder mehr als vier Wochen in der Zukunft liegt.

FilteredMealException

Typ: class extends Exception

Paket: view-model/repository/error-handling

Beschreibung: Diese Exception tritt auf, wenn kein Gericht des Speiseplans den ausgewählten Filtern entspricht.

NoConnectionException

Typ: class extends Exception

Paket: view-model/repository/error-handling

Beschreibung: Diese Exception tritt auf, wenn lokal für ein Datum keine Daten gespeichert sind und keine Verbindung zum Server aufgebaut werden kann.

NoMealException

Typ: class extends Exception

Paket: view-model/repository/error-handling

Beschreibung: Diese Exception tritt auf, wenn ein Gericht nicht in der Datenbank vorhanden ist.

3.1.14 Paket interface - Interfaces zum Model

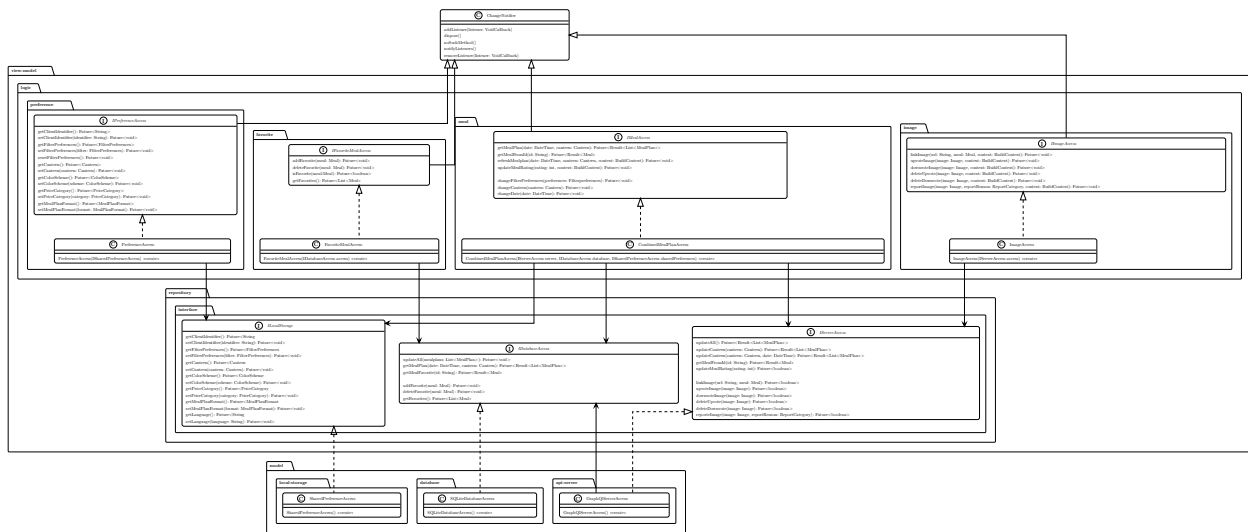


Abbildung 3.25: Klassendiagramm der Interfaces und deren Implementierungen im View-Model und Model

IDatabaseAccess

Typ: interface

Paket: view-model/repository/interface

Beschreibung: Dieses Interface ermöglicht das Ändern von Daten in der Datenbank und das Laden von Daten aus der Datenbank.

Methoden:

- + `updateAll(mealplans: List<MealPlan>): Future<void>`
Die gesamte Datenbank wird mit den übergebenen Speiseplänen aktualisiert.
- + `getMealPlan(date: DateTime, canteen: Canteen): Future<Result<List<MealPlan>>>`
Eine Liste der Speisepläne für die verschiedenen Linien für einen Tag in einer Mensa werden zurückgegeben. Falls diese nicht vorhanden sind, wird eine entsprechende Exception zurückgegeben.
- + `getMealFavorite(id: String): Future<Result<Meal>>`
Gibt die Details des Gerichts mit dem übergebenen Identifikator zurück, falls vorhanden. Ansonsten wird die `NoMealException` zurückgegeben
- + `addFavorite(Meal meal): Future<void>`
Das übergebene Gericht wird zu den Favoriten in der Datenbank hinzugefügt.
- + `deleteFavorite(Meal meal): Future<void>`
Das übergebene Gericht wird aus den Favoriten in der Datenbank entfernt.
- + `getFavorites(): Future<List<Meal>>`
Eine Liste aller vom Nutzer favorisierten Gerichte aus der Datenbank wird zurückgegeben.

ILocalStorage**Typ:** interface**Paket:** view-model/repository/interface

Beschreibung: Dieses Interface ermöglicht das Speichern und Laden von Daten mithilfe lokaler Speichermöglichkeiten außerhalb einer Datenbank. Ist ein Key, der abgefragt wird, nicht gespeichert, so wird ein Standardwert zurückgegeben.

Methoden:

- + getClientIdentifier(): Future<String>
Der Geräteidentifikator wird zurückgegeben.
- + setClientIdentifier(identifizier: String): Future<void>
Der übergebene Identifikator wird als Geräteidentifikator gespeichert.
- + getFilterPreferences(): Future<FilterPreference>
Gibt die vom Nutzer gespeicherten Filtereinstellungen zurück.
- + setFilterPreferences(FilterPreference filter): Future<void>
Die übergebene Filterkonfiguration wird gespeichert.
- + getCanteen(): Future<Canteen>
Die zuletzt angezeigte Mensa wird zurückgegeben.
- + setCanteen(Canteen canteen): Future<void>
Die übergebene Mensa wird gespeichert.
- + getColorScheme(): Future<ColorScheme>
Das gespeicherte Farbschema wird zurückgegeben.
- + setColorScheme(ColorScheme scheme): Future<void>
Das übergebene Farbschema wird gespeichert.
- + getPriceCategory(): Future<PriceCategory>
Die eingestellte Preisklasse wird zurückgegeben.
- + setPriceCategory(PriceCategory category): Future<void>
Die übergebene Preisklasse wird gespeichert.
- + getMealPlanFormat(): Future<MealPlanFormat>
Die zuletzt Darstellungsform der Speiseplanansicht wird zurückgegeben.
- + setMealPlanFormat(MealPlanFormat format): Future<void>
Die übergebene Darstellungsform wird gespeichert.
- + getLanguage(): Future<String>
Die gespeicherte Sprache wird übergeben.
- + setLanguage(language: String): Future<void>
Die übergebene Sprache wird gespeichert.

IServerAccess**Typ:** interface**Paket:** view-model/repository/interface

Beschreibung: Dieses Interface ist eine interne Schnittstelle zur Kommunikation mit dem Server zum Datenaustausch.

Methoden:

- + `updateAll(): Future<Result<List<MealPlan>>>`
Die Speisepläne aller Mensen werden aktualisiert und in der Datenbank gespeichert. Falls dies nicht möglich ist, so wird `false` zurückgegeben.
- + `updateCanteen(canteen: Canteen, date: DateTime): Future<Result<List<MealPlan>>>`
Eine Liste der verschiedenen Speisepläne der Linien wird zurückgegeben oder einen Fehler, falls diese nicht vorhanden sind bzw. nicht geladen werden konnten.
- + `getMealFromId(id: String): Future<Result<Meal>>`
Das Gericht mit dem übergebenen ID wird zurückgegeben oder ein Fehler, falls dieses nicht vorhanden ist oder nicht geladen werden konnte.
- + `updateMealRating(rating: int): Future<boolean>`
Die Bewertung eines Nutzers zu einem bestimmten Gericht wird mit dem Server synchronisiert. Falls dies nicht möglich ist, so wird `false` zurückgegeben.
- + `linkImage(url:String, meal: Meal): Future<boolean>`
Die übergebene URL wird serverseitig mit dem übergebenen Bild verknüpft. Falls dies nicht möglich ist, so wird `false` zurückgegeben.
- + `upvoteImage(image: Image): Future<boolean>`
Die Bewertung des übergebenen Bildes wird dem Server übermittelt. Falls dies nicht möglich ist, so wird `false` zurückgegeben.
- + `downvoteImage(image: Image): Future<boolean>`
Die Bewertung des übergebenen Bildes wird dem Server übermittelt. Falls dies nicht möglich ist, so wird `false` zurückgegeben.
- + `deleteUpvote(image: Image): Future<boolean>`
Die Entfernung der positiven Bewertung des übergebenen Bildes wird dem Server übermittelt. Falls dies nicht möglich ist, so wird `false` zurückgegeben.
- + `deleteDownvote(image: Image): Future<boolean>`
Die Entfernung der negativen Bewertung des übergebenen Bildes wird dem Server übermittelt. Falls dies nicht möglich ist, so wird `false` zurückgegeben.
- + `reportImage(image: Image, reportReason: ReportCategory): Future<boolean>`
Die Meldung des übergebenen Bildes wird dem Server übermittelt. Falls dies nicht möglich ist, so wird `false` zurückgegeben.

3.1.15 Paket `model`

In diesem Paket werden die Interfaces, die im View-Model gespeichert sind, implementiert.

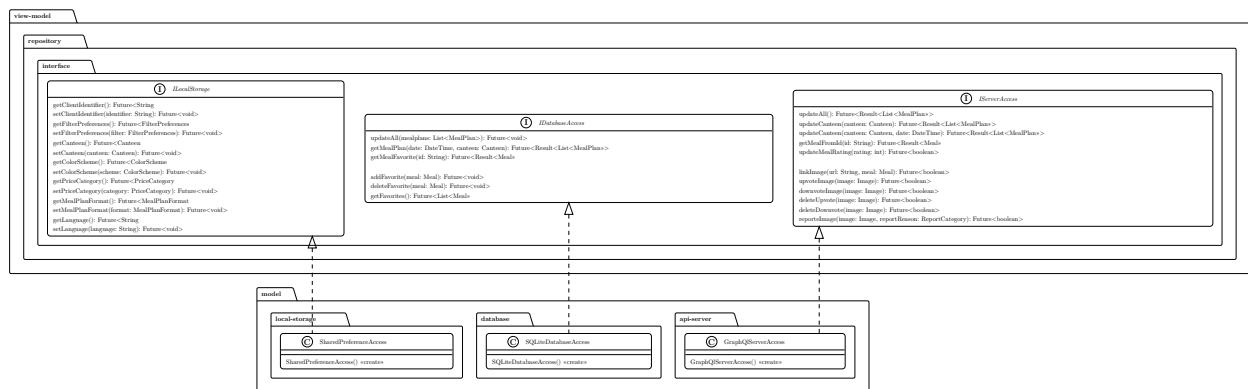


Abbildung 3.26: Klassendiagramm der Implementierungen der Interfaces des View-Models

GraphQLServerAccess

Typ: class implements IServerAccess

Paket: model/api-server

Beschreibung: Diese Klasse ist für die Kommunikation mit dem Server durch GraphQL verantwortlich.

Methoden:

```
+ factory GraphQLServerAccess() <<create>>
```

Dieser Konstruktor gibt eine Instanz der Klasse zurück.

SQLiteDatabaseAccess

Typ: class implements IDatabaseAccess

Paket: model/database

Beschreibung: Diese Klasse ist für den Zugriff auf die SQLite-Datenbank des Clients verantwortlich.

Methoden:

```
+ factory SQLiteDatabaseAccess() <<create>>
```

Dieser Konstruktor gibt eine Instanz der Klasse zurück.

SharedPreferencesAccess

Typ: class implements ILocalStorage

Paket: model/local-storage

Beschreibung: Diese Klasse ist für den Zugriff auf Shared Preferences des Clients verantwortlich.

Methoden:

+ factory SharedPreferenceAccess() <<create>>

Dieser Konstruktor gibt eine Instanz der Klasse zurück.

3.2 Feinentwurf Backend

3.2.1 Paket startup

Hier wird das Paket definiert, das den Einstiegspunkt für den Server bereitstellt und die in 2.3 Grobentwurf Backend beschriebene Komponentenstruktur erzeugt.

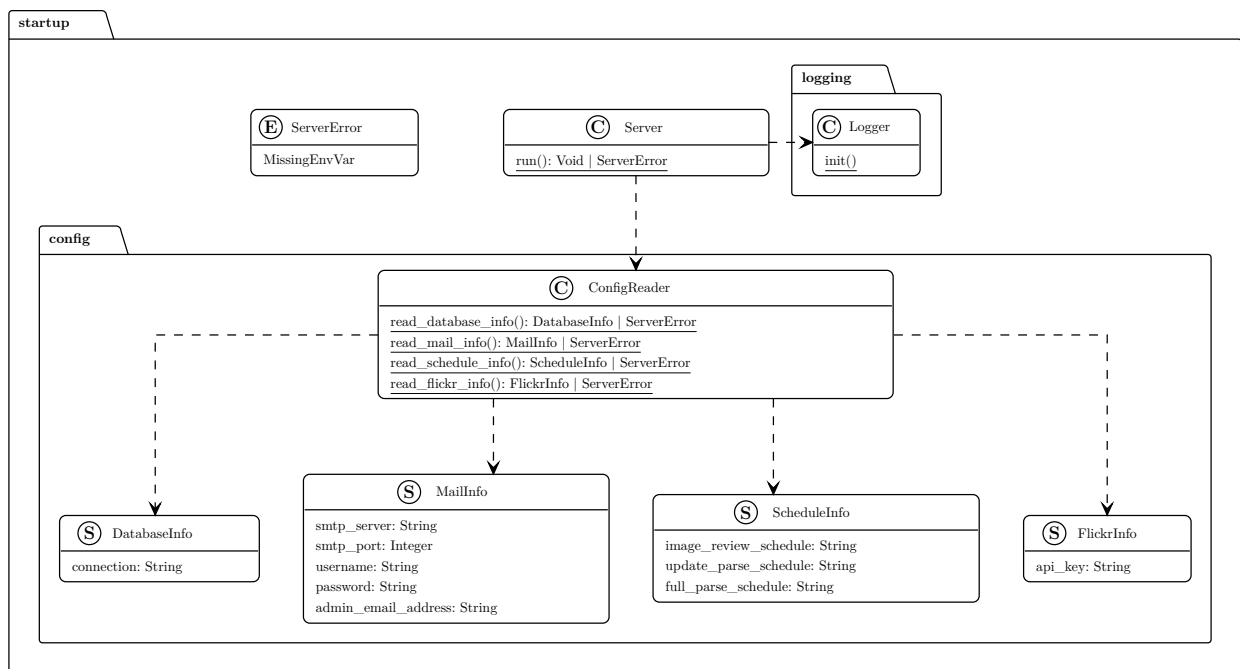


Abbildung 3.27: Klassendiagramm des startup-Pakets

Server

Typ: class

Paket: startup

Beschreibung: Klasse, die die gesammelte Serverfunktion nach außen anbietet.

Methoden:

+ {static} run(): Void | ServerError

Führt den Server und alles, was dazugehört aus. Dabei werden zunächst Konfigurationen eingelesen und dann die Komponentenstruktur aufgebaut.

Logger

Typ: class

Paket: startup

Beschreibung: Klasse zum Initialisieren des der Protokollierungsfunktion.

Methoden:

+ {static} init()

Initialisiert die Protokollierungsfunktionen. Protokollierungen passieren in jeder Komponente über einen globalen Kontext, sodass diese in den weiteren Klassenbeschreibungen nicht sichtbar sind.

ConfigReader

Typ: class

Paket: startup/config

Beschreibung: Klasse zum Einlesen von Konfigurationen aus Umgebungsvariablen.

Methoden:

+ {static} read_database_info(): DatabaseInfo | ServerError

Liest die Konfiguration für den Zugang zur Datenbank aus den Umgebungsvariablen und gibt diese zurück. Falls notwendige Konfigurationen nicht vorhanden sind, wird ein Fehler zurückgegeben.

+ {static} read_mail_info(): MailInfo | ServerError

Liest die Konfiguration für den Zugang zum Mailserver aus den Umgebungsvariablen und gibt diese zurück. Falls notwendige Konfigurationen nicht vorhanden sind, wird ein Fehler zurückgegeben.

+ {static} read_schedule_info(): ScheduleInfo | ServerError

Liest die Konfiguration für den Scheduler aus den Umgebungsvariablen und gibt diese zurück. Falls notwendige Konfigurationen nicht vorhanden sind, wird ein Fehler zurückgegeben.

+ {static} read_flickr_info(): FlickrInfo | ServerError

Liest die Konfiguration für den Zugang zur Flickr-API aus den Umgebungsvariablen und gibt diese zurück. Falls notwendige Konfigurationen nicht vorhanden sind, wird ein Fehler zurückgegeben.

DatabaseInfo

Typ: struct

Paket: startup/config

Beschreibung: Struktur mit allen Informationen für eine Verbindung zur Datenbank.

Attribute:

+ connection: String

Zeichenkette mit Verbindungsinformationen zur Datenbank

MailInfo**Typ:** struct**Paket:** startup/config**Beschreibung:** Struktur mit allen Informationen zum Verbinden mit einem Mainserver und senden von Emails an Administratoren.**Attribute:**

- + smtp_server: String
Domänenname für die Verbindung zu einem Mail-Server über das SMTP-Protokoll.
- + smtp_port: Integer
Port, auf welchem der in `smtp_server` angegebene Mail-Server auf SMTP-Anfragen hört.
- + username: String
Benutzername zur Verbindung mit dem Mail-Server.
- + password: String
Passwort zur Verbindung mit dem Mail-Server.
- + admin_email_address: String
E-Mail-Adresse eines Administrators, der bei Meldeanträgen benachrichtigt wird.

ScheduleInfo**Typ:** struct**Paket:** startup/config**Beschreibung:** Struktur mit allen Informationen für regelmäßig auszuführende Aktionen.**Attribute:**

- + image_review_schedule: String
Cron¹ Zeitplan für Bildüberprüfungen.
- + update_parse_schedule: String
Cron Zeitplan für Aktualisierung der heutigen Speisepläne.
- + full_parse_schedule: String
Cron Zeitplan für Aktualisierung aller Speisepläne.

FlickrInfo**Typ:** struct**Paket:** startup/config**Beschreibung:** Struktur mit allen Informationen zum Zugriff auf die Flickr-API.**Attribute:**

- + api_key: String
API-Schlüssel für die Flickr-API.

¹<https://cron.help/>

3.2.2 Komponente Scheduler

Diese Komponente löst regelmäßige Ereignisse wie z.B. das Abfragen des aktuellen Speiseplans aus.

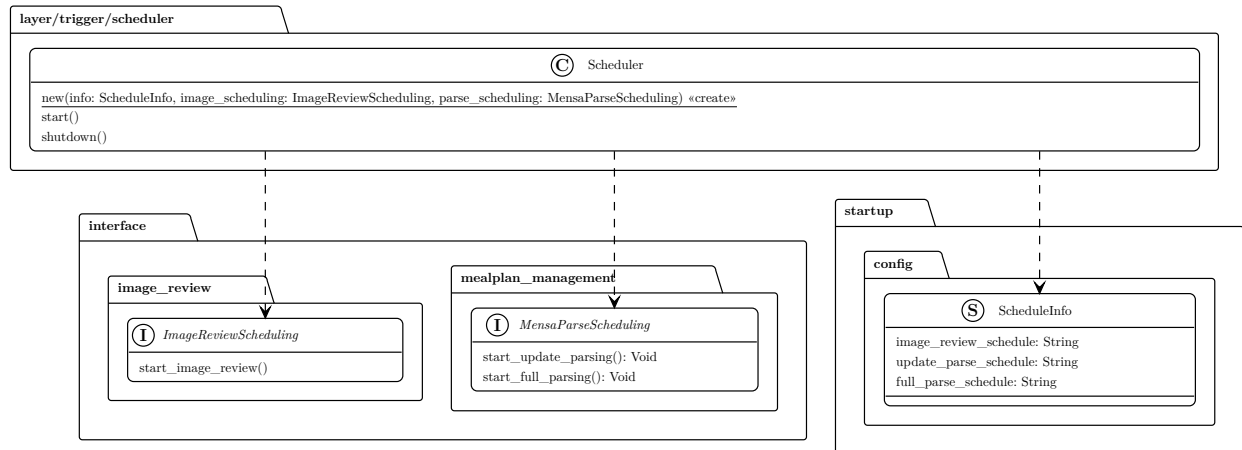


Abbildung 3.29: Klassendiagramm der Scheduling-Komponente

Scheduler

Typ: class

Paket: layer/trigger/scheduler

Beschreibung: Klasse zum Planen von regelmäßigen Ereignissen.

Methoden:

- + `new(info: ScheduleInfo, image_scheduling: ImageReviewScheduling, parse_scheduling: MensaParseScheduling) <<create>>`
Erzeugt einen neuen Scheduler mit dem in `info` angegebenen Zeitplan und den in `scheduling` angegebenen Aktionen.
- + `start()`
Startet den Scheduler. Dieser läuft im Hintergrund, bis er mit `shutdown()` gestoppt wird.
- + `shutdown()`
Stoppt den Scheduler.

Sequenzen der Scheduling-Komponente

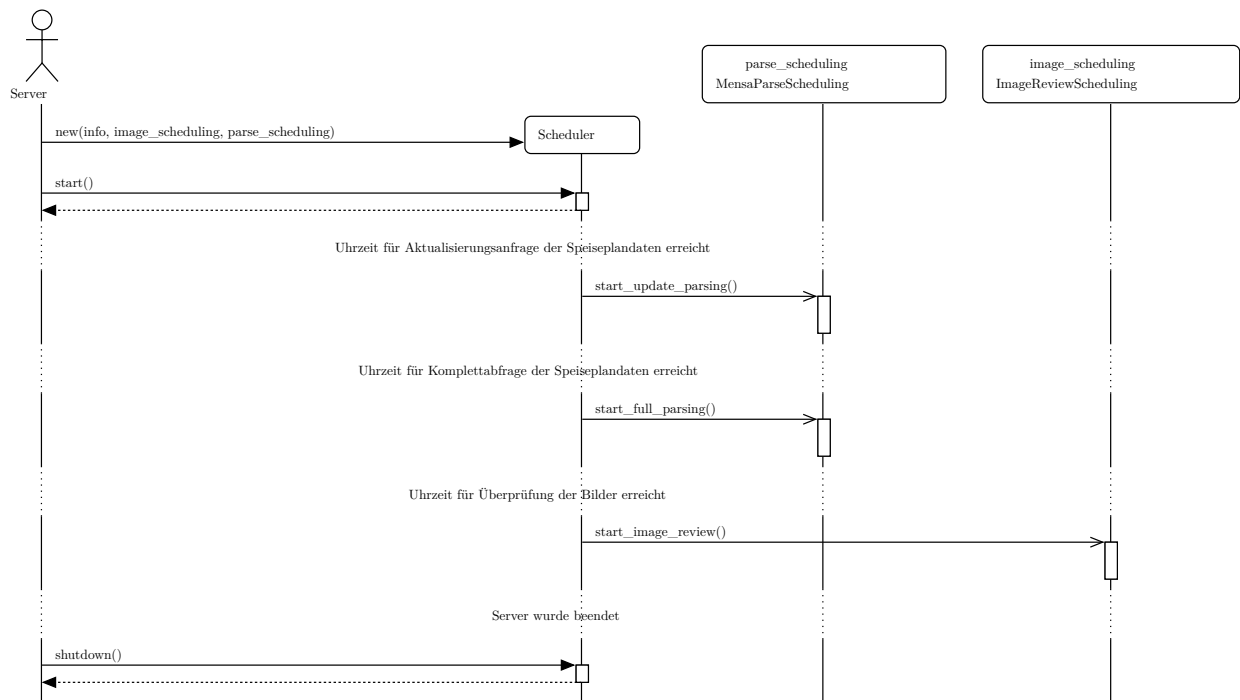


Abbildung 3.30: Sequenzdiagramm der Scheduling-Komponente

Tests der Scheduling-Komponente

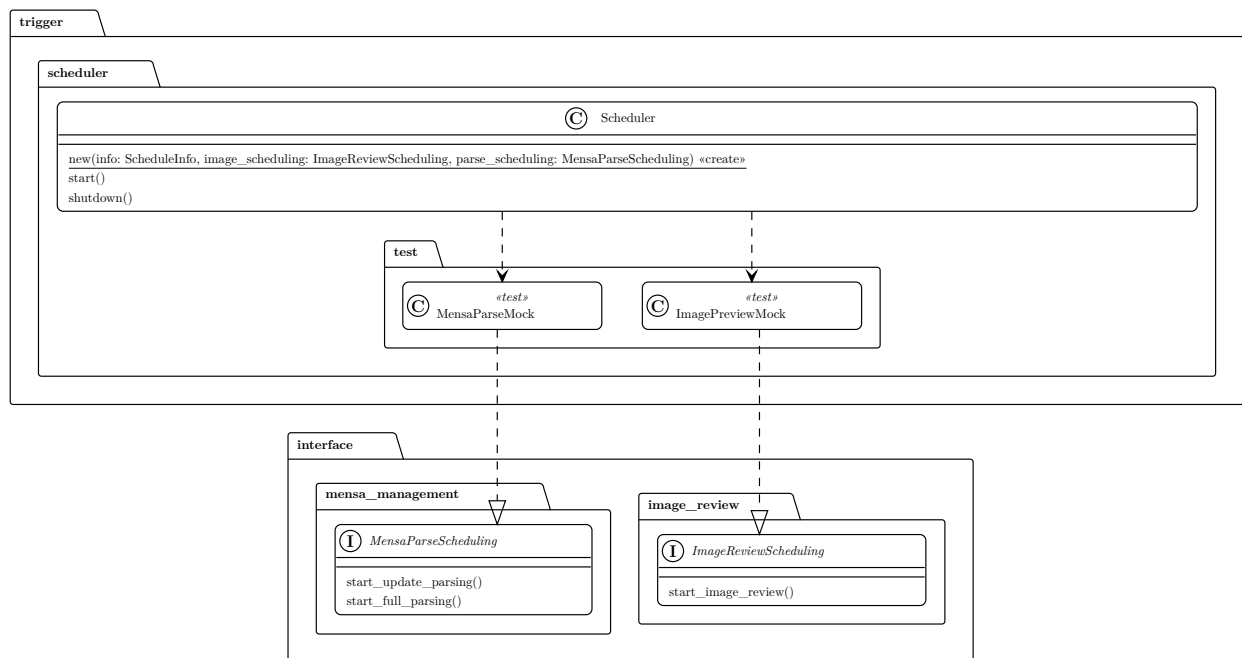


Abbildung 3.31: Klassendiagramm der Test-Klassen der Scheduling-Komponente

MensaParseMock

Typ: class implements MensaParseScheduling

Paket: layer/trigger/scheduler/test

Beschreibung: Diese Testklasse implementiert MensaParseScheduling und simuliert diese Schnittstelle, um die überliegende Komponente isoliert testen zu können.

ImagePreviewMock

Typ: class implements ImageReviewScheduling

Paket: layer/trigger/scheduler/test

Beschreibung: Diese Testklasse implementiert ImageReviewScheduling und simuliert diese Schnittstelle, um die überliegende Komponente isoliert testen zu können.

3.2.3 Komponente GraphQL

Diese Komponente enthält den Webserver, der API-Anfragen ermöglicht und stellt den Einstiegspunkt für diese dar.

Im folgenden Klassendiagramm werden ebenfalls Attribute modelliert, die durch die GraphQL-Library für API-Anfragen verfügbar sind. Für die genauen Beschreibungen der GraphQL-Attribute und Funktionen siehe 4.3 API-Schema.

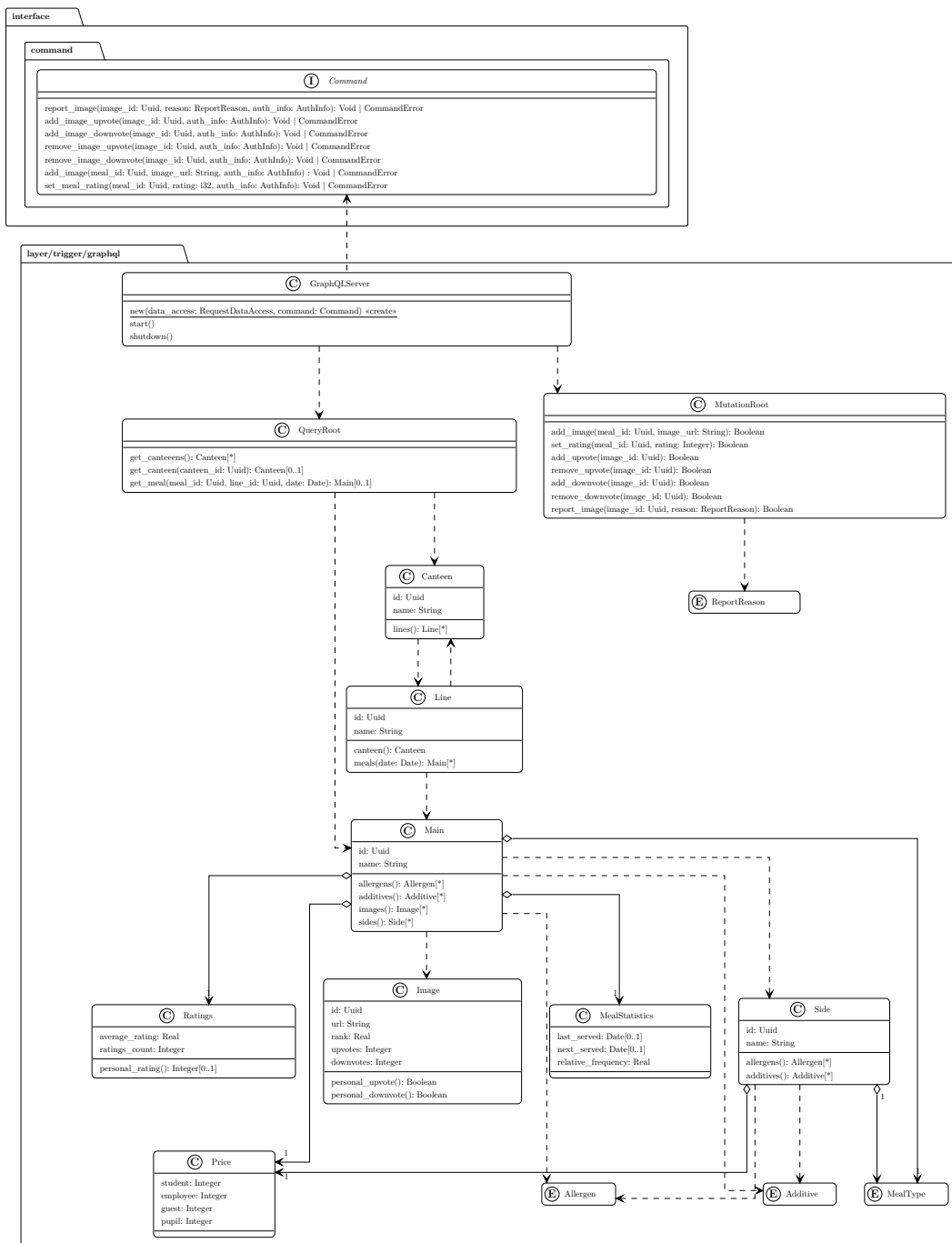


Abbildung 3.32: Klassendiagramm der GraphQL-Komponente

GraphQLServer

Typ: class

Paket: layer/trigger/graphql

Beschreibung: Eine Klasse, die den Webserver für GraphQL-Anfragen steuert.

Methoden:

- + {static} new(data_access: RequestDataAccess, command: Command)
Erzeugt ein neues Objekt mit den übergebenen Zugängen zum Datenspeicher und der Logik für Befehle
- + start()
Startet den GraphQL-Server. Dieser läuft im Hintergrund, bis er mit shutdown() gestoppt wird.
- + shutdown()
Beendet den GraphQL-Server.

QueryRoot

Typ: class

Paket: layer/trigger/graphql

Beschreibung: Eine Klasse, die die GraphQL-Root-Querys implementiert.

Methoden:

- + get_canteens(): Canteen[*]
Behandlungsmethode für die getCanteens GraphQL-Anfrage.
- + get_canteen(canteen_id: Uuid): Canteen[0..1]
Behandlungsmethode für die getCanteen GraphQL-Anfrage.
- + get_meal(meal_id: Uuid, line_id: Uuid, date: Date): Meal[0..1]
Behandlungsmethode für die getMeal GraphQL-Anfrage.

MutationRoot

Typ: class

Paket: layer/trigger/graphql

Beschreibung: Eine Klasse, die die GraphQL-Root-Mutations implementiert.

Methoden:

- + add_image(meal_id: Uuid, image_url: String): Boolean
Behandlungsmethode für die addImage GraphQL-Anfrage.
- + set_rating(meal_id: Uuid, rating: Integer): Boolean
Behandlungsmethode für die setRating GraphQL-Anfrage.
- + add_upvote(image_id: Uuid): Boolean
Behandlungsmethode für die addUpvote GraphQL-Anfrage.

- + `remove_upvote(image_id: Uuid): Boolean`
Behandlungsmethode für die `removeUpvote` GraphQL-Anfrage.
- + `add_downvote(image_id: Uuid): Boolean`
Behandlungsmethode für die `addDownvote` GraphQL-Anfrage.
- + `remove_downvote(image_id: Uuid): Boolean`
Behandlungsmethode für die `removeDownvote` GraphQL-Anfrage.
- + `report_image(image_id: Uuid, reason: ReportReason): Boolean`
Behandlungsmethode für die `reportImage` GraphQL-Anfrage.

Canteen

Typ: class

Paket: layer/trigger/graphql

Beschreibung: Eine Klasse, die eine Mensa in GraphQL repräsentiert.

Attribute:

- + `id: Uuid`
Attribut, auf welches über die GraphQL-API mit `id` zugegriffen werden kann.
- + `name: String`
Attribut, auf welches über die GraphQL-API mit `name` zugegriffen werden kann.

Methoden:

- + `lines(): Line[*]`
Bereitstellungsmethode für das `lines` GraphQL-Attribut.

Line

Typ: class

Paket: layer/trigger/graphql

Beschreibung: Eine Klasse, die eine Linie in GraphQL repräsentiert.

Attribute:

- + `id: Uuid`
Attribut, auf welches über die GraphQL-API mit `id` zugegriffen werden kann.
- + `name: String`
Attribut, auf welches über die GraphQL-API mit `name` zugegriffen werden kann.

Methoden:

- + `canteen(): Canteen`
Bereitstellungsmethode für das `canteen` GraphQL-Attribut.
- + `meals(date: Date): Meal[*]`
Behandlungsmethode für die `meals()` GraphQL-Anfrage.

Main**Typ:** class**Paket:** layer/trigger/graphql**Beschreibung:** Eine Klasse, die ein Hauptgericht in GraphQL repräsentiert.**Attribute:**

- + **id:** Uuid
Attribut, auf welches über die GraphQL-API mit **id** zugegriffen werden kann.
- + **name:** String
Attribut, auf welches über die GraphQL-API mit **name** zugegriffen werden kann.
- + **price:** Price
Attribut, auf welches über die GraphQL-API mit **price** zugegriffen werden kann.
- + **meal_type:** MealType
Attribut, auf welches über die GraphQL-API mit **mealType** zugegriffen werden kann.

Methoden:

- + **statistics():** MealStatistics
Bereitstellungsmethode für das **statistics** GraphQL-Attribut.
- + **allergens():** Allergen[*]
Bereitstellungsmethode für das **allergens** GraphQL-Attribut.
- + **additives():** Additive[*]
Bereitstellungsmethode für das **additives** GraphQL-Attribut.
- + **ratings():** Ratings
Bereitstellungsmethode für das **ratings** GraphQL-Attribut.
- + **images():** Image[*]
Bereitstellungsmethode für das **images** GraphQL-Attribut.
- + **sides():** Side[*]
Bereitstellungsmethode für das **sides** GraphQL-Attribut.

Image**Typ:** class**Paket:** layer/trigger/graphql**Beschreibung:** Eine Klasse, die eine Bild in GraphQL repräsentiert.**Attribute:**

- + **id:** Uuid
Attribut, auf welches über die GraphQL-API mit **id** zugegriffen werden kann.
- + **url:** String
Attribut, auf welches über die GraphQL-API mit **url** zugegriffen werden kann.
- + **rank:** Real
Attribut, auf welches über die GraphQL-API mit **rank** zugegriffen werden kann.

+ `upvotes: Integer`

Attribut, auf welches über die GraphQL-API mit `upvotes` zugegriffen werden kann.

+ `downvotes: Integer`

Attribut, auf welches über die GraphQL-API mit `downvotes` zugegriffen werden kann.

Methoden:

+ `personal_upvote(): Boolean`

Bereitstellungsmethode für das `personalUpvote` GraphQL-Attribut.

+ `personal_downvote(): Boolean`

Bereitstellungsmethode für das `personalDownvote` GraphQL-Attribut.

Ratings

Typ: class

Paket: layer/trigger/graphql

Beschreibung: Eine Klasse, die eine die Übersicht zu Bewertungen eines Gerichts in GraphQL repräsentiert.

Attribute:

+ `average_rating: Real`

Attribut, auf welches über die GraphQL-API mit `averageRating` zugegriffen werden kann.

+ `ratings_count: Integer`

Attribut, auf welches über die GraphQL-API mit `ratingsCount` zugegriffen werden kann.

Methoden:

+ `personal_rating(): Integer[0..1]`

Bereitstellungsmethode für das `personalRating` GraphQL-Attribut.

Price

Typ: class

Paket: layer/trigger/graphql

Beschreibung: Eine Klasse, die die Preise für ein Gericht in GraphQL repräsentiert.

Attribute:

+ `student: Integer`

Attribut, auf welches über die GraphQL-API mit `student` zugegriffen werden kann.

+ `employee: Integer`

Attribut, auf welches über die GraphQL-API mit `employee` zugegriffen werden kann.

+ `guest: Integer`

Attribut, auf welches über die GraphQL-API mit `guest` zugegriffen werden kann.

+ `pupil`: `Integer`

Attribut, auf welches über die GraphQL-API mit `pupil` zugegriffen werden kann.

MealStatistics

Typ: `class`

Paket: `layer/trigger/graphql`

Beschreibung: Eine Klasse, die Statistiken zu einem Gericht in GraphQL repräsentiert.

Attribute:

+ `last_served`: `Date[0..1]`

Attribut, auf welches über die GraphQL-API mit `lastServed` zugegriffen werden kann.

+ `next_served`: `Date[0..1]`

Attribut, auf welches über die GraphQL-API mit `nextServed` zugegriffen werden kann.

+ `relative_frequency`: `Real`

Attribut, auf welches über die GraphQL-API mit `relativeFrequency` zugegriffen werden kann.

Side

Typ: `class`

Paket: `layer/trigger/graphql`

Beschreibung: Eine Klasse, die eine Beilage zu einem Gericht in GraphQL repräsentiert.

Attribute:

+ `id`: `Uuid`

Attribut, auf welches über die GraphQL-API mit `id` zugegriffen werden kann.

+ `name`: `String`

Attribut, auf welches über die GraphQL-API mit `name` zugegriffen werden kann.

+ `price`: `Price`

Attribut, auf welches über die GraphQL-API mit `price` zugegriffen werden kann.

+ `meal_type`: `MealType`

Attribut, auf welches über die GraphQL-API mit `mealType` zugegriffen werden kann.

Methoden:

+ `allergens()`: `Allergen[*]`

Bereitstellungsmethode für das `allergens` GraphQL-Attribut.

+ `additives()`: `Additive[*]`

Bereitstellungsmethode für das `additives` GraphQL-Attribut.

Allergen

Typ: enum

Paket: layer/trigger/graphql

Beschreibung: Enumerationstyp, der Allergene in GraphQL repräsentiert. Eine ausführliche Auflistung der Varianten befindet sich unter 4.1.4 Entitäten. Dieser Enumerationstyp muss aus Gründen der Unabhängigkeit der einzelnen Schichten und der Funktionsweise der GraphQL-API hier erneut definiert werden, obwohl ein gleicher Typ schon im Paket `util` existiert.

Additive

Typ: enum

Paket: layer/trigger/graphql

Beschreibung: Enumerationstyp, der Zusatzstoffe in GraphQL repräsentiert. Eine ausführliche Auflistung der Varianten befindet sich unter 4.1.4 Entitäten. Dieser Enumerationstyp muss aus Gründen der Unabhängigkeit der einzelnen Schichten und der Funktionsweise der GraphQL-API hier erneut definiert werden, obwohl ein gleicher Typ schon im Paket `util` existiert.

MealType

Typ: enum

Paket: layer/trigger/graphql

Beschreibung: Enumerationstyp, der den Typ eines Gerichts repräsentiert. Eine ausführliche Auflistung der Varianten befindet sich unter 4.1.4 Entitäten. Dieser Enumerationstyp muss aus Gründen der Unabhängigkeit der einzelnen Schichten und der Funktionsweise der GraphQL-API hier erneut definiert werden, obwohl ein gleicher Typ schon im Paket `util` existiert.

ReportReason

Typ: enum

Paket: layer/trigger/graphql

Beschreibung: Enumerationstyp, der den Grund für einen Meldeantrag eines Bildes repräsentiert. Eine ausführliche Auflistung der Varianten befindet sich unter 4.1.4 Entitäten. Dieser Enumerationstyp muss aus Gründen der Unabhängigkeit der einzelnen Schichten und der Funktionsweise der GraphQL-API hier erneut definiert werden, obwohl ein gleicher Typ schon im Paket `util` existiert.

Sequenzen der GraphQL-Komponente

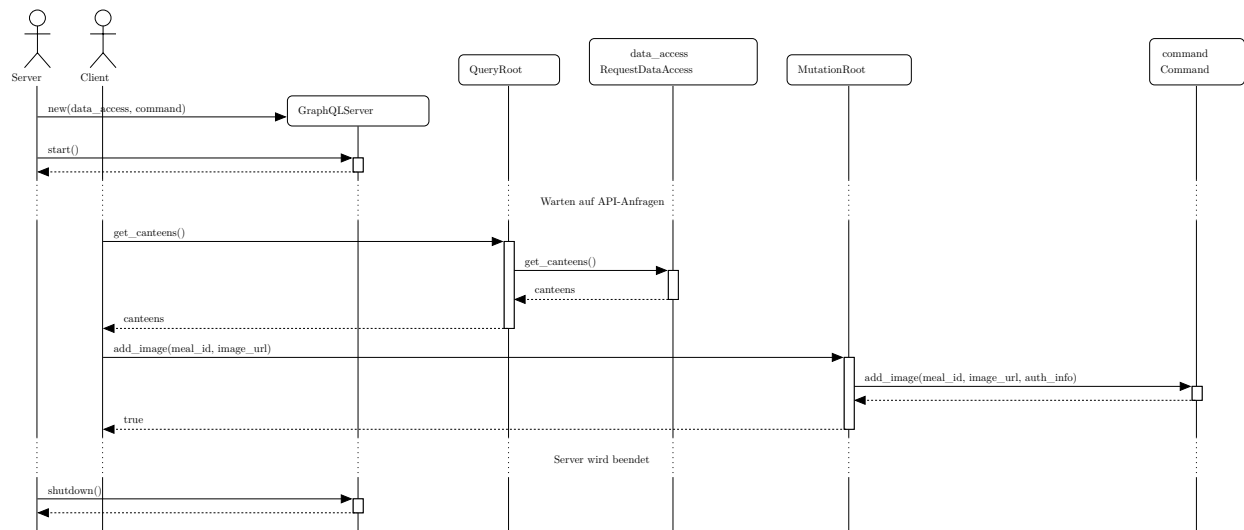


Abbildung 3.33: Sequenzdiagramm der GraphQL-Komponente

Tests der GraphQL-Komponente

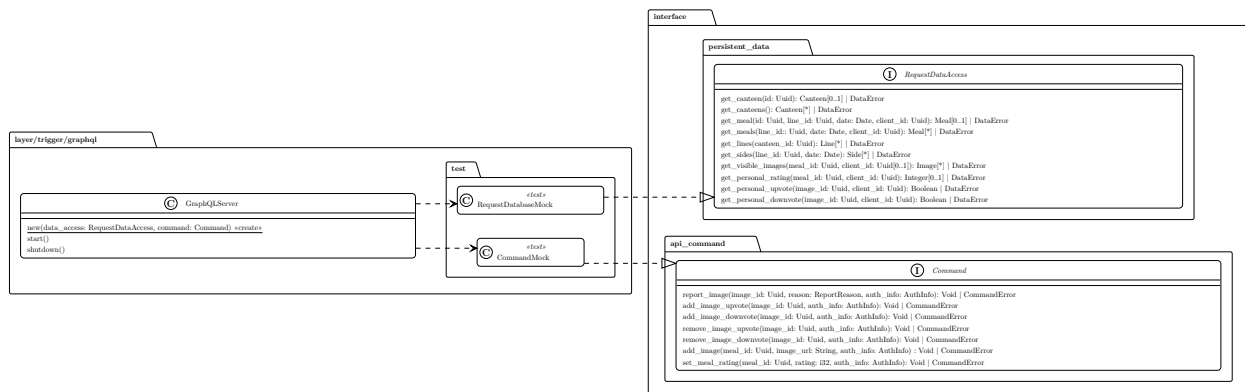


Abbildung 3.34: Klassendiagramm der GraphQL-Test-Komponente

MensaParseMock

Typ: class implements `MensaParseScheduling`

Paket: `layer/trigger/graphql/test`

Beschreibung: Diese Klasse simuliert die Schnittstelle `MensaParseScheduling`, um die überliegende Komponente isoliert testen zu können.

ImagePreviewMock

Typ: class implements `ImageReviewScheduling`

Paket: `layer/trigger/graphql/test`

Beschreibung: Diese simuliert die Schnittstelle `ImageReviewScheduling`, um die überliegende Komponente isoliert testen zu können.

3.2.4 Komponente Command

Diese Komponente enthält all die Logik, die bei API-Anfragen benötigt wird, die mehr als nur Daten abfragen.

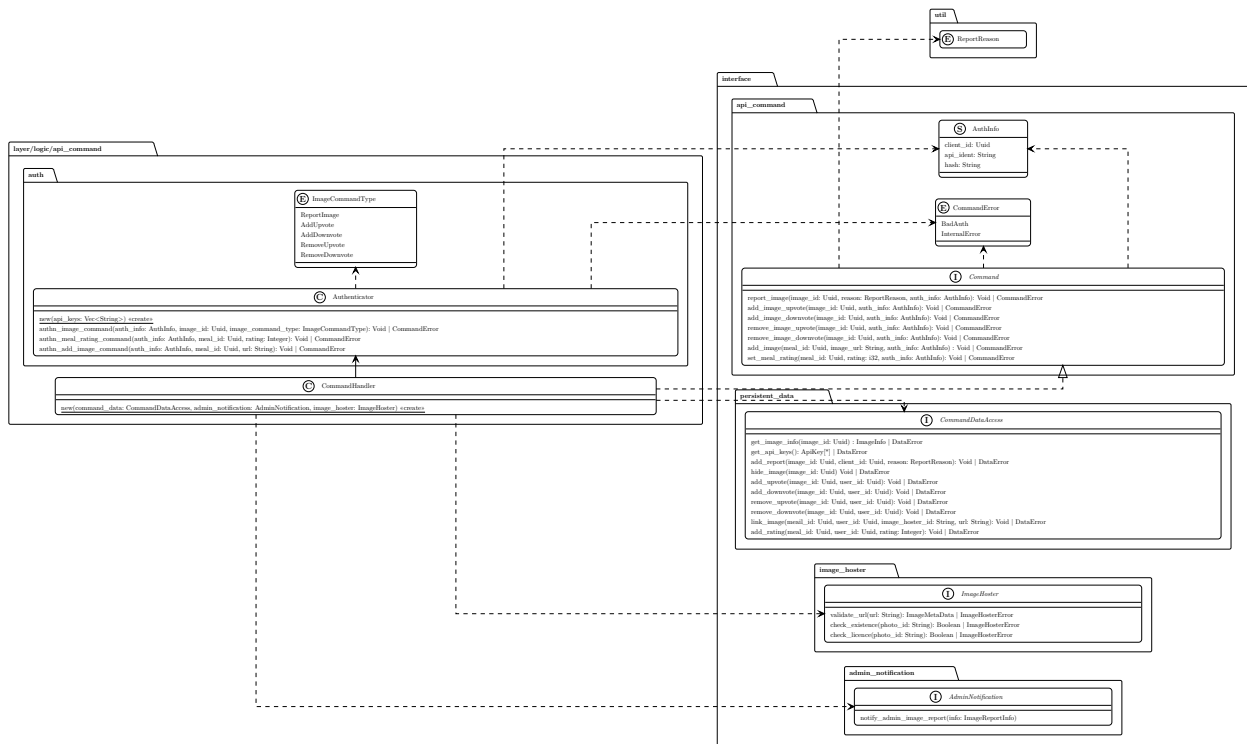


Abbildung 3.35: Klassendiagramm der Command-Komponente

Command

Typ: interface

Paket: interface/api_command

Beschreibung: Schnittstelle für den Zugriff auf Befehle, die von einer API ausgelöst werden können.

Methoden:

- + `report_image(image_id: Uuid, reason: ReportReason, auth_info: AuthInfo): Void | CommandError`
Befehl zum Melden eines Bildes. Dabei wird auch geprüft, ob das Bild schon automatisch versteckt werden soll.
- + `add_image_upvote(image_id: Uuid, auth_info: AuthInfo): Void | CommandError`
Befehl zum Aufwerten eines Bildes. Gleichzeitige Abwertungen desselben Nutzers werden entfernt.

- + `add_image_downvote(image_id: Uuid, auth_info: AuthInfo): Void | CommandError`
Befehl zu Abwerten eines Bildes. Gleichzeitige Aufwertungen desselben Nutzers werden entfernt.
- + `remove_image_upvote(image_id: Uuid, auth_info: AuthInfo): Void | CommandError`
Befehl zum Entfernen einer Aufwertung eines Bildes, falls vorhanden.
- + `remove_image_downvote(image_id: Uuid, auth_info: AuthInfo): Void | CommandError`
Befehl zum Entfernen einer Abwertung eines Bildes, falls vorhanden.
- + `add_image(meal_id: Uuid, image_url: String, auth_info: AuthInfo): Void | CommandError`
Befehl zum Verlinken eines Bildes zu einem Gericht.
- + `set_meal_rating(meal_id: Uuid, rating: Integer, auth_info: AuthInfo): Void | CommandError`
Befehl zum Bewerten eines Gerichts.

AuthInfo

Typ: struct

Paket: interface/api_command

Beschreibung: Struktur mit allen nötigen Informationen zur Authentifizierung eines Clients.

Attribute:

- + `client_id: Uuid`
Identifikator des Clients.
- + `api_ident: String`
Anfang des API-Schlüssels, um diesen zu identifizieren.
- + `hash: String`
Hash aller Attribute des Befehls einschließlich des Befehlsnamen.

CommandError

Typ: enum

Paket: interface/api_command

Beschreibung: Enumerationstyp für mögliche Fehlschläge von Befehlen.

Varianten:

- + `BadAuth`
Authentifizierungsinformationen nicht korrekt.
- + `InternalError`
Sonstiger Fehler bei z.B. Zugriff auf Ressourcen.

CommandHandler

Typ: class implements Command

Paket: layer/logic/api_command

Beschreibung: Eine Klasse, die das Ausführen von Befehlen erlaubt.

Methoden:

```
+ {static} new(command_data: CommandDataAccess, admin_notification:
  AdminNotification, image_hoster: ImageHoster) <<create>>
  Erzeugt ein neues Objekt mit den übergebenen Zugängen zu Daten, Administratorbenach-
  richtigungen und Bildhoster.
```

Authenticator

Typ: class

Paket: layer/logic/api_command/auth

Beschreibung: Eine Klasse, die für das Authentifizieren von Befehlen zuständig ist.

Methoden:

```
+ {static} new(api_keys: String[*]) <<create>>
  Erzeugt ein neues Objekt, welches Anfragen authentifiziert und dabei nur die übergebenen
  API-Schlüssel als gültig erachtet.
+ authn_image_command(auth_info: AuthInfo, image_id: Uuid, image_command_type:
  ImageCommandType): Void | CommandError
  Authentifiziert einen Bilderbefehl, indem überprüft wird, ob der Hash zum gegebenen
  API-Schlüssel korrekt ist.
+ authn_meal_rating_command(auth_info: AuthInfo, meal_id: Uuid, rating:
  Integer): Void | CommandError
  Authentifiziert den Befehl zum Bewerten von Gerichten, indem überprüft wird, ob der Hash
  zum gegebenen API-Schlüssel korrekt ist.
+ authn_add_image_command(auth_info: AuthInfo, meal_id: Uuid, url: String):
  Void | CommandError
  Authentifiziert den Befehl zum Hinzufügen eines Bildes, indem überprüft wird, ob der Hash
  zum gegebenen API-Schlüssel korrekt ist.
```

ImageCommandType

Typ: enum

Paket: layer/logic/api_command

Beschreibung: Enumerationstyp für die verschiedenen Arten von Bildbefehlen.

Varianten:

```
+ ReportImage
  Befehl zum Melden eines Bildes.
```

- + AddUpvote
Befehl zum Hinzufügen einer Aufwertung.
- + AddDownvote
Befehl zum Hinzufügen einer Abwertung.
- + RemoveUpvote
Befehl zum Entfernen einer Aufwertung.
- + RemoveDownvote
Befehl zum Entfernen einer Abwertung.

Sequenzen der Command-Komponente

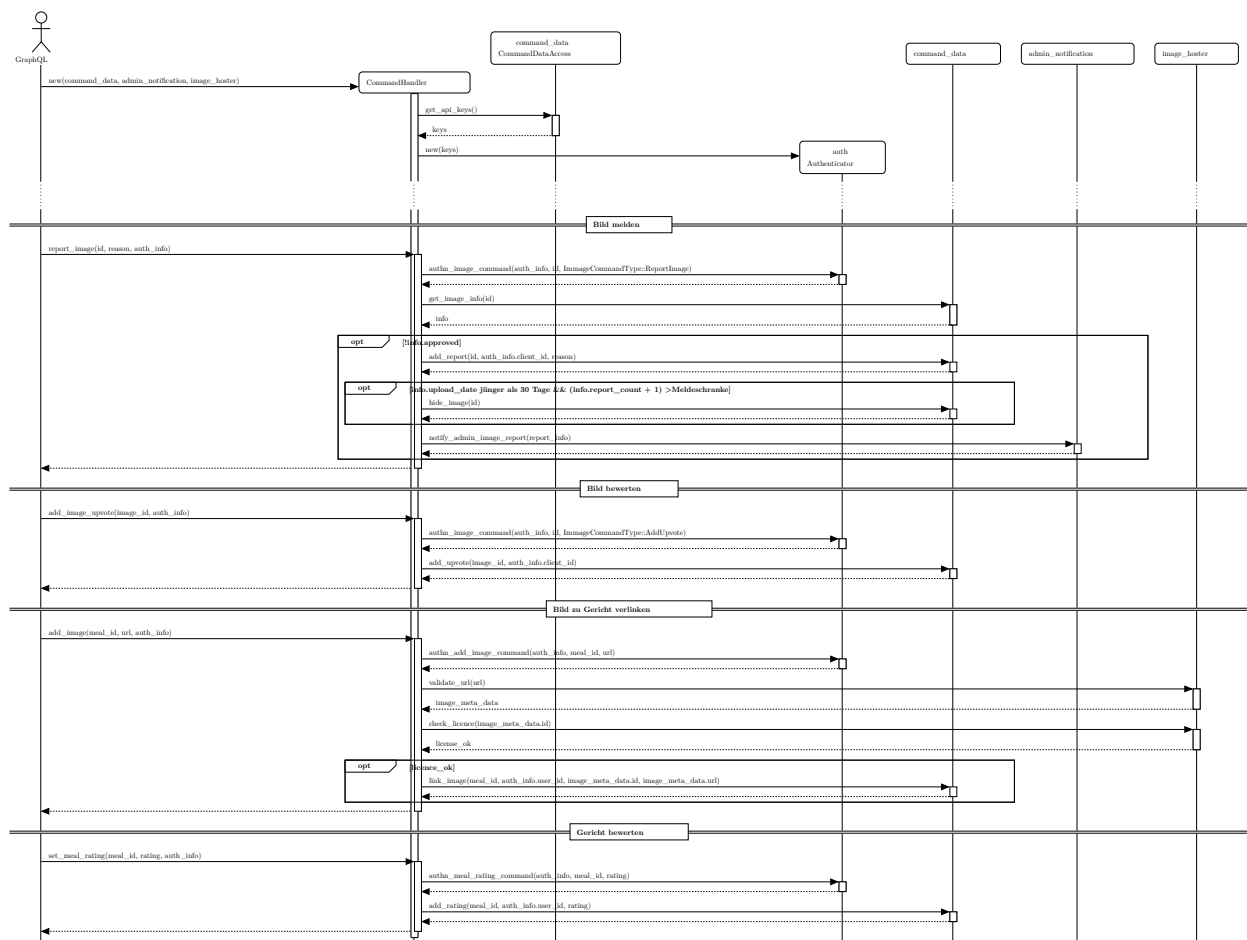


Abbildung 3.36: Sequenzdiagramm der Command-Komponente

Tests der Command-Komponente

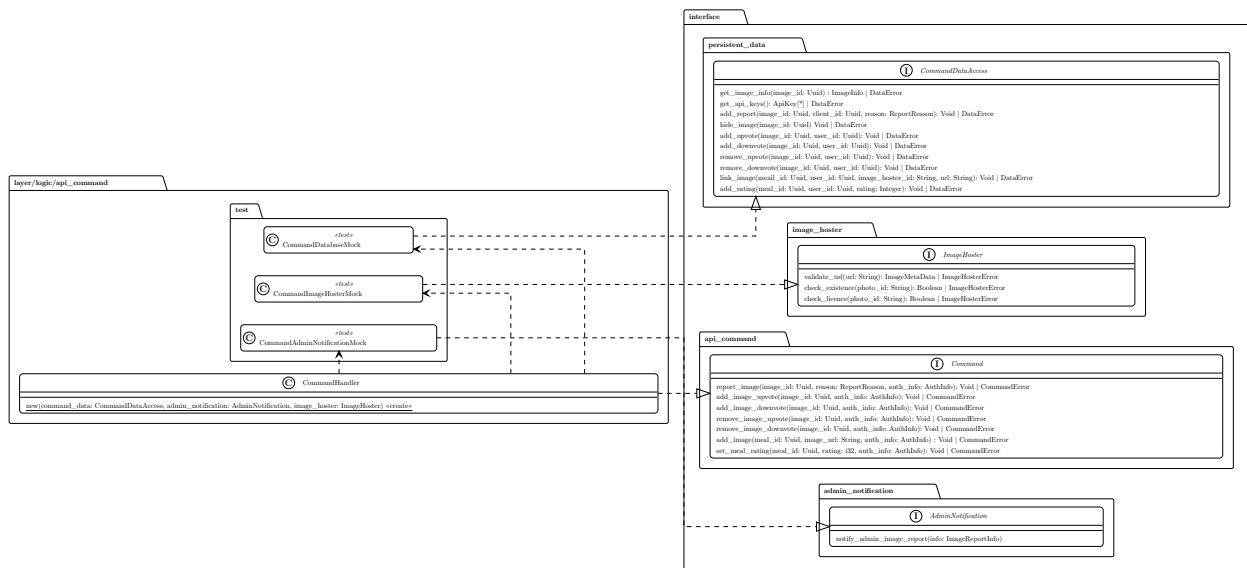


Abbildung 3.37: Klassendiagramm der Test-Klassen der Command-Komponente

CommandDatabaseMock

Typ: class implements `CommandDataAccess`

Paket: `layer/logic/api_command/test`

Beschreibung: Diese Testklasse implementiert `CommandDataAccess` und simuliert diese Schnittstelle, um die überliegende Komponente isoliert testen zu können.

CommandImageHosterMock

Typ: class implements `ImageHoster`

Paket: `layer/logic/api_command/test`

Beschreibung: Diese Testklasse implementiert `ImageHoster` und simuliert diese Schnittstelle, um die überliegende Komponente isoliert testen zu können.

CommandAdminNotificationMock

Typ: class implements `AdminNotification`

Paket: `layer/logic/api_command/test`

Beschreibung: Diese Testklasse simuliert die `AdminNotification`-Schnittstelle, um die überliegende Komponente isoliert testen zu können.

3.2.5 Komponente MealplanManagement

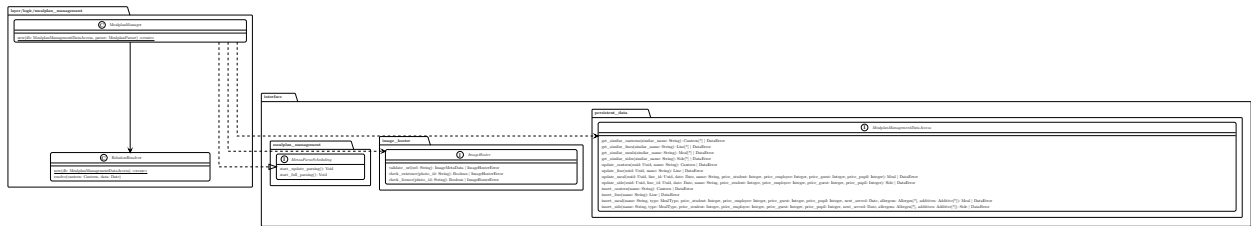


Abbildung 3.38: Klassendiagramm der MealplanManager-Komponente

MensaParseScheduling

Typ: interface

Paket: interface/mealplan_management

Beschreibung: Schnittstelle für die Aktivierung eines Parse-Vorgangs der Mensa-Webseite.

Methoden:

- + `start_update_parsing()`
Beginnt das Parse-Vorgang der Mensa-Webseite für die Gerichte des heutigen Tages.
- + `start_full_parsing()`
Beginnt den Parse-Vorgang der Mensa-Webseite für die Gerichte der nächsten vier Wochen.

MealplanManager

Typ: class implements MensaParseScheduling

Paket: layer/logic/mealplan_management

Beschreibung: Implementiert MensaParseScheduling und dessen Funktionen. Die Klasse steuert die Verwaltung geparster Objekte, bevor sie in die Datenbank eingefügt werden.

Methoden:

- + `{static} new(db: MealplanManagementDataAccess, parser: MealplanParser)`
`<<create>>`
Erstellt eine neue Instanz eines MealplanManagers.

RelationResolver

Typ: class

Paket: layer/logic/mealplan_management

Beschreibung: Diese Klasse verwaltet die Relationen von bekannten und neuen Daten.

Methoden:

+ {static} new(db: MealplanManagementDataAccess) <<create>>

Erstellt eine neue Instanz.

+ resolve(canteen: Canteen, data: Date)

Löst alle Probleme verursacht von Datenbankrelationen betroffener Objekte des Mensa-Parsers. Dazu gehört das Entscheiden, ob ein Gericht eine Beilage oder eine Hauptspeise ist. Sind Probleme durch Relationen behoben worden, so werden die Objekte in die Datenbank eingefügt.

Sequenzen der MealplanManager-Komponente

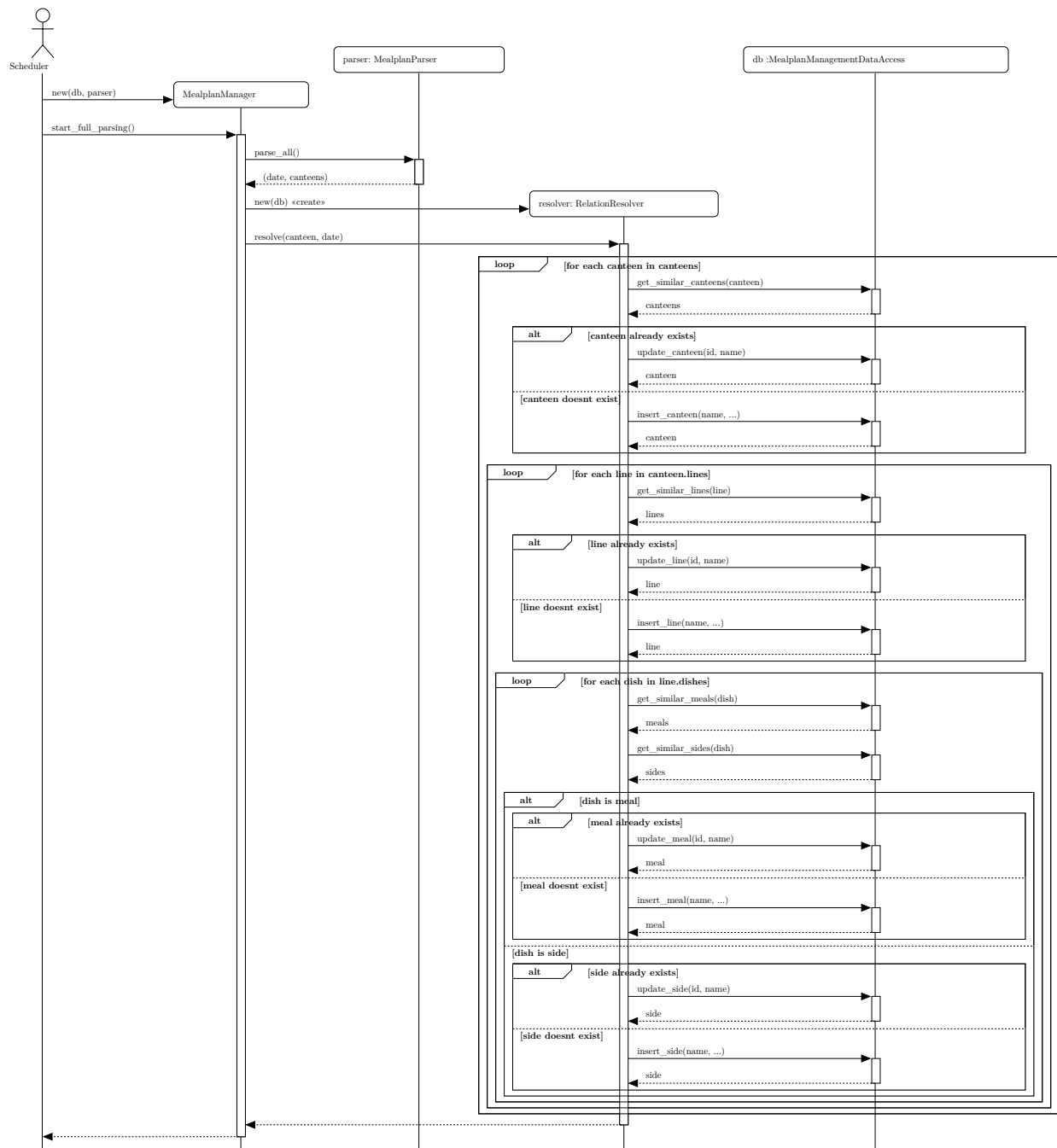


Abbildung 3.39: Sequenzdiagramm der MealplanManager-Komponente

Tests der MealplanManager-Komponente

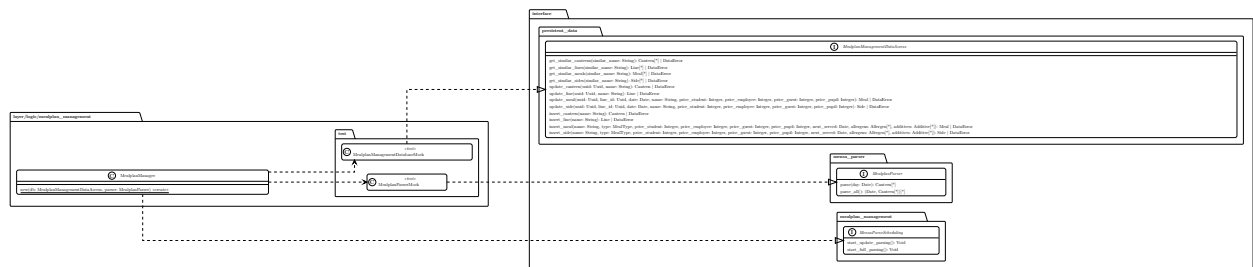


Abbildung 3.40: Klassendiagramm der Test-Klassen der MealplanManager-Komponente

MealplanManagementDatabaseMock

Typ: class implements MealplanManagementDataAccess

Paket: layer/logic/mealplan_management/test

Beschreibung: Diese Testklasse implementiert MealplanManagementDataAccess und simuliert diese Schnittstelle, um die überliegende Komponente isoliert testen zu können.

MealplanParserMock

Typ: class implements MealplanParser

Paket: layer/logic/mealplan_management/test

Beschreibung: Diese Testklasse implementiert MealplanParser und simuliert diese Schnittstelle, um die überliegende Komponente isoliert testen zu können.

3.2.6 Komponente ImageReview

Diese Komponente prüft, ob verlinkte Bilder immer noch beim Bildhoster vorhanden sind, und entfernt sie gegebenenfalls aus der Datenbank.

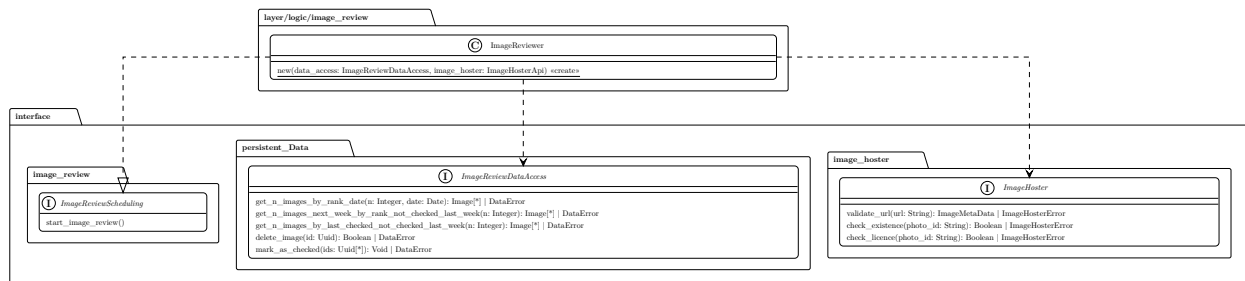


Abbildung 3.41: Klassendiagramm der ImageReview-Komponente

ImageReviewScheduling

Typ: interface

Paket: interface/mensa_management

Beschreibung: Ein Interface, dass das Ausführen von Bildüberprüfungen erlaubt.

Methoden:

```
+ start_image_review()
    Startet die Bildüberprüfung.
```

ImageReviewer

Typ: class implements ImageReviewScheduling

Paket: layer/logic/image_review

Beschreibung: Eine Klasse, die für das Überprüfen der Bilder steuert.

Methoden:

```
+ {static} new(data_access: ImageReviewDataAccess, image_hoster:
    ImageHosterApi) <<create>>
    Erzeugt ein neues Objekt mit den übergebenen Zugängen zu Datenspeicher und Bildhoster.
```

Sequenzen der ImageReview-Komponente

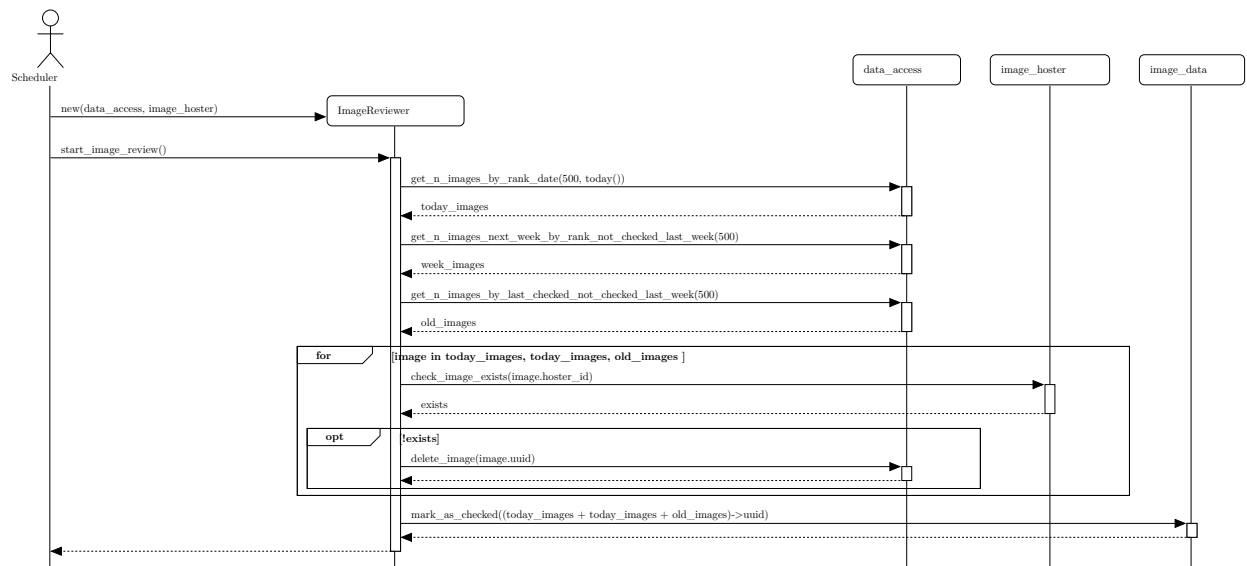


Abbildung 3.42: Sequenzdiagramm der ImageReview-Komponente

Tests der ImageReview-Komponente

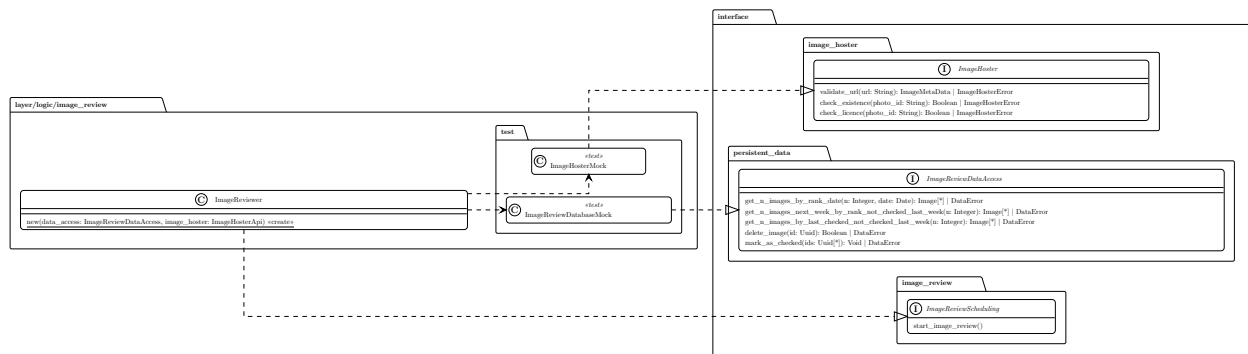


Abbildung 3.43: Klassendiagramm der Testklassen der ImageReview-Komponente

ImageHosterMock

Typ: class implements ImageHoster

Paket: layer/logic/image_review/test

Beschreibung: Diese Klasse simuliert die Schnittstelle `ImageHoster`, um die überliegende Komponente isoliert testen zu können.

ImageReviewerDatabaseMock

Typ: class implements ImageReviewerDataAccess

Paket: layer/logic/image_review/test

Beschreibung: Diese Klasse simuliert die Schnittstelle `ImageReviewerDataAccess`, um die überliegende Komponente isoliert testen zu können.

3.2.7 Komponente Database

Diese Komponente ermöglicht Datenbankzugriffe für die überliegende Komponente **GraphQL**, Komponente **Command**, Komponente **MealplanManagement** und Komponente **ImageReview**. Jede der genannten Komponenten hat eine auf sie zugeschnittene Schnittstelle.

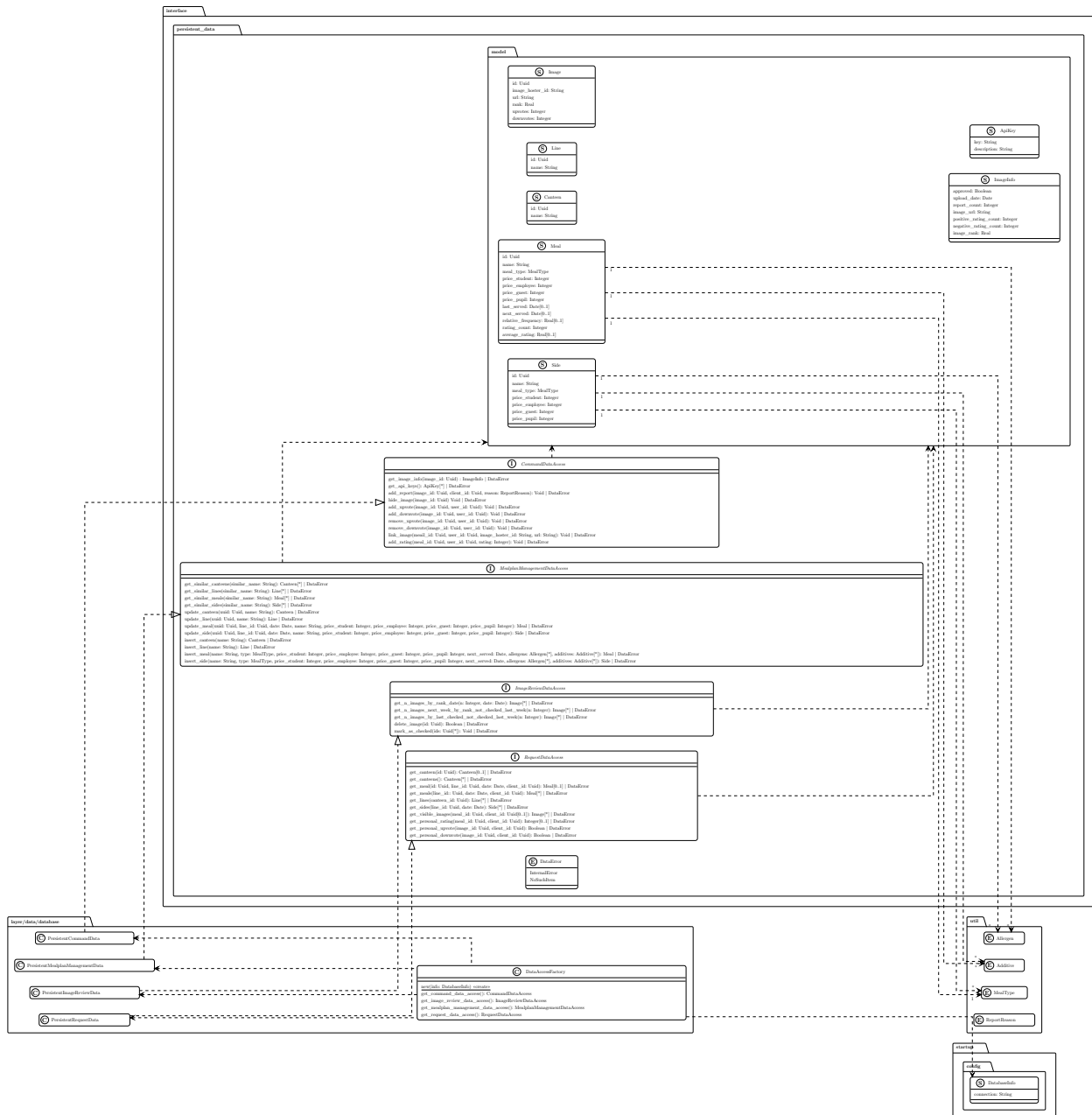


Abbildung 3.44: Klassendiagramm der Database-Komponente

RequestDataAccess**Typ:** interface**Paket:** interface/persistent_data**Beschreibung:** Eine Schnittstelle für GraphQL-Datenbankanfragen.**Methoden:**

- + `get_canteen(id: Uuid): Canteen[0..1] | DataError`
Übergibt die Mensen mit der passenden Uuid aus der Datenbank.
- + `get_canteens(): Canteen[*] | DataError`
Übergibt alle Mensen aus der Datenbank.
- + `get_meal(id: Uuid, line_id: Uuid, date: Date, client_id: Uuid): Meal[0..1] | DataError`
Übergibt die Gerichte, bei denen die Parameter übereinstimmen aus der Datenbank.
- + `get_meals(line_id: Uuid, date: Date, client_id: Uuid): Meal[*] | DataError`
Übergibt alle Gerichte, die mit den Parametern übereinstimmen aus der Datenbank.
- + `get_lines(canteen_id: Uuid): Line[*] | DataError`
Übergibt die Linie aus der Datenbank.
- + `get_sides(line_id: Uuid, date: Date): Side[*] | DataError`
Übergibt alle Beilagen einer Linie zu einem bestimmten Tag.
- + `get_visible_images(meal_id: Uuid, client_id: Uuid[0..1]): Image[*] | DataError`
Übergibt alle Bilder, die einem bestimmten Gericht und Nutzer zugeordnet sind. Bilder, die der Nutzer gemeldet hat, werden nicht übergeben.
- + `get_personal_rating(meal_id: Uuid, client_id: Uuid): Integer[0..1] | DataError`
Übergibt die Bewertung des Nutzers zu einem Gericht.
- + `get_personal_upvote(image_id: Uuid, client_id: Uuid): Boolean | DataError`
Prüft, ob das Bild von dem Nutzer aufgewertet wurde.
- + `get_personal_downvote(image_id: Uuid, client_id: Uuid): Boolean | DataError`
Prüft, ob das Bild von dem Nutzer abgewertet wurde.

CommandDataAccess**Typ:** interface**Paket:** interface/persistent_data**Beschreibung:** Eine Schnittstelle für GraphQL-Manipulationen.**Methoden:**

- + `get_image_info(image_id: Uuid) : ImageInfo | DataError`
Gibt Informationen zu einem Bild zurück.
- + `hide_image(image_id: Uuid) Void | DataError`
Markiert ein Bild als versteckt, sodass es anderen Nutzern nicht mehr angezeigt wird.

- + `add_report(image_id: Uuid, client_id: Uuid, reason: ReportReason): Void | DataError`
Speichert einen Meldeantrag eines Nutzers zu einem Bild.
- + `add_upvote(image_id: Uuid, user_id: Uuid): Void | DataError`
Speichert eine positive Bewertung des angegebenen Nutzers für das angegebene Bild. Eine mögliche negative Bewertung desselben Nutzers wird dabei überschrieben.
- + `add_downvote(image_id: Uuid, user_id: Uuid): Void | DataError`
Speichert eine negative Bewertung des angegebenen Nutzers für das angegebene Bild. Eine mögliche positive Bewertung desselben Nutzers wird dabei überschrieben.
- + `remove_upvote(image_id: Uuid, user_id: Uuid): Void | DataError`
Entfernt eine positive Bewertung des angegebenen Nutzers zum angegebenen Bild.
- + `remove_downvote(image_id: Uuid, user_id: Uuid): Void | DataError`
Entfernt eine negative Bewertung des angegebenen Nutzers zum angegebenen Bild.
- + `link_image(meal_id: Uuid, user_id: Uuid, image_host_id: String, url: String): Void | DataError`
Fügt ein Bild hinzu und verlinkt es mit dem Gericht hinter der UUID.
- + `add_rating(meal_id: Uuid, user_id: Uuid, rating: Integer): Void | DataError`
Fügt einem Gericht eine Bewertung hinzu. Die Bewertung wird mit der UUID des Nutzers verlinkt.
- + `get_api_keys(): ApiKey[*] | DataError`
Lädt alle API-Schlüssel aus der Datenbank und gibt diese zurück.

ImageReviewDataAccess

Typ: interface

Paket: interface/persistent_data

Beschreibung: Eine Schnittstelle zum Erhalten von Bild-Daten.

Methoden:

- + `get_n_images_by_rank_date(n: Integer, date: Date): Image[*] | DataError`
Gibt die ersten `n` Bilder sortiert nach dem Bilddrang zurück, die zu Gerichten gehören, die am gegebenen Tag angeboten werden.
- + `get_n_images_next_week_by_rank_not_checked_last_week(n: Integer): Image[*] | DataError`
Gibt die ersten `n` Bilder sortiert nach dem Bilddrang zurück, die zu Gerichten gehören, die im Laufe der nächsten Woche angeboten werden und in der letzten Woche nicht schon überprüft wurden.
- + `get_n_images_by_last_checked_not_checked_last_week(n: Integer): Image[*] | DataError`
Gibt die ersten `n` Bilder sortiert nach dem Datum der letzten Überprüfung (aufsteigend) zurück, die in der letzten Woche nicht schon überprüft wurden.

- + `delete_image(id: Uuid): Boolean | DataError`
Entfernt ein Bild inklusive seiner Verweise.
- + `mark_as_checked(ids: Uuid[*]): Void | DataError`
Markiert alle Bilder mit den übergebenen UUIDs als überprüft.

MealplanManagementDataAccess

Typ: interface

Paket: interface/persistent_data

Beschreibung: Eine Schnittstelle für das Überprüfen von Relationen und das Einfügen von Strukturen. Die Komponente `MealplanManagement` verwendet diese Schnittstelle.

Methoden:

- + `get_similar_canteens(similar_name: String): Canteen[*] | DataError`
Übergibt alle Mensen, die einen ähnlichen Namen haben.
- + `get_similar_lines(similar_name: String): Line[*] | DataError`
Übergibt alle Linien, die einen ähnlichen Namen haben.
- + `get_similar_meals(similar_name: String): Meal[*] | DataError`
Übergibt alle Gerichte, die einen ähnlichen Namen haben.
- + `get_similar_sides(similar_name: String): Side[*] | DataError`
Übergibt alle Beilagen, die einen ähnlichen Namen haben.
- + `update_canteen(uuid: Uuid, name: String): Canteen | DataError`
Überschreibt die Bezeichnung einer Mensa. Gibt die betroffene Mensa zurück.
- + `update_line(uuid: Uuid, name: String): Line | DataError`
Überschreibt die Bezeichnung einer Linie. Gibt die betroffene Linie zurück.
- + `update_meal(uuid: Uuid, line_id: Uuid, date: Date, name: String, price_student: Integer, price_employee: Integer, price_guest: Integer, price_pupil: Integer): Meal | DataError`
Überschreibt die Bezeichnung und die Preise eines Gerichts zu einem bestimmten Tag im Speiseplan. Gibt das betroffene Gericht zurück.
- + `update_side(uuid: Uuid, line_id: Uuid, date: Date, name: String, price_student: Integer, price_employee: Integer, price_guest: Integer, price_pupil: Integer): Side | DataError`
Überschreibt die Bezeichnung und die Preise einer Beilage zu einem bestimmten Tag im Speiseplan. Gibt die betroffene Beilage zurück.
- + `insert_canteen(name: String): Canteen | DataError`
Fügt eine neue Mensa der Datenbank hinzu.
- + `insert_line(name: String): Line | DataError`
Fügt eine neue Linie der Datenbank hinzu.
- + `insert_meal(name: String, type: MealType, price_student: Integer, price_employee: Integer, price_guest: Integer, price_pupil: Integer, next_served: Date, allergens: Allergen[*], additives: Additive[*]): Meal |`

`DataError`

Fügt ein neues Gericht der Datenbank hinzu.

```
+ insert_side(name: String, type: MealType, price_student: Integer,  
  price_employee: Integer, price_guest: Integer, price_pupil: Integer,  
  next_served: Date, allergens: Allergen[*], additives: Additive[*]): Side |  
DataError
```

Fügt eine neue Beilage der Datenbank hinzu.

Canteen

Typ: struct

Paket: interface/persistent_data/model

Beschreibung: Diese Mensa-Struktur ist angelehnt an die Datenbankentitäten `Canteen`. Diese Struktur wird verwendet für Datenbankoperationen.

Attribute:

```
+ id: Uuid  
  Id der Mensa  
+ name: String  
  Name der Mensa
```

Line

Typ: struct

Paket: interface/persistent_data/model

Beschreibung: Diese Linien-Struktur ist angelehnt an die Datenbankentitäten `Line`. Diese Struktur wird verwendet für Datenbankoperationen.

Attribute:

```
+ id: Uuid  
  Id der Linie  
+ name: String  
  Name der Linie
```

Meal

Typ: struct

Paket: interface/persistent_data/model

Beschreibung: Diese Gericht-Struktur ist angelehnt an die Datenbankentitäten `Food`, `FoodAllergen` und `FoodAdditive`. Diese Struktur wird verwendet für Datenbankoperationen.

Attribute:

```
+ id: Uuid
  Id des Gerichts.
+ name: String
  Name des Gerichts.
+ meal_type: MealType
  Der Typ des Gerichts.
+ price_student: Integer
  Der Preis des Gerichts für Studenten.
+ price_employee: Integer
  Der Preis des Gerichts für Angestellte.
+ price_guest: Integer
  Der Preis des Gerichts für Gäste.
+ price_pupil: Integer
  Der Preis des Gerichts für Schüler.
+ last_served: Date[0..1]
  Der Tag, an dem das Gericht zuletzt serviert wurde.
+ next_served: Date[0..1]
  Der Tag, an dem das Gericht als nächstes serviert wird.
+ relative_frequency: Real[0..1]
  Die relative Häufigkeit des Gerichts.
+ rating_count: Integer
  Die Anzahl aller Bewertungen des Gerichts.
+ average_rating: Real[0..1]
  Das Durchschnittsbewertung des Gerichts.
```

Side

Typ: struct

Paket: interface/persistent_data/model

Beschreibung: Diese Beilagen-Struktur ist angelehnt an die Datenbankentitäten Food, FoodAllergen und FoodAdditive. Diese Struktur wird verwendet für Datenbankoperationen.

Attribute:

```
+ id: Uuid
  Id der Beilage.
+ name: String
  Name der Beilage.
+ meal_type: MealType
  Der Typ der Beilage.
+ price_student: Integer
  Der Preis der Beilage für Studenten.
```

- + `price_employee: Integer`
Der Preis der Beilage für Angestellte.
- + `price_guest: Integer`
Der Preis der Beilage für Gäste.
- + `price_pupil: Integer`
Der Preis der Beilage für Schüler.

Image

Typ: struct

Paket: interface/persistent_data/model

Beschreibung: Diese Bild-Struktur ist angelehnt an die Datenbankentität `Image`. Diese Struktur wird verwendet für Datenbankoperationen.

Attribute:

- + `id: Uuid`
Id des Bildes.
- + `image_host_id: String`
ImageHoster-Identifikator des Bildes.
- + `url: String`
Direkter Link zum Bild auf der Hoster-Webseite.
- + `rank: Real`
Rang des Bildes.
- + `upvotes: Integer`
Menge der Upvotes des Bildes.
- + `downvotes: Integer`
Menge der Downvotes des Bildes.

ImageInfo

Typ: struct

Paket: interface/persistent_data/model

Beschreibung: Diese Bild-Info-Struktur ist angelehnt an die Datenbankentität `ImageRating` und `ImageReport`. Diese Struktur wird verwendet für Datenbankoperationen.

Attribute:

- + `approved: Boolean`
Ab das Bild von einem Administrator als valide bestätigt wurde.
- + `upload_date: Date`
Hochladedatum des Bildes.
- + `report_count: Integer`
Anzahl der offenen Meldeanträge zu diesem Bild.

- + `image_url: String`
Direkter Web-Link zum Bild.
- + `positive_rating_count: Integer`
Anzahl an positiven Bewertungen zu diesem Bild.
- + `negative_rating_count: Integer`
Anzahl an negativen Bewertungen zu diesem Bild.
- + `image_rank: Real`
Bildrang, nach dem Bilder geordnet sind.

DataAccessFactory

Typ: class

Paket: layer/data/database

Beschreibung: Diese Klasse dient zur Verteilung der unterschiedlichen Datenbankanfragen.

Methoden:

- + `{static} new(info: DatabaseInfo) <<create>>`
Erstellt ein Objekt zur Verteilung der Datenbankanfragen. Bei der Erstellung wird eine Verbindung zur Datenbank hergestellt.
- + `get_api_key_data_access(): ApiKeyDataAccess`
Übergibt ein Objekt, in dem sich alle Datenbankanfragen befinden, die ApiKeys betreffen.
- + `get_command_data_access(): CommandDataAccess`
Übergibt ein Objekt, in dem sich alle Datenbankanfragen befinden, die GraphQL-Manipulationen betreffen.
- + `get_image_review_data_access(): ImageReviewDataAccess`
Übergibt ein Objekt, in dem sich alle Datenbankanfragen befinden, die Bildvalidierung betreffen.
- + `get_mealplan_management_data_access(): MealplanManagementDataAccess`
Übergibt ein Objekt, in dem sich alle Datenbankanfragen befinden, die den Mensa-Parser betreffen.
- + `get_request_data_access(): RequestDataAccess`
Übergibt ein Objekt, in dem sich alle Datenbankanfragen befinden, die GraphQL-Objektanfragen betreffen.

PersistentRequestData

Typ: class implements RequestDataAccess

Paket: layer/data/database

Beschreibung: Diese Klasse implementiert alle Datenbankanfragen, die GraphQL-Objektanfragen betreffen.

PersistentCommandData

Typ: class implements `CommandDataAccess`

Paket: `layer/data/database`

Beschreibung: Diese Klasse implementiert alle Datenbankabfragen, die GraphQL-Manipulationen betreffen.

PersistentImageReviewData

Typ: class implements `ImageReviewDataAccess`

Paket: `layer/data/database`

Beschreibung: Diese Klasse implementiert alle Datenbankabfragen, die Bildvalidierung betreffen.

PersistentMealplanManagementData

Typ: class implements `MealplanManagementDataAccess`

Paket: `layer/data/database`

Beschreibung: Diese Klasse implementiert alle Datenbankabfragen, die den Mensa-Parser betreffen.

DataError

Typ: enum

Paket: `layer/data/database`

Beschreibung: Enumerationstyp für mögliche Fehlschläge von Datenabfragen.

Varianten:

- + `NoSuchItem`

- Die angefragten Daten existieren nicht.

- + `InternalError`

- Fehler bei der Abfrage oder Verbindung mit der Datenbank.

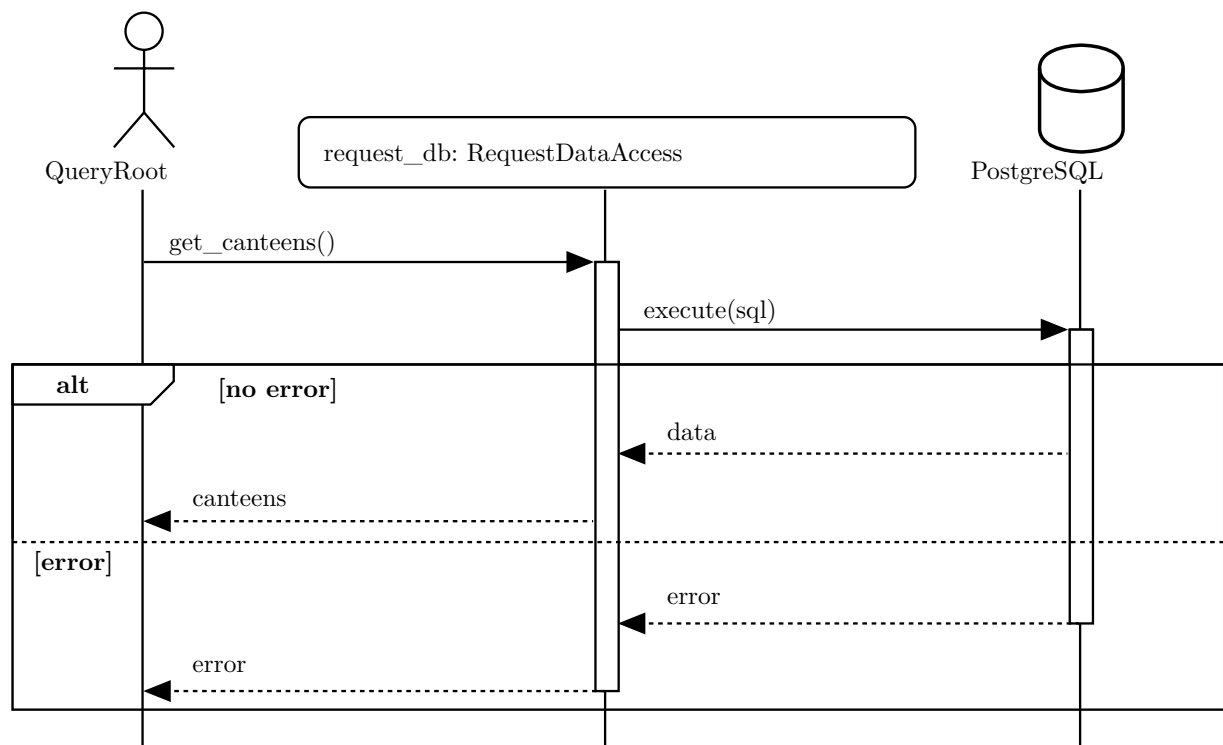
Sequenzen der Database-Komponente

Abbildung 3.45: Sequenzdiagramm für eine Datenbankabfrage über die RequestDataAccess-Schnittstelle der Database-Komponente

3.2.8 Komponente SwKaParser

Diese Komponente ist für die Abfrage der aktuellen Speisepläne von den Webseiten der Mensen zuständig. Die Aufgabe des MensaParsers ist das Aufrufen und Laden der SwKa-Webseite² sowie das Transformieren von Text in Datenobjekte.

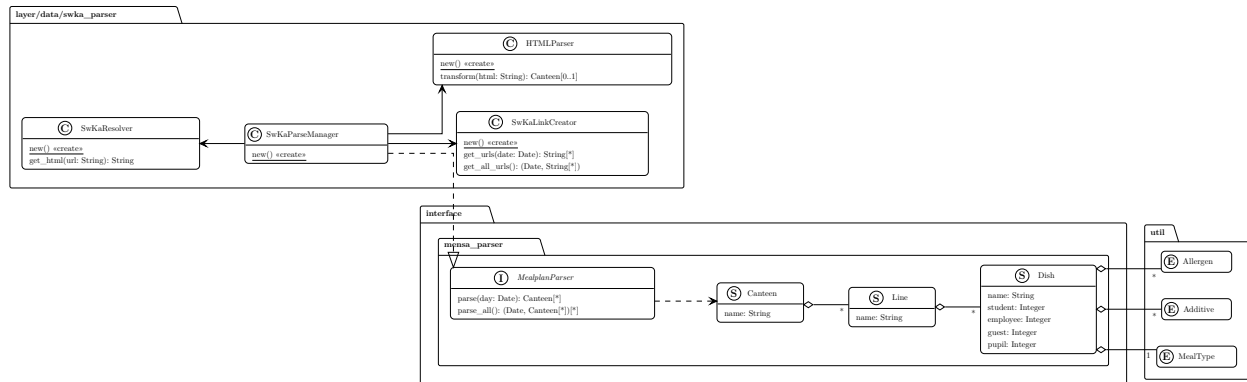


Abbildung 3.46: Klassendiagramm der SwKaParser-Komponente

MealplanParser

Typ: interface

Paket: interface/mensa_parser

Beschreibung: Diese Schnittstelle erlaubt das Lesen von Speiseplänen.

Methoden:

+ `parse(day: Date): Canteen[*]`

Beginnt den Parse-Vorgang der Mensa-Webseite für Gerichte, die an übergebenen Tag angeboten werden.

+ `parse_all(): (Date, Canteen[*])`

Beginnt den Parse-Vorgang der Mensa-Webseite für Gerichte der nächsten vier Tage.

Canteen

Typ: struct

Paket: interface/mensa_parser

Beschreibung: Die Mensa-Struktur, die alle Speiseplandaten einer Mensa beinhaltet, die ohne Vorverarbeitung ausgelesen werden können.

Attribute:

+ `name: String`

Bezeichnung der Mensa.

²www.sw-ka.de/de/hochschulgastronomie/speiseplan

Line**Typ:** struct**Paket:** interface/mensa_parser**Beschreibung:** Die Linien-Struktur, die eine Bezeichnung sowie alle Gerichte beinhaltet, die ohne Vorverarbeitung ausgelesen werden können.**Attribute:**

- + name: String
Bezeichnung der Linie.

Dish**Typ:** struct**Paket:** interface/mensa_parser**Beschreibung:** Diese Struktur beinhaltet alle Daten zu einem Gericht, die der SwKa-Webseite entnommen werden können.**Attribute:**

- + name: String
Bezeichnung des Gerichts.
- + student: Integer
Preis des Gerichts für Studenten.
- + employee: Integer
Preis des Gerichts für Angestellte.
- + guest: Integer
Preis des Gerichts für Gäste.
- + pupil: Integer
Preis des Gerichts für Schüler.

SwKaParseManager**Typ:** class implements MealplanParser**Paket:** layer/data/swka_parser**Beschreibung:** Eine Implementierung eines MealplanParsers angepasst für die SwKa-Webseite des Studierendenwerks.**Methoden:**

- + {static} new() <<create>>
Erstellt eine Instanz. Beim Instanziiieren werden alle benötigten Objekte für den Parse-Vorgang erstellt. Dazu gehört der SwKaResolver, der SwKaLinkCreator und der HTMLParser.

SwKaResolver

Typ: class

Paket: layer/data/swka_parser

Beschreibung: Diese Klasse baut eine Verbindung zu einer Webseite auf und lädt dessen HTML-Code.

Methoden:

- + {static} new() <<create>>
Erstellt eine Instanz des SwKaResolvers.
- + get_html(url: String): String
Baut eine Verbindung zur Webseite mit der übergebenen URL auf. Lädt den HTML-Code der Webseite in einen String und gibt diesen zurück. Ist keine Verbindung möglich, wird ein leerer String zurückgegeben.

SwKaLinkCreator

Typ: class

Paket: layer/data/swka_parser

Beschreibung: Diese Klasse baut spezifische Links für die Anfragen an die SwKa-Webseite. Jede Mensa des Studierendenwerks wird über eine andere URL abgefragt.

Methoden:

- + {static} new() <<create>>
Erstellt eine Instanz des SwKaLinkCreators.
- + get_urls(date: Date): String[*]
Erstellt alle URLs zu den Mensen der SwKa-Webseite an dem übergebenen Tag und gibt diese zurück.
- + get_all_urls(): (Date, String[*])
Erstellt alle URLs zu den Mensen der SwKa-Webseite der nächsten vier Tagen.

HTMLParser

Typ: class

Paket: layer/data/swka_parser

Beschreibung: Diese Klasse interpretiert einen HTML-String in eine Daten-Struktur.

Methoden:

- + {static} new() <<create>>
Erstellt eine Instanz des HTMLParsers.
- + transform(html: String): Canteen[0..1]
Diese Funktion transformiert den übergebenen HTML-String in eine Mensen-Struktur.

Sequenzen der SwKaParser-Komponente

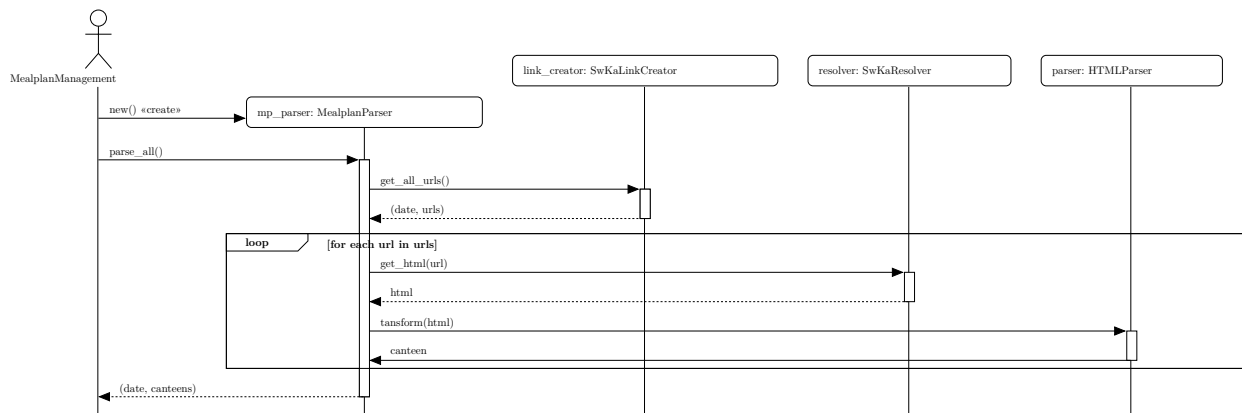


Abbildung 3.47: Sequenzdiagramm einer Parse-Abfolge in der SwKaParser-Komponente

3.2.9 Komponente FlickrApi

Diese Komponente ist für die Kommunikation mit dem Bildhoster „Flickr“³ zuständig. Um Bilder oder Bildlinks (URLs) zu überprüfen, muss eine Verbindung zur Flickr-API aufgebaut, Anfragen gestellt und resultierende Daten verarbeitet werden.

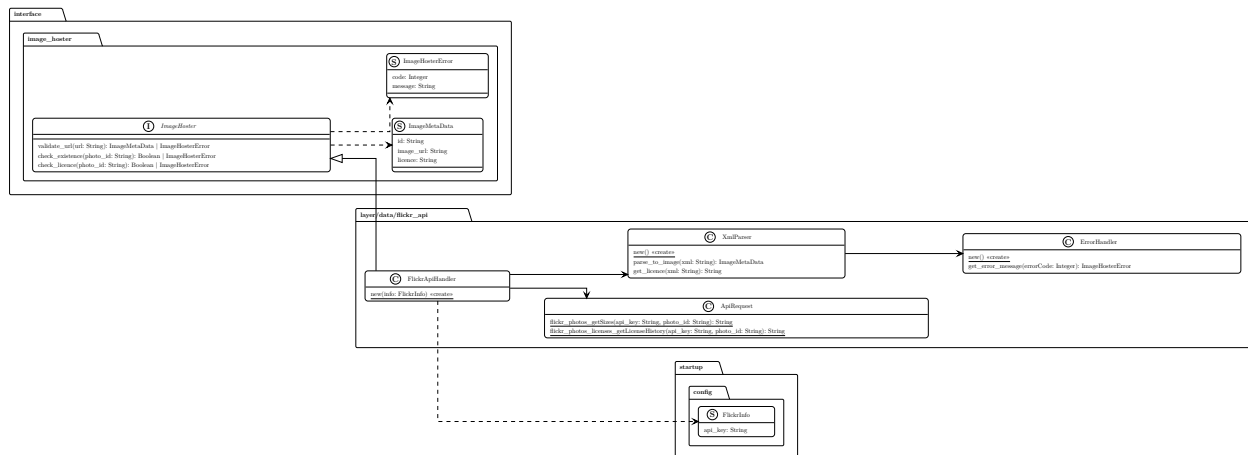


Abbildung 3.48: Klassendiagramm der FlickrApi-Komponente

ImageHoster

Typ: interface

Paket: interface/image_host

Beschreibung: Schnittstelle für die Kommunikation mit dem jeweiligen Image-Hoster.

Methoden:

- ```
+ validate_url(url: String): ImageMetadata | ImageHostError
 Prüft, ob der übergebene Link zum ImageHoster führt. Prüft zusätzlich alle anderen Anforderungen an das jeweilige Bild hinter dem Link und übergibt ein Struct mit allen Informationen zurück.

+ check_existence(photo_id: String): Boolean | ImageHostError
 Prüft, ob ein Bild noch beim Hoster zur Verfügung steht.

+ check_licence(photo_id: String): Boolean | ImageHostError
 Überprüft die Lizenz eines Bildes anhand des Bild-Identifikators des Hosters.
```

## FlickrApiHandler

**Typ:** class implements ImageHoster

**Paket:** layer/data/flickr\_api

**Beschreibung:** Implementierung des Bildhosters Flickr.

<sup>3</sup><https://www.flickr.com/services/api/>

**Methoden:**

- + {static} new(info: FlickrInfo) <<create>>  
Erstellt eine Instanz des Handlers.

**ApiRequest**

**Typ:** class

**Paket:** layer/data/flickr\_api

**Beschreibung:** Baut eine Verbindung zur Flickr-API auf und stellt Anfragen.

**Methoden:**

- + {static} flickr\_photos\_getSizes(api\_key: String, photo\_id: String): String  
Fragt Links zu allen verfügbaren Größen des übergebenen Bildes an.
- + {static} flickr\_photos\_licenses\_getLicenseHistory(api\_key: String, photo\_id: String): String  
Gibt eine Liste von allen vergangenen und aktuellen Lizenzen des Bildes zurück.

**ErrorHandler**

**Typ:** class

**Paket:** layer/data/flickr\_api

**Beschreibung:** Interpretiert und behandelt auftretende Fehler bei Flickr-API-Anfragen.

**Methoden:**

- + {static} new() <<create>>  
Erstellt eine Instanz des Error-Handlers.
- + get\_error\_message(errorCode: Integer): ImageHosterError  
Interpretiert den übergebenen Fehlercode.

**XmlParser**

**Typ:** class

**Paket:** layer/data/flickr\_api

**Beschreibung:** Klasse zur Informationsbeschaffung aus XML-Strings.

**Methoden:**

- + {static} new() <<create>>  
Erstellt eine Instanz des XMLParsers.
- + parse\_to\_image(xml: String): ImageMetaData  
Übersetzt einen XML-String in ein ImageMetaData-Struct.
- + get\_licence(xml: String): String  
Filtert die aktuelle Lizenz aus einem XML-String.

## Sequenzen der FlickrApi-Komponente

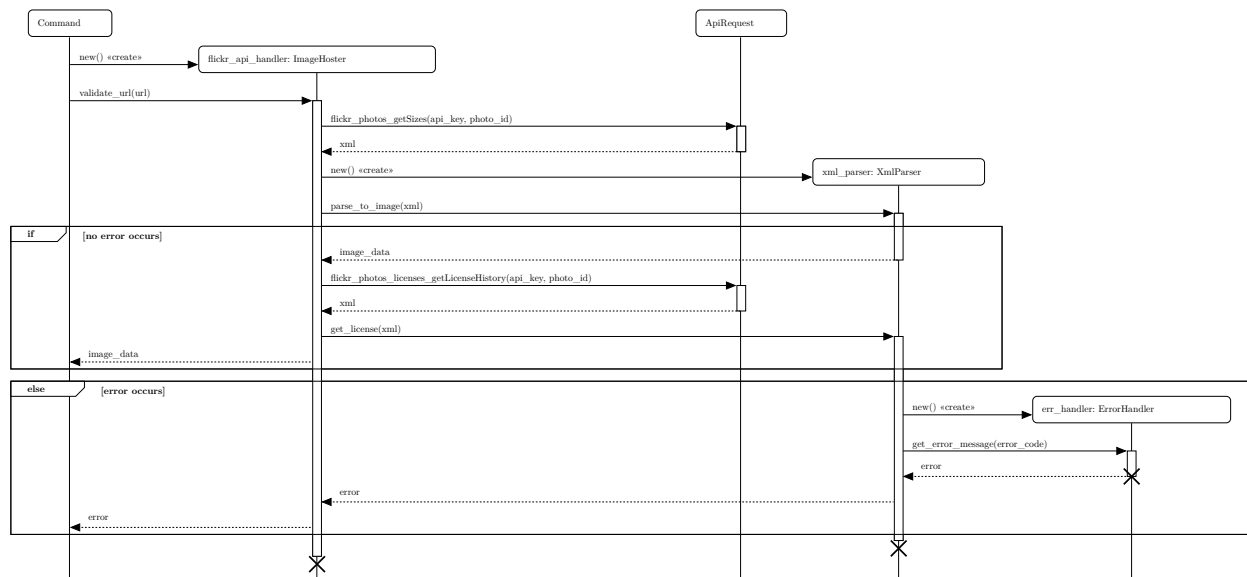


Abbildung 3.49: Sequenzdiagramm der Validierung einer übergebenen URL in der FlickrApi-Komponente

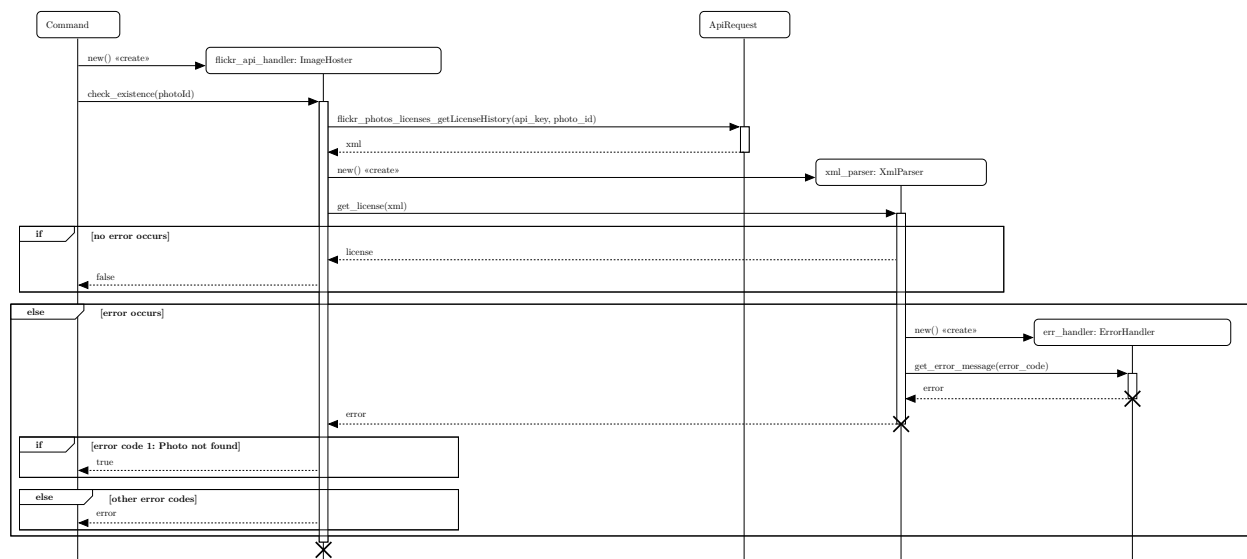


Abbildung 3.50: Sequenzdiagramm der Überprüfen der Existenz eines Bildes in der FlickrApi-Komponente

### 3.2.10 Komponente Mail

Diese Komponente ist für das Senden von E-Mails an den Administrator zuständig.

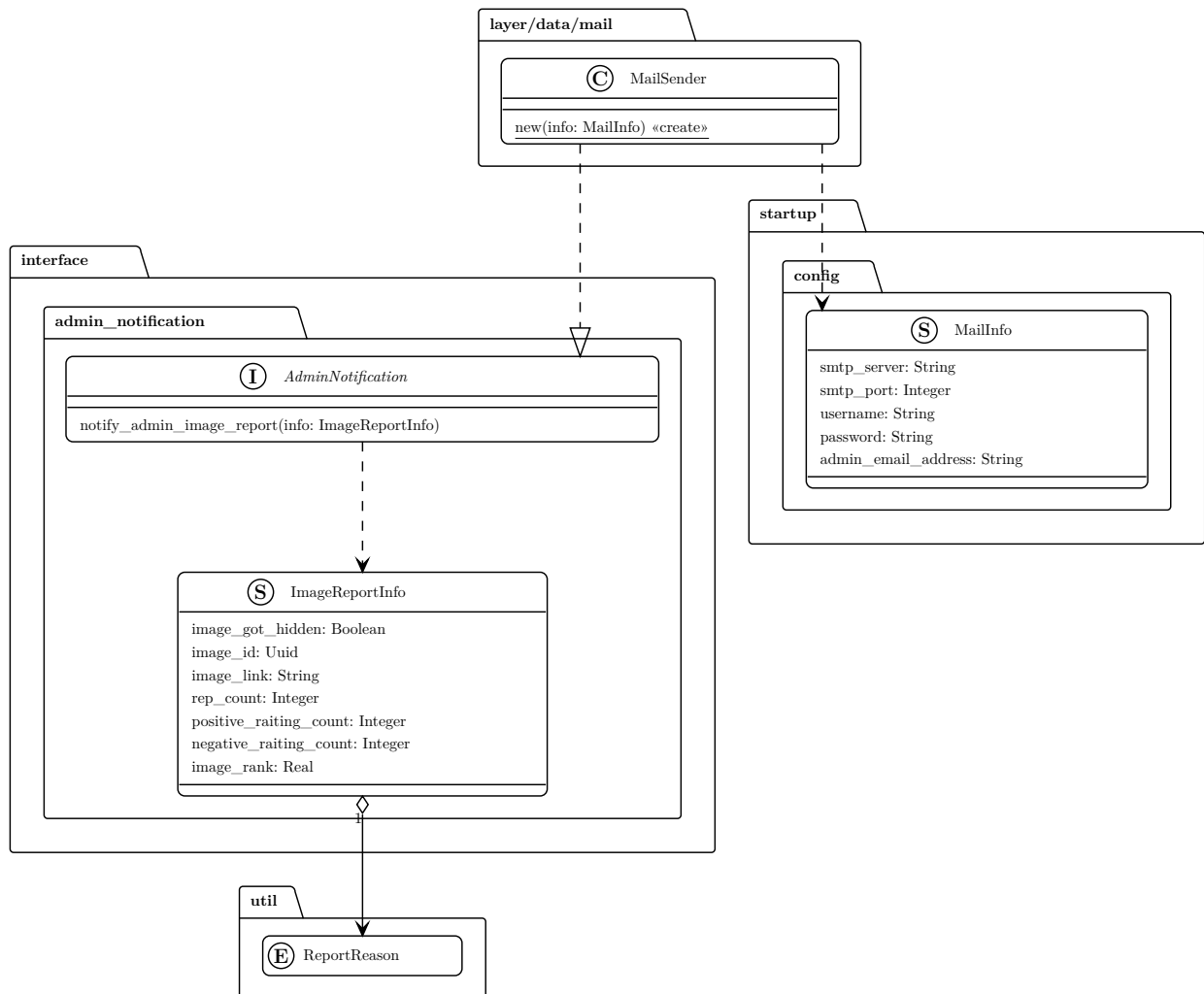


Abbildung 3.51: Klassendiagramm der Mail-Komponente

#### AdminNotification

**Typ:** interface

**Paket:** interface/admin\_notification

**Beschreibung:** Interface für die Benachrichtigung von Administratoren.

**Methoden:**

+ `notify_admin_image_report(info: ImageReportInfo)`

Benachrichtigt den Administrator über ein neu gemeldetes Bild und die vom automatischen System vorgenommene Reaktion.

**ImageReportInfo**

**Typ:** struct

**Paket:** interface/admin\_notification

**Beschreibung:** Struktur mit Informationen über einen Meldeantrag.

**Attribute:**

- + reason: ReportReason  
Grund für Meldung als ReportReason.
- + image\_got\_hidden: Boolean  
Ob das Bild schon vom automatischen System ausgeblendet wurde.
- + image\_id: Integer  
Eindeutiger Identifikator des gemeldeten Bildes.
- + image\_link: String  
Link zum Bild.
- + report\_count: Integer  
Anzahl der Meldeanträge zum Bild, inklusive dem Aktuellen.
- + positive\_raiting\_count: Integer  
Anzahl der positiven Bewertungen des gemeldeten Bildes.
- + negative\_raiting\_count: Integer  
Anzahl der negativen Bewertungen des gemeldeten Bildes.
- + get\_image\_rank: Real  
Bildrang des gemeldeten Bildes.

**MailSender**

**Typ:** class implements AdminNotification

**Paket:** layer/data/mail

**Beschreibung:** Klasse für das senden von E-Mails an einen Administrator.

**Methoden:**

- + {static} new(config: MailConfig) <<create>>  
Erzeugt ein neues Objekt mit gegebener Konfiguration zum Senden von E-Mails.



### Sequenzen der Mail-Komponente

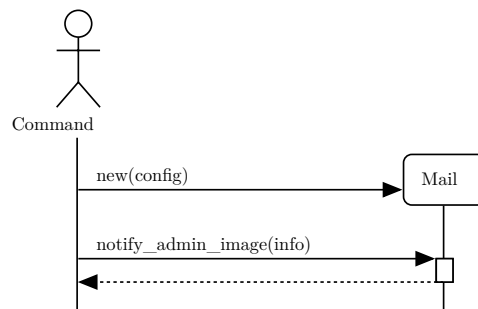


Abbildung 3.52: Sequenzdiagramm der Mail-Komponente

### 3.2.11 Paket `util`

Dieses Paket beinhaltet einige Enumerationstypen, die in mehreren Komponenten werden.

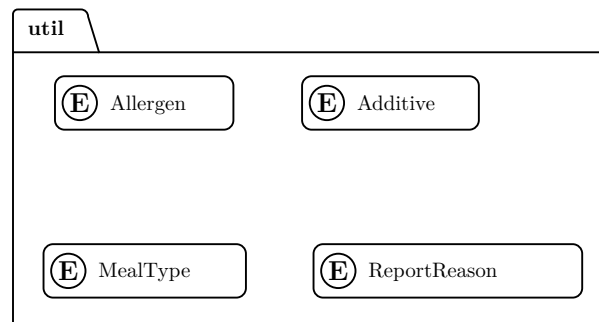


Abbildung 3.53: Klassendiagramm des `util`-Pakets

#### **Allergen**

**Typ:** `enum`

**Paket:** `util`

**Beschreibung:** Dieser Enumerationstyp stellt alle möglichen Allergene dar. Eine genauere Ausführung dieser ist in 4.1.4 Entitäten zu finden.

#### **Additive**

**Typ:** `enum`

**Paket:** `util`

**Beschreibung:** Dieser Enumerationstyp stellt alle möglichen Zusatzstoffe dar. Eine genauere Ausführung dieser ist in 4.1.4 Entitäten zu finden.

#### **MealType**

**Typ:** `enum`

**Paket:** `util`

**Beschreibung:** Dieser Enumerationstyp stellt alle möglichen Gerichtstypen dar. Eine genauere Ausführung dieser ist in 4.1.4 Entitäten zu finden.

#### **ReportReason**

**Typ:** `enum`

**Paket:** `util`

**Beschreibung:** Dieser Enumerationstyp stellt alle möglichen Meldegründe dar. Eine genauere Ausführung dieser ist in 4.1.4 Entitäten zu finden.

## Kapitel 4

# Schemata

In diesem Kapitel werden die Schemata für die Datenbanken, sowohl die des Servers als auch die lokale Datenbank im Client, beschreiben. Zusätzlich wird die API-Schnittstelle zwischen Client und Server definiert.

## 4.1 Datenbankschema Server

### 4.1.1 Entity-Relationship-Model

Im Folgenden wird das Datenbankschema als ER-Model dargestellt, wobei dabei zu beachten ist, dass die Attribute der einzelnen Entitäten hier fehlen und im Relationenschema genauer beschrieben werden.

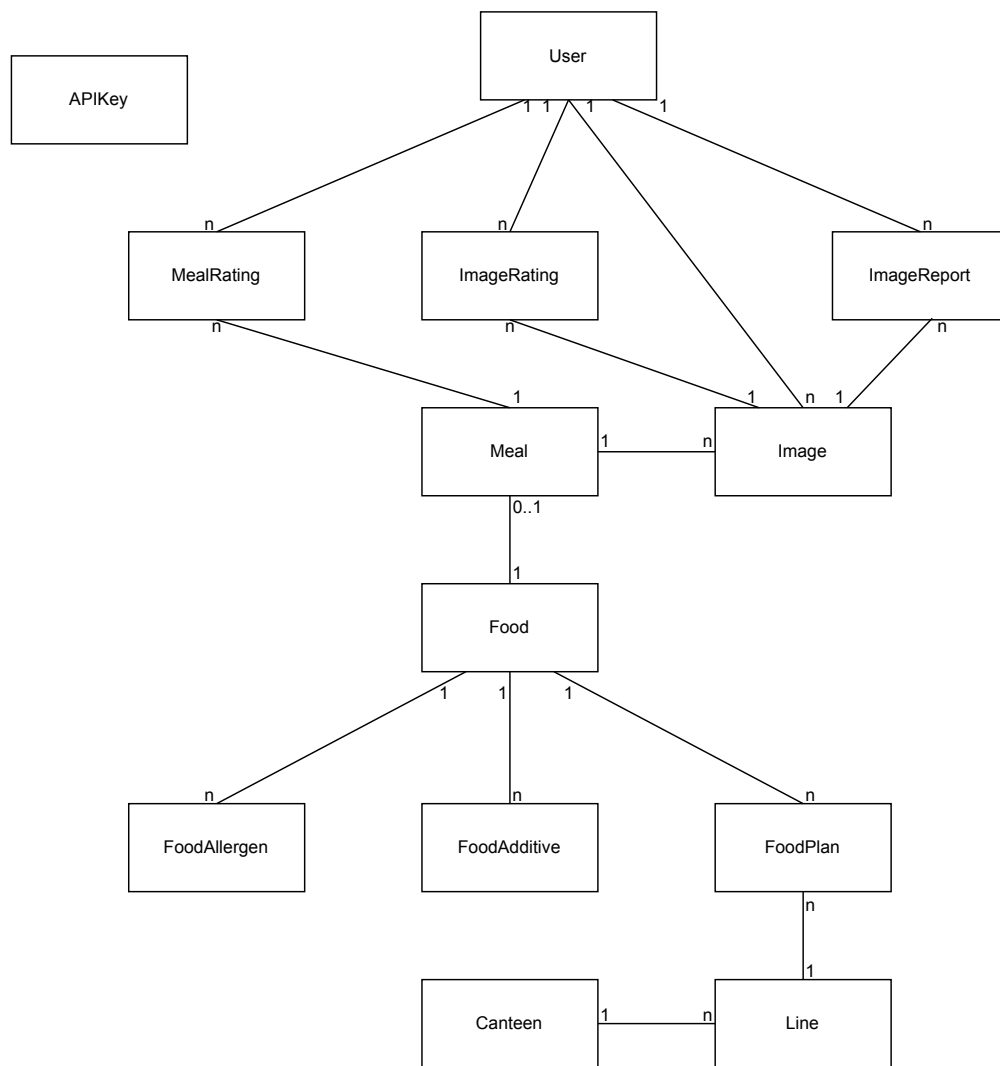


Abbildung 4.1: Entity-Relationship-Model der Datenbank des Backends ohne Attribute

### 4.1.2 Relationenschema

Im Folgenden ist das Relationenschema mit den Attributen der jeweiligen Entitäten zu sehen. Dabei sind die Schlüssel unterstrichen und die Fremdschlüssel kursiv dargestellt. Die angegebenen

Typen entsprechen dabei den PostgreSQL-Typen.

User = (userID: UUID)

Food = (foodID: UUID, name: text, foodType: ENUM)

Meal = (foodID: UUID)

MealRating = (userID: UUID, foodID: UUID, rating: smallint)

Image = (imageID: UUID, userID: UUID, foodID: UUID, id: text, url: text, linkDate: date, lastVerifiedDate: date, approved: boolean, currentlyVisible: boolean)

ImageRating = (imageID: UUID, userID: UUID, rating: smallint)

ImageReport = (imageID: UUID, userID: UUID, reportDate: date, category: ENUM)

FoodAllergen = (foodID: UUID, allergen: ENUM)

FoodAdditive = (foodID: UUID, additive: ENUM)

FoodPlan = (lineID: UUID, foodID: UUID, date: date, priceStudent: Preis, priceEmployee: Preis, pricePupil: Preis, priceGuest: Preis)

Line = (lineID: UUID, canteenID: UUID, name: text, position: smallint)

Canteen = (canteenID: UUID, name: text)

APIKeys = (key: text, description: text)

### 4.1.3 Domains

#### Preis

Preis ist vom Typ `smallint` und wird aus Gründen der Erweiter- und Wartbarkeit als Domain modelliert.

### 4.1.4 Entitäten

Im Folgenden werden die einzelnen Entitäten genauer beschrieben. Dabei sind alle Attribute, die „ID“ im Name enthalten nur für Schlüssel relevant und ihr Wert hat bis auf die Identifikation keine weitere Bedeutung. Sie werden deswegen im weiteren Prozess nicht weiter beschreiben.

#### User

Diese Entität enthält alle Identifikatoren der Nutzer, die schon mit dem Sever kommuniziert haben. Die `userID` wird in der Datenbank zur Identifikation der Nutzer für Bewertungen und Bilder benutzt.

#### Food

Diese Entität enthält alle Gerichte und Beilagen, die vom Studierendenwerk angeboten werden.

Attribute:

**name:** Name des Gerichts

**foodType:** Gerichtstyp, der folgende Werte annehmen kann (Beschreibungen vgl. `MealType`):

- VEGAN
- VEGETARIAN
- BEEF
- BEEF\_AW
- PORK
- PORK\_AW
- FISH
- UNKNOWN

### **Meal**

Diese Entität enthält den Identifikator aller Gerichte, da diese einige weiteren Eigenschaften haben als Beilagen und deswegen extra behandelt werden müssen.

### **MealRating**

Diese Entität enthält alle Bewertungen, die Nutzer zu Gerichten abgegeben haben.

Attribute:

**rating**: vom Nutzer abgegebene Bewertung

### **Image**

Diese Entität enthält Informationen zu einem Bild, das von einem Nutzer hochgeladen wurde.

Attribute:

**id**: von Flickr vergebender Identifikator

**url**: URL, die direkt zu dem Bild führt (und nicht zur Rahmenseite)

**linkDate**: Datum, an dem das Bild verlinkt wurde

**lastVerifiedDate**: Datum, an dem das Bild zuletzt überprüft wurde

**approved**: gibt an, ob ein Administrator das Bild bereits überprüft hat

**currentlyVisible**: gibt an, ob ein Bild angezeigt werden darf - tritt nicht ein, wenn es zu oft gemeldet wurde

### **ImageRating**

Diese Entität enthält alle Bewertungen, die Nutzer zu Bildern abgegeben haben.

Attribute:

**rating**: vom Nutzer abgegebene Bewertung

### **ImageReport**

Diese Entität enthält alle Informationen zu den Meldeanträgen, die noch nicht von einem Administrator überprüft wurden. Sobald ein Administrator ein Bild überprüft hat, werden die zugehörigen Meldeanträge entfernt.

Attribute:

**reportDate**: Tag der Meldung

**category:** Kategorie des Meldegrundes, der vom Nutzer angegeben wird, der folgende Werte annehmen kann (Beschreibungen vgl. ReportReason):

- OFFENSIVE
- ADVERT
- NO\_MEAL
- WRONG\_MEAL
- VIOLATES\_RIGHTS
- OTHER

### **FoodAllergen**

Diese Entität enthält alle Allergene, die in Gerichten enthalten sind.

Attribute:

**allergen:** Allergen, wobei es folgende Allergene gibt (Beschreibungen vgl. Allergen):

- CA
- DI
- EI
- ER
- FI
- GE
- HF
- HA
- KA
- KR
- LU
- MA
- ML
- PA
- PE
- PI
- QU
- RO
- SA
- SE
- SF
- SN
- SO
- WA
- WE
- WT

**FoodAdditive**

Diese Entität enthält alle Zusatzstoffe, die in Gerichten enthalten sind.

Attribute:

**additive:** Zusatzstoffe, wobei es folgende Zusatzstoffe gibt (Beschreibungen vgl. Additive):

- COLORANT
- PRESERVING\_AGENTS
- ANTIOXIDANT\_AGENTS
- FLAVOUR\_ENHANCER
- PHOSPHATE
- SURFACE\_WAXED
- SULPHUR
- ARTIFICALLY\_BLACKENED\_OLIVES
- SWEETENER
- LAXATIVE\_IF\_OVERUSED
- PHENYLALANINE
- ALCOHOL
- PRESSED\_MEET
- GLAZING\_WITH\_CACAO
- PRESSED\_FISH

**FoodPlan**

Diese Entität enthält den Speiseplan und die verschiedenen Preise.

Attribute:

**date:** Datum des Speiseplaneintrags

**priceStudent:** Preis für Studierende

**priceEmployee:** Preis für Mitarbeitende

**pricePupil:** Preis für Schüler

**priceGuest:** Preis für Gäste

**Line**

Diese Entität enthält alle Linien der Mensen des Studierendenwerks.

Attribute:

**name:** Name der Linie

**position:** Position auf der die Linie auf der Webseite des Studierendenwerks angezeigt wird.

**Canteen**

Diese Entität enthält die verschiedenen Mensen, die es gibt.

Attribute:

**name:** Name der Mensa



**APIKey**

Diese Entität enthält alle API Schlüssel.

Attribute:

**key:** API Schlüssel

**description:** Beschreibung des Schlüssels

## 4.2 Datenbankschema Client

Für die Datenbank im Client wird mit SQLite gearbeitet. Diese wird dazu genutzt, Speis für die nächste Woche lokal zu speichern, um auch ohne Verbindung zum Server ältere Daten abrufen zu können und so das Benutzererlebnis zu verbessern. Dabei werden nicht alle Daten gespeichert. Dabei wurde darauf geachtet, dass die Konvertierung der Daten zwischen GraphQL und der Datenbank und zwischen der Datenbank und den Dart-Klassen möglichst einfach ist, weshalb sich das Schema stark von dem im Server unterscheidet.

Da SQLite keine ENUMs unterstützt, werden hier CHECK constraints für TEXT benutzt, die dafür sorgen, dass nur vorgegebene Werte erlaubt sind. Diese werden im Folgenden als ENUMs bezeichnet.

### 4.2.1 Entity-Relationship-Model

Im Folgenden wird das Datenbankschema als ER-Model dargestellt, wobei dabei zu beachten ist, dass die Attribute der einzelnen Entitäten hier fehlen und im Relationenschema genauer beschrieben werden.

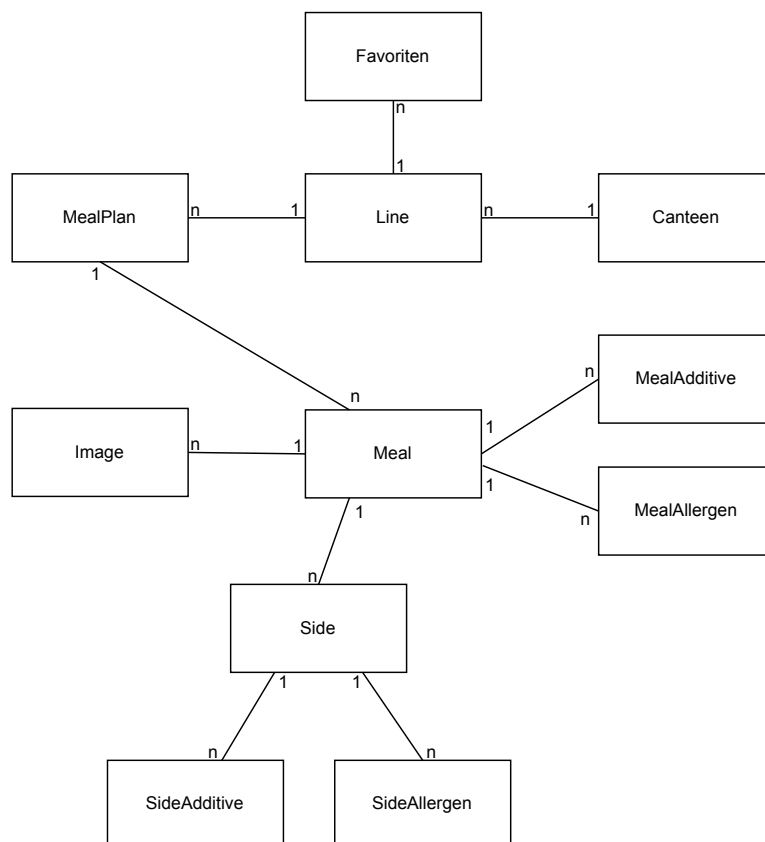


Abbildung 4.2: Entity-Relationship-Model der Datenbank des Frontends ohne Attribute

### 4.2.2 Relationenschema

Im Folgenden ist das Relationenschema mit den Attributen der jeweiligen Entitäten zu sehen. Dabei sind die Schlüssel unterstrichen und die Fremdschlüssel kursiv dargestellt. Die angegebenen Typen entsprechen dabei den SQLite-Typen.

MealPlan = (mealplanID: TEXT, *lineID*: TEXT, *date*: TEXT, isClosed: BOOLEAN)

Line = (lineID: TEXT, *canteenID*: TEXT, name: TEXT, position: INT)

Canteen = (canteenID: TEXT, name: TEXT)

Meal = (mealID: TEXT, *mealplanID*: TEXT, name: TEXT, foodtype: ENUM, priceStudent: INT, priceEmployee: INT, pricePupil: INT, priceGuest: INT, individualRating: INT, numberOfRatings: INT, averageRating: DECIMAL(1,1), lastServed: TEXT, nextServed: TEXT, relativeFrequency: ENUM)

Side = (sideID: TEXT, *mealID*: TEXT, name: TEXT, foodtype: ENUM, priceStudent: INT, priceEmployee: INT, pricePupil: INT, priceGuest: INT)

Image = (imageId: TEXT, url: TEXT)

MealAdditive = (*mealID*: TEXT, additive: ENUM)

MealAllergen = (*mealID*: TEXT, allergen: ENUM)

SideAdditive = (*sideID*: TEXT, additive: ENUM)

SideAllergen = (*sideID*: TEXT, allergen: ENUM)

Favorites = (favoriteID: TEXT, *lineID*: TEXT, lastDate: TEXT, foodType: ENUM, priceStudent: INT, priceEmployee: INT, pricePupil: INT, priceGuest: INT)

### 4.2.3 Entitäten

Im Folgenden werden die einzelnen Entitäten genauer beschrieben. Dabei sind alle Attribute, die IDs enthalten nur für Schlüssel relevant und ihr Wert hat bis auf die Identifikation keine weitere Bedeutung. Sie werden deswegen im weiteren Prozess nicht weiter beschreiben.

#### MealPlan

Diese Entität speichert den Speiseplan für die nächste Woche.

Attribute:

**date**: Datum des Speiseplaneintrags

**isClosed**: Besagt, ob die Linie an einem gewissen Tag offen ist, oder nicht **date**: Datum des Speiseplaneintrags

**Line**

Diese Entität enthält alle Linien der Mensen des Studierendenwerks.

Attribute:

**name:** Name der Linie

**position:** Position auf der die Linie auf der Webseite des Studierendenwerks angezeigt wird.

**Canteen**

Diese Entität enthält die verschiedenen Mensen, die es gibt.

Attribute:

**name:** Name der Mensa

**Meal**

Diese Entität enthält alle Gerichte, die in der nächsten Woche angeboten werden.

Attribute:

**name:** Der Name des Gerichts

**foodtype:** Gerichtstyp, der folgende Werte annehmen kann (Beschreibungen vgl. MealType):

- VEGAN
- VEGETARIAN
- BEEF
- BEEF\_AW
- PORK
- PORK\_AW
- FISH
- UNKNOWN

**priceStudent:** Preis für Studierende

**priceEmployee:** Preis für Mitarbeitende

**pricePupil:** Preis für Schüler

**priceGuest:** Preis für Gäste

**individualRating:** Die Bewertung des Gerichts vom Nutzer

**numberOfRatings:** Die Gesamtanzahl an Bewertungen von diesem Gericht

**averageRating:** Die Durchschnittsbewertung von diesem Gericht

**lastServed:** Das Datum, an dem dieses Gericht zuletzt serviert wurde

**nextServed:** Das Datum, an dem dieses Gericht das nächste Mal serviert wird

**relativeFrequency:** Häufigkeit, wie oft ein Gericht in drei Monaten angeboten wird, was folgende Werte annehmen kann:

- NEW
- RARE
- NORMAL

**Side**

Diese Entität enthält alle zwischengespeicherten Beilagen, die einem Gericht zugeordnet werden.

Attribute:

**name:** Der Name der Beilage

**foodtype:** Gerichtstyp, der folgende Werte annehmen kann (Beschreibungen vgl. MealType):

- VEGAN
- VEGETARIAN
- BEEF
- BEEF\_AW
- PORK
- PORK\_AW
- FISH
- UNKNOWN

**priceStudent:** Preis für Studierende

**priceEmployee:** Preis für Mitarbeitende

**pricePupil:** Preis für Schüler

**priceGuest:** Preis für Gäste

**Image**

Diese Entität enthält alle Verweise zu den Bildern für Gerichte.

Attribute:

**url:** Der Link zum Bild

**MealAdditive**

Diese Entität enthält alle Zusatzstoffe, die in Gerichten enthalten sind.

Attribute:

**additive:** Zusatzstoffe, wobei es folgende Zusatzstoffe gibt (Beschreibungen vgl. Additive):

- COLORANT
- PRESERVING\_AGENTS
- ANTIOXIDANT\_AGENTS
- FLAVOUR\_ENHANCER
- PHOSPHATE
- SURFACE\_WAXED
- SULPHUR
- ARTIFICALLY\_BLACKENED\_OLIVES
- SWEETENER
- LAXATIVE\_IF\_OVERUSED
- PHENYLALANINE
- ALCOHOL
- PRESSED\_MEET

- GLAZING\_WITH\_CACAO
- PRESSED\_FISH

### **MealAllergen**

Diese Entität enthält alle Allergene, die in Gerichten enthalten sind.

Attribute:

**allergen:** Allergen, wobei es folgende Allergene gibt (Beschreibungen vgl. Allergen):

- CA
- DI
- EI
- ER
- FI
- GE
- HF
- HA
- KA
- KR
- LU
- MA
- ML
- PA
- PE
- PI
- QU
- RO
- SA
- SE
- SF
- SN
- SO
- WA
- WE
- WT

### **SideAdditive**

Diese Entität enthält alle Zusatzstoffe, die in Gerichten enthalten sind.

Attribute:

**additive:** Zusatzstoffe, wobei es folgende Zusatzstoffe gibt (Beschreibungen vgl. Additive):

- COLORANT
- PRESERVING\_AGENTS
- ANTIOXIDANT\_AGENTS
- FLAVOUR\_ENHANCER

- PHOSPHATE
- SURFACE\_WAXED
- SULPHUR
- ARTIFICALLY\_BLACKENED\_OLIVES
- SWEETENER
- LAXATIVE\_IF\_OVERUSED
- PHENYLALANINE
- ALCOHOL
- PRESSED\_MEET
- GLAZING\_WITH\_CACAO
- PRESSED\_FISH

### **SideAllergen**

Diese Entität enthält alle Allergene, die in Gerichten enthalten sind.

Attribute:

**allergen:** Allergen, wobei es folgende Allergene gibt (Beschreibungen vgl. Allergen):

- CA
- DI
- EI
- ER
- FI
- GE
- HF
- HA
- KA
- KR
- LU
- MA
- ML
- PA
- PE
- PI
- QU
- RO
- SA
- SE
- SF
- SN
- SO
- WA
- WE
- WT

**Favorites**

Diese Entität speichert alle Favoriten des Nutzers.

Attribute:

**lastDate:** Das Datum, an dem das Gericht zuletzt angeboten wurde.

**foodtype:** Gerichtstyp, der folgende Werte annehmen kann (Beschreibungen vgl. MealType):

- VEGAN
- VEGETARIAN
- BEEF
- BEEF\_AW
- PORK
- PORK\_AW
- FISH
- UNKNOWN

**priceStudent:** Preis für Studierende

**priceEmployee:** Preis für Mitarbeitende

**pricePupil:** Preis für Schüler

**priceGuest:** Preis für Gäste



## 4.3 API-Schema

Für die Kommunikation zwischen dem App-Client und dem Server wird eine GraphQL-API verwendet. Im Folgenden wird diese API sowie die Daten, die angefragt werden können, beschrieben.

### 4.3.1 Authentifizierung

Für die Authentifizierung des Nutzers werden sowohl ein API-Schlüssel sowie eine Client-ID als Werte im HTTP Header an den Server übertragen. Dieser verwendet diese beiden Werte im Hintergrund, um zu überprüfen, ob die empfangene Anfrage berechtigt ist Änderungsanfragen durchzuführen bzw. authentifiziert den Client darüber, um nutzerbezogene Daten zuzuordnen.

### 4.3.2 Typen

Im Folgenden werden die einzelnen Datentypen, sowie deren Attribute beschrieben, die über die GraphQL-API abgerufen werden können.

#### Main

Dieser Typ repräsentiert ein Hauptgericht einer Linie. Das Hauptgericht ist hierbei unabhängig vom Datum, an welchem dieses angeboten wird sondern stellt lediglich das Gericht an sich dar.

Attribute:

**id:** Der Identifikator des Hauptgerichtes.

**name:** Der Name des Hauptgerichtes.

**price:** Die Preise des Gerichtes jeweils für die vier Personengruppen Studenten, Mitarbeiter, Schüler und Gäste. (Price)

**mealType:** Der Typ des Gerichtes. (MealType)

**allergens:** Eine Liste an Allergenen, die im Gericht enthalten sind. (Allergen)

**additives:** Eine Liste an Zusatzstoffen, die im Gericht enthalten sind. (Additive)

**ratings:** Werte zu den Bewertungen, die von Nutzern für das Gericht abgegeben wurden. (Ratings)

**images:** Eine Liste an Bildern, die von Nutzern für das Gericht hinzugefügt wurden. (Image)

**statistics:** Eine Sammlung an statistischen Werten zu dem Gericht. (MealStatistics)

**sides:** Eine Liste an Beilagen zu dem Hauptgericht. (Side)

**line:** Die Linie an der das Gericht angeboten wird. (Line)

#### Side

Dieser Typ repräsentiert eine Beilage, die zu einem Hauptgericht angeboten werden kann.

**id:** Identifikator der Beilage.

**name:** Name der Beilage.

**price:** Die Preise der Beilage jeweils für die vier Personengruppen Studenten, Mitarbeiter, Schüler und Gäste. (Price)

**mealType:** Der Typ der Beilage. (MealType)

**allergens:** Eine Liste an Allergenen, die im Gericht enthalten sind. (Allergen)

**additives:** Eine Liste an Zusatzstoffen, die im Gericht enthalten sind. (Additive)

### MealStatistics

Dieser Typ repräsentiert eine Sammlung statistischer Daten über ein bestimmtes Hauptgericht.

**lastServed:** Das Datum, an dem das Gericht zuletzt angeboten wurde.

**nextServed:** Das Datum, an dem das Gericht das nächste Mal angeboten wird.

**relativeFrequency:** Ein Prozentwert, der angibt mit welcher relativen Häufigkeit das Gericht angeboten wird.

### Canteen

Dieser Typ repräsentiert eine Mensa. Eine Mensa besteht aus mindestens einer Speiseausgabe.

**id:** Identifikator der Mensa.

**name:** Name der Mensa.

**lines:** Eine Liste an Speiseausgaben dieser Mensa. (Line)

### Line

Dieser Typ repräsentiert eine Speiseausgabe einer Mensa. An einer Speiseausgabe werden ein oder mehrere Hauptgerichte angeboten.

**id:** Identifikator der Speiseausgabe.

**name:** Name der Speiseausgabe.

**meals(date):** Eine Liste an Hauptgerichten, die an dieser Speiseausgabe angeboten werden. Es wird eine leere Liste zurückgegeben wenn die Mensa am entsprechenden Tag geschlossen ist. Außerdem kann ein Null-Wert zurückgegeben werden, wenn für den entsprechenden Tag noch keine Daten zur Verfügung stehen. (Main)

**canteen:** Die Mensa, in der sich diese Speiseausgabe befindet. (Canteen)

### Price

Dieser Typ repräsentiert die Preise zu einem Gericht jeweils aufgegliedert nach Studenten, Mitarbeitern, Gästen und Schülern.

**student:** Der Preis für Studenten.

**employee:** Der Preis für Mitarbeiter.

**guest:** Der Preis für Gäste.

**pupil:** Der Preis für Schüler.

### Ratings

Dieser Typ repräsentiert Werte zu den Bewertungen, die von Nutzern zu einem Hauptgericht abgegeben wurden.

**averageRating:** Die durchschnittliche Bewertung, die von allen Nutzern abgegeben wurde.

**ratingsCount:** Die Anzahl an Bewertungen die zu dem entsprechenden Gericht bereits abgegeben wurden.

**personalRating:** Die eigene Bewertung, falls der Nutzer über eine Client-ID identifiziert wurde und bereits eine Bewertung abgegeben hat.

### Image

Dieser Typ repräsentiert ein Bild, welches von einem Nutzer zu einem Hauptgericht hinzugefügt wurde.

**id:** Der Identifikator des Bildes.

**url:** Die URL unter der das Bild abgerufen werden kann.

**rank:** Der vom Server berechnete Bildrang.

**upvotes:** Die Anzahl der von Nutzern, die das Bild als hilfreich markiert haben.

**downvotes:** Die Anzahl der von Nutzern, die das Bild als nicht hilfreich markiert haben.

**personalUpvote:** Falls der Nutzer über eine Client-ID identifiziert wurde, gibt dieser Wert an ob der Nutzer das Bild bereits als hilfreich markiert hat.

**personalDownvote:** Falls der Nutzer über eine Client-ID identifiziert wurde, gibt dieser Wert an ob der Nutzer das Bild bereits als nicht hilfreich markiert hat.

### MealType

Dieses Enum definiert die möglichen Typen eines Gerichtes.

- **VEGAN:** Das Gericht ist Vegan.
- **VEGETARIAN:** Das Gericht ist Vegetarisch.
- **BEEF:** Das Gericht enthält Rindfleisch.
- **BEEF\_AW:** Das Gericht enthält regionales Rindfleisch aus artgerechter Tierhaltung.
- **PORK:** Das Gericht enthält Schweinefleisch:
- **PORK\_AW:** Das Gericht enthält regionales Schweinefleisch aus artgerechter Tierhaltung.
- **FISH:** Das Gericht enthält Fisch.
- **UNKNOWN:** Das Gericht kann keiner der vorherigen Kategorien zugeordnet werden.

### Allergen

Dieses Enum definiert die möglichen Allergene, die in einem Gericht enthalten sein können.

- **CA:** Das Gericht enthält Cashewnüsse.
- **DI:** Das Gericht enthält Dinkel und Gluten aus Dinkel.
- **EI:** Das Gericht enthält Eier.
- **ER:** Das Gericht enthält Erdnüsse.
- **FI:** Das Gericht enthält Fisch.
- **GE:** Das Gericht enthält Gerste und Gluten aus Gerste.
- **HF:** Das Gericht enthält Hafer und Gluten aus Hafer.
- **HA:** Das Gericht enthält Haselnüsse.
- **KA:** Das Gericht enthält Kamut und Gluten aus Kamut.
- **KR:** Das Gericht enthält Krebstiere.
- **LU:** Das Gericht enthält Lupine.
- **MA:** Das Gericht enthält Mandeln.
- **ML:** Das Gericht enthält Milch / Laktose.
- **PA:** Das Gericht enthält Paranüsse.
- **PE:** Das Gericht enthält Pekannüsse.
- **PI:** Das Gericht enthält Pistazie.

- QU: Das Gericht enthält Queenslandnüsse / Macadamianüsse.
- RO: Das Gericht enthält Roggen und Gluten aus Roggen.
- SA: Das Gericht enthält Sesam.
- SE: Das Gericht enthält Sellerie.
- SF: Das Gericht enthält Schwefeldioxid / Sulfid.
- SN: Das Gericht enthält Senf.
- SO: Das Gericht enthält Soja.
- WA: Das Gericht enthält Walnüsse.
- WE: Das Gericht enthält Weizen und Gluten aus Weizen.
- WT: Das Gericht enthält Weichtiere.

### Additive

Dieses Enum definiert die möglichen Zusatzstoffe, die in einem Gericht enthalten sein können.

- COLORANT: Das Gericht enthält Farbstoffe.
- PRESERVING\_AGENTS: Das Gericht enthält Konservierungsstoffe.
- ANTIOXIDANT\_AGENTS: Das Gericht enthält Antioxidationsmittel.
- FLAVOUR\_ENHANCER: Das Gericht enthält Geschmacksverstärker.
- PHOSPHATE: Das Gericht enthält Phosphat.
- SURFACE\_WAXED: Das Gericht enthält eine gewachste Oberfläche.
- SULPHUR: Das Gericht ist geschwefelt.
- ARTIFICALLY\_BLACKENED\_OLIVES: Das Gericht enthält künstlich geschwärzte Oliven.
- SWEETENER: Das Gericht enthält Süßungsmittel.
- LAXATIVE\_IF\_OVERUSED: Das Gericht kann bei übermäßigem Verzehr abführend wirken.
- PHENYLALANINE: Das Gericht enthält eine Phenylalaninquelle.
- ALCOHOL: Das Gericht kann Restalkohol enthalten.
- PRESSED\_MEAT: Das Gericht ist aus Fleischstücken zusammengefügt.
- GLAZING\_WITH\_CACAO: Das Gericht enthält eine Kakaohaltige Fettglasur.
- PRESSED\_FISH: Das Gericht ist aus Fischstücken zusammengefügt.

### ReportReason

Dieses Enum definiert die möglichen Gründe, für die ein Bild gemeldet werden kann.

- OFFENSIVE: Das Bild ist anstößig.
- ADVERT: Das Bild enthält Werbung.
- NO\_MEAL: Das Bild zeigt kein Gericht.
- WRONG\_MEAL: Das Bild zeigt nicht das richtige Gericht.
- VIOLATES\_RIGHTS: Das Bild verletzt die Rechte anderer Personen.
- OTHER: Das Bild wurde aufgrund eines anderen als den oben genannten Gründen gemeldet.

### 4.3.3 Querys

Im Folgenden werden die Möglichen Abfragen (Querys), die an die GraphQL API gesendet werden können, beschrieben.

**getCanteens**

Diese Abfrage gibt eine Liste aller Mensen zurück. (Canteen)

**getCanteen(id)**

Diese Abfrage gibt eine spezifische Mensa zu der angegebenen ID zurück. Existiert keine Mensa zur angegebenen ID wird ein Null-Wert zurück gegeben. (Canteen)

**getMeal(mealId, lineId, date)**

Diese Abfrage gibt das Hauptgericht, mit der angegebenen ID, mit den Preisen und Beilagen, mit welchen das Gericht, am angegebenen Datum, an der angegebenen Linie, angeboten wurde, zurück. Existiert das Hauptgericht nicht, oder wurde es nicht am angegebenen Datum, an der angegebenen Linie angeboten, wird ein Null-Wert zurückgegeben. (Main)

### 4.3.4 Mutations

Im Folgenden werden die möglichen Änderungsanfragen (Mutations), die an die GraphQL API gesendet werden können, beschrieben.

**addImage(mealId, imageUrl)**

Diese Abfrage fügt dem angegebenen Hauptgericht ein Bild hinzu. Als Link wird der Flickr Link des Bildes verwendet über welchen die entsprechenden Daten im Backend von Flickr abgerufen und gespeichert werden können. Falls das Gericht nicht existiert, der Link nicht zu Flickr führt bzw. die Lizenz des Bildes nicht „Creative Commons“ ist oder ein anderer Fehler beim Hinzufügen des Bildes aufgetreten ist wird eine Fehlermeldung zurückgegeben. Ist das hinzufügen des Bildes erfolgreich, wird ein Wahr-Wert zurückgegeben

**setRating(mealId, rating)**

Diese Abfrage fügt dem angegebenen Hauptgericht eine Bewertung für den authentifizierten Nutzer hinzu oder verändert eine bestehende. Falls das Gericht nicht existiert oder ein anderer Fehler beim Hinzufügen der Bewertung aufgetreten ist wird eine Fehlermeldung zurückgegeben. Ist das Hinzufügen oder Verändern der Bewertung erfolgreich, wird ein Wahr-Wert zurückgegeben.

**addUpvote(imageId)**

Diese Abfrage fügt dem angegebenen Bild ein Upvote von dem authentifizierten Nutzer hinzu. Falls das Bild nicht existiert oder ein anderer Fehler beim Hinzufügen des Upvotes aufgetreten ist wird eine Fehlermeldung zurückgegeben. Ist das Hinzufügen des Upvotes erfolgreich, wird ein Wahr-Wert zurückgegeben.

**removeUpvote(imageId)**

Diese Abfrage entfernt den Upvote für das angegebenen Bild von dem authentifizierten Nutzer. Falls das Bild nicht existiert oder ein anderer Fehler beim Entfernen des Upvotes aufgetreten ist wird eine Fehlermeldung zurückgegeben. Ist das Entfernen des Upvotes erfolgreich, wird ein Wahr-Wert zurückgegeben.

**addDownvote(imageld)**

Diese Abfrage fügt dem angegebenen Bild ein Downvote von dem authentifizierten Nutzer hinzu. Falls das Bild nicht existiert oder ein anderer Fehler beim Hinzufügen des Downvotes aufgetreten ist wird eine Fehlermeldung zurückgegeben. Ist das Hinzufügen des Downvote erfolgreich, wird ein Wahr-Wert zurückgegeben.

**removeDownvote(imageld)**

Diese Abfrage entfernt den Downvote für das angegebenen Bild von dem authentifizierten Nutzer. Falls das Bild nicht existiert oder ein anderer Fehler beim Entfernen des Downvote aufgetreten ist wird eine Fehlermeldung zurückgegeben. Ist das Entfernen des Downvote erfolgreich, wird ein Wahr-Wert zurückgegeben.

**reportImage(imageld, reason)**

Diese Abfrage fügt einen Meldeantrag für das angegebene Bild von dem authentifizierten Nutzer hinzu. Falls das Bild nicht existiert, der Meldeantrag nicht erstellt werden konnte oder ein anderer Fehler aufgetreten wird eine Fehlermeldung zurückgegeben. Ist das Hinzufügen des Meldeantrags erfolgreich, wird ein Wahr-Wert zurückgegeben.

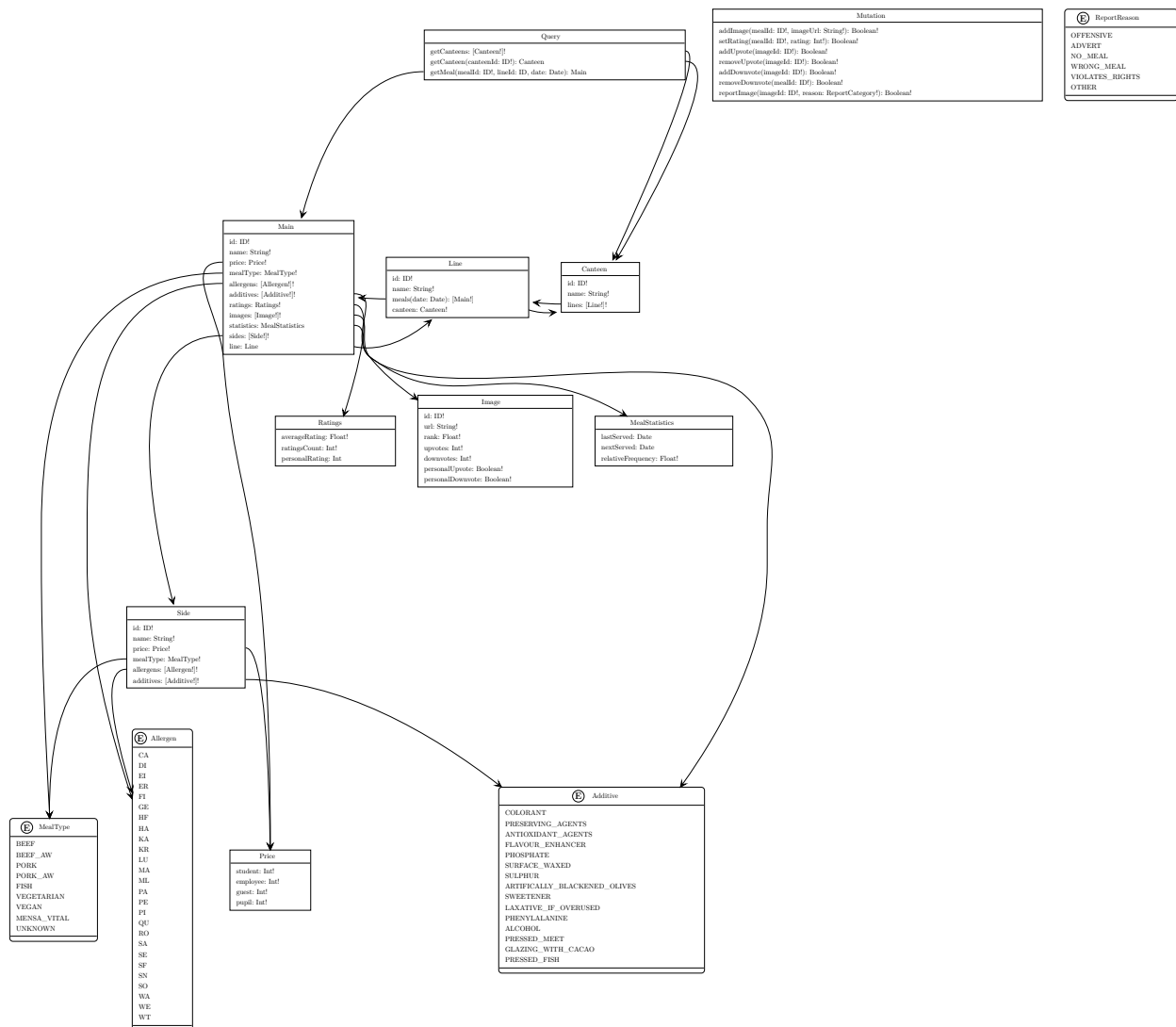


Abbildung 4.3: Schema der GraphQL-API

## Kapitel 5

# Glossar

**Asynchrone Programmierung in Dart** Eine asynchrone Methode gibt in Dart immer einen Future-Datentyp zurück. Durch `await` kann auf das Ergebnis einer asynchronen Methode gewartet werden.

**automatisches System** Logik, die entscheidet, ob ein Bild nach einem Meldeantrag direkt ausgeblendet, und damit Clients nicht mehr angezeigt wird.

**ChangeNotifier** Der `ChangeNotifier`<sup>1</sup> ist eine aus der Provider-Library von Dart bereitgestellte Klasse, die ihre Listener über Veränderungen von Daten informieren kann. Der `ChangeNotifier` ist eine Form von `Observable`.

**Data** Schicht der Backend-Architektur, die für die Interaktion mit der Außenwelt zuständig ist.

**Enumerationstyp** Datentyp, der mehrere eigens spezifizierte Werte annehmen kann.

**Flickr** Bildhoster, bei welchem die Bilder zu Gerichten hinterlegt werden.

**Future** Ein `Future`<sup>2</sup>-Typ ist das Ergebnis einer asynchronen Operation. Der Datentyp wird direkt zurückgegeben und evtl. zu einem späteren Zeitpunkt von der asynchronen Funktion vervollständigt.

**GraphQL** Typ einer Schnittstelle, über die Client und Server miteinander über HTTP-Anfragen kommunizieren können.

**Hash** Eine nicht-umkehrbare Funktion, bei der kleine Änderungen in der Eingabe zu großen und unvorhersehbaren Änderungen in der Ausgabe resultieren. Mithilfe dieser kann die Existenz des API-Schlüssels nachgewiesen werden, ohne diesen jedoch mitzusenden.

**HTTP** Standard-Web-Kommunikation für Server. Dabei werden Anfragen über URL-Parameter formuliert.

---

<sup>1</sup><https://api.flutter.dev/flutter/foundation/ChangeNotifier-class.html>

<sup>2</sup><https://api.flutter.dev/flutter/dart-async/Future-class.html>



**Logic** Schicht der Backend-Architektur, in der sich alle anwendungsspezifische Logik befindet.

**Mutation** Typ einer GraphQL-Anfrage, die mit Seiteneffekten wie Änderungen am Datenstand behaftet ist.

**Provider** Der Provider<sup>3</sup> ist eine Form des State-Managements, das mit Flutter benutzt werden kann. Dabei wird in einem Widget, das über allen Widgets steht, die über Änderungen des ChangeNotifier benachrichtigt werden müssen. Dieser liefert eine Instanz des angegebenen ChangeNotifiers. Die Klassen, die über Änderungen vom ChangeNotifier benachrichtigt werden wollen, können die Klasse Consumer nutzen.

**Query** Typ einer GraphQL-Anfrage, die nur Daten abfragt

**Shared Preferences** Für Flutter wird ein Package `shared-preferences` zur Verfügung gestellt, dass das Speichern von Key-Value-Paaren plattformübergreifend ermöglicht.

**SMTP** Simple Mail Transfer Protocol. Protokoll zum Senden von E-Mails.

**State** Enthält den Zustand eines `StatefulWidget`. Der `State` eines `\gls {StatefulWidget}` besitzt immer eine `build(context: BuildContext)`-Methode, die ein `Widget` zurückgibt.

**StatefulWidget** Widget mit Zustand, der bei Erstellung des Widgets eingelesen werden muss oder sich während der Lebenszeit des Widgets ändern kann. Dazu gibt es eine eigene `\gls {State}`-Klasse, die hier immer `private` sind und deshalb nicht modelliert werden. Ein `StatefulWidget` besitzt eine `createState()`-Methode, die `State<SomeWidget>` zurückgibt und üblicherweise den Konstruktor ihres `State` aufruft.

**StatelessWidget** Widget ohne Zustand, das vor allem benutzt wird, wenn die das Widget von nichts anderem als die Konfigurationsinformationen abhängen. Ein `StatelessWidget` besitzt immer eine `build(context: BuildContext)`-Methode, die ein `Widget` zurückgibt.

**SwKa** Abkürzung für Studierendenwerk Karlsruhe.

**Trigger** Schicht der Backend-Architektur, in der alle zu behandelnden Ereignisse, wie API-Anfragen ausgelöst werden.

**Umgebungsvariable** Möglichkeit, einem Programm beim Start Informationen zu übergeben.

**URL** Uniform Resource Locator. Identifikator für eine Web-Ressource.

**UUID** Universally Unique Identifier, nahezu garantiert eindeutiger Identifikator der zufällig und somit dezentral vergeben werden kann.

**XML** Extensible Markup Language, maschinenlesbare Sprache zur Strukturierung von Daten.

---

<sup>3</sup><https://docs.flutter.dev/data-and-backend/state-mgmt/simple>