Karlsruhe Institute of Technology

Mensa-App

Applikation für Mitteilung und kollektiven Austausch von Speiseplaninformationen in universitären Gastronomieeinrichtungen

PSE: Qualitätssicherung

Praxis der Softwareentwicklung
Sommersemester 2023

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

25. August 2023

Inhaltsverzeichnis

1 Einleitung

Mit der Qualitätssicherung nähert sich dieses Projekt seinem Ende. Vor kurzer Zeit existierte lediglich nur eine Idee für eine Mensa-App, die nun in die Tat umgesetzt wurde. Es verbleibt nur noch, das Erreichen und Verifizieren der Ziele und Anforderungen, welche ganz zu Beginn im Pflichtenheft festgelegt wurden. Genau das ist Ziel dieses Dokuments und wird im folgenden ausführlicher dargestellt.
Das Pflichtenheft definierte mehrere Eigenschaften, die das Produkt erfüllen muss oder erfüllen kann. Zusätzlich zu den gegebenen Anforderungen wurden auch Tests definiert, die geprüft werden müssen.
Die Qualitätssicherung geht über die funktionalen Anforderungen hinaus und überprüft auch nichtfunktionale Anforderungen. So wird in diesem Dokument beispielsweise auch die Codequalität behandelt.
Zusätzlich dazu entstanden mehrere Testberichte, die die Durchführung von Integrations- und Funktionstests dokumentieren.

1.1 Kriterien

Wie schon erwähnt, wurden im Pflichtenheft mehrere Kriterien definiert, die hier aktualisiert nochmals aufgelistet sind.
Dabei sind

- ✔ alle grün markierten Kriterien **umgesetzt und getestet**.
- ✘ alle rot markierten Kriterien **nicht umgesetzt**.

Alle Musskriterien und ein sehr großer Teil der Kannkriterien konnten erfolgreich umgesetzt und validiert werden.

Musskriterien

- ✔ **[MK01]** Gerichte anzeigen
- ✔ **[MK02]** Preise für verschiedene Preisklassen anzeigen
- ✔ **[MK03]** Gerichte nach Gerichtstyp und Allergenen filtern
- ✔ **[MK04]** Bilder zu Gerichten verlinken und anzeigen
- ✔ **[MK05]** Von Nutzern verlinkte Bilder melden

Kannkriterien

- ✔ **[KK01]** Light/Darkmode in der App
- ✔ **[KK02]** Gerichte favorisieren
- ✔ **[KK03]** Gerichte mit ein bis fünf Sterne bewerten
- ✔ **[KK04]** Bilder zu Gerichten bewerten
- ✔ **[KK05]** Häufigkeit der Gerichte anzeigen
- ✘ **[KK06]** Benachrichtigungen für favorisierte Gerichte
- ✘ **[KK07]** Favoriten und Einstellungen im- und exportieren
- ✔ **[KK08]** Gerichte nach Preis, Bewertung, Favoriten und Häufigkeit filtern
- ✔ **[KK09]** Gerichte nach Linie, Bewertung, Preis und Häufigkeit sortieren

1.2 Funktionen

Im Folgenden befindet sich eine Zusammenfassung der funktionalen Anforderungen des Produkts, die im Pflichtenheft definiert wurden. Diese Anforderungen wurden aus den oben genannten Kriterien abgeleitet. Eine Funktion wurde dabei genau dann umgesetzt, wenn ihr Kriterium umgesetzt wurde.

Speiseplanansicht

- ✓ **[F101]** App starten
- ✓ **[F102]** Speiseplanansicht anzeigen
- ✓ **[F103]** Speiseplan aktualisieren
- ✓ **[F104]** Mensa wechseln
- ✓ **[F105]** Tag ändern
- ✓ **[F106]** Ansicht von Galerie zu Liste wechseln
- ✓ **[F107]** Ansicht von Liste zu Galerie wechseln
- ✓ **[F108]** Vorherigen Tag anzeigen
- ✓ **[F109]** Nächsten Tag anzeigen

Detailansicht

- ✓ **[F201]** Detailansicht für Gerichte anzeigen
- ✓ **[F202]** Detailansicht schließen
- ✓ **[F203]** Bewertung abgeben
- ✓ **[F204]** Bewertung anpassen
- ✓ **[F205]** Bild hochladen
- ✓ **[F206]** Allergene und Zusatzstoffe einblenden
- ✓ **[F207]** Allergene und Zusatzstoffe ausblenden

Bilder

- ✓ **[F301]** Galeriedialog öffnen
- ✓ **[F302]** Galeriedialog verlassen
- ✓ **[F303]** Bild melden
- ✓ **[F304]** Bilder nach rechts durchwechseln
- ✓ **[F305]** Bilder nach links durchwechseln
- ✓ **[F306]** Aufwertung zu einem Bild hinzufügen
- ✓ **[F307]** Aufwertung eines Bildes entfernen
- ✓ **[F308]** Abwertung zu einem Bild hinzufügen
- ✓ **[F309]** Abwertung eines Bildes entfernen

Filterfunktion

- ✓ **[F401]** Filterdialog öffnen
- ✓ **[F402]** Filterdialog schließen
- ✓ **[F403]** Filteroptionen auswählen
- ✓ **[F404]** Filterkonfiguration speichern
- ✓ **[F405]** Filter zurücksetzen
- ✓ **[F406]** Filter aktivieren
- ✓ **[F407]** Filter deaktivieren

Favoriten

- ✓ **[F501]** Favoritenansicht anzeigen
- ✓ **[F502]** Gericht favorisieren
- ✓ **[F503]** Gericht defavorisieren

Benachrichtigungen

- ✗ **[F701]** Benachrichtigungen aktivieren
- ✗ **[F702]** Benachrichtigungen deaktivieren
- ✗ **[F703]** Benachrichtigungszeitpunkt wählen

Einstellungen

- ✓ **[F601]** Einstellungsansicht anzeigen
- ✓ **[F602]** Farbschema einstellen
- ✓ **[F603]** Preisklasse einstellen
- ✗ **[F604]** Einstellungen und Favoriten exportieren
- ✗ **[F605]** Einstellungen und Favoriten importieren

Server

- ✓ **[F801]** Server starten
- ✓ **[F802]** Heutigen Speiseplan aktualisieren
- ✓ **[F803]** Speisepläne für die nächsten vier Wochen aktualisieren
- ✓ **[F804]** API-Anfragen beantworten
- ✓ **[F805]** API-Anfragen bezüglich Bilder beantworten
- ✓ **[F806]** Nutzerdaten aktualisieren
- ✓ **[F807]** Meldeantrag behandeln
- ✓ **[F808]** Ereignis protokollieren

Alle diese Anforderungen werden in der Sektion Abnahmetest referenziert und überprüft.

2 Unit-Tests

Mithilfe von umfangreichen Unit-Tests wurde schon während der Implementierung ein korrektes Verhalten der einzelnen Komponenten sichergestellt. Dabei wurde versucht, jede Zeile des Programmcodes durch einen Testfall abzudecken. Im folgenden Kapitel sind alle Unit-Tests mit Zugehörigkeiten und Abdeckung aufgelistet. Dabei sind Frontend und Backend getrennt aufgeführt. Außerdem behandeln das Frontend und das Backend ihre Tests unterschiedlich. Mehr dazu in den jeweiligen Sektionen.

2.1 Frontend

Im Folgenden werden die Unittests aufgelistet, die im Frontend durchgeführt wurden. Mit Hilfe des in Android-Studio eingebauten Tools wurde die Line-Coverage dieser Tests ermittelt, indem alle Tests hintereinander ausgeführt wurden (siehe Datei `allTests.dart`). Anschließend wurde aus diesen Informationen eine Line-Coverage berechnet. Dabei wurden Integrationstest nicht mit ausgeführt und sind deshalb auch nicht in der Line-Coverage enthalten. Auch wurde für die Line-Coverage automatisch generierter Code nicht betrachtet. Das betrifft den Code, der für die Kommunikation mit dem Server über GraphQL generiert wurde. Ebsenso wurde die gesamte View-Schicht für die Tests nicht betrachtet, da für Widgets keine Unit-Tests geschrieben wurden. Jedes Widget wurde manuell auf seine Funktionalität und sein Design überprüft. Dadurch wird für den betrachteten Code eine Line-Coverage von 92.63 % erreicht. Diese und folgende Angaben gelten, wenn die Datenbank während der Tests neu generiert wurde.
In der folgenden Tabelle ist die Line-Coverage je Ordner in Prozent und absolut angegeben.

Ordner	**Line Coverage**	
model/api_server	**89.1%**	246 / 276
model/database	**92.3%**	453 / 491
model/database/model	**93.5%**	272 / 291
model/local_storage	**100.0%**	60 / 60
view_model/logic/favorite	**96.0%**	30 / 31
view_model/logic/image	**100.0%**	32 / 32
view_model/logic/meal	**92.6%**	262 / 283
view_model/logic/preference	**100.0%**	33 / 33
view_model/repository/data_classes/filter	**88.5%**	69 / 78
view_model/repository/data_classes/meal	**92.4%**	115 / 124
view_model/repository/data_classes/mealplan	**93.1%**	27 / 29
view_model/repository/error_handling	**100.0%**	10 / 10
In Summe	**92.6%**	1609 / 1737

Tabelle 2: Line-Coverage im Frontend

In der folgenden Tabelle werden aus Übersichtlichkeitsgründen Gruppen mit mehreren Tests als ein Eintrag aufgeführt.

Layer	Klasse	Tests
View-Model	**CanteenTest**	test_constructor
	FilterPreferenceTest	test_constructor_with_values
		test_constructor_without_values
		test_categories
		test_frequencies
		test_setter
	ImageDataTest	test_constructor
		test_votes
	LineTest	test_constructor
	MealPlanTest	test_constructor
		test_copy_constructor_all
		test_copy_constructor_nothing_copied
	MealTest	test_constructor
		test_copy_constructor_whole_new_meal
		test_copy_constructor_nothing_copied
		test_favorite
		test_setter
		test_remove_image
	PriceTest	test_constructor
		test_get_price_by_category
	SideTest	test_constructor
		test_copy_constructor_all
		test_copy_constructor_nothing_copied
		test_equals
	FavoriteMealAccessTest	test_initialisation
		test_favorite_check
		test_adding_and_removing_meals
	ImageAccessTest	test_upvote
		test_delete_upvote
		test_downvote
		test_delete_downvote
		test_link_image
		test_report_image
	MealPlanAccessTest	test_initialisation
		test_init_with_no_canteen_and_connection
		test_filter_meals
		test_reset_filter_preferences
		test_activate_filters
		test_sorting
		test_edge_cases
		test_refresh_meal_plan
		test_get_available_canteens
		test_change_canteen
		test_get_meal
		test_change_meal_rating
	PreferencesAccessTest	test_initialisation
		test_setters
		test_init_with_no_standard_values

Layer	Klasse	Tests
Model	**SharedPreferencesTest**	test_empty_preferences
		test_client_identifier
		test_meal_plan_format
		test_color_scheme
		test_price_category
		test_canteen
		test_filter_preferences
	SqLiteAccessTest	test_favorites
		test_update_all
		test_update_meal
		test_get_canteens
		test_get_canteen_by_id
		test_clean_up
		test_remove_image
	GraphQlServerAccess	test_remove_downvote
		test_remove_upvote
		test_add_downvote
		test_add_upvote
		test_link_image
		test_report_image
		test_rate_meal
		test_date_format
		test_update_all
		test_update_canteen
		test_meal_from_id
		test_get_canteen

2.2 Backend

Die gesamte Testabdeckung im Backend liegt bei $> 93.3\%$.
Dabei wurden nur die Komponenten im `layer`-Ordner beachtet, da diese die Hauptfunktionalität des Backends beinhalten. Die übrigen Pakete sind wenig funktional und nur für das Starten und den Zusammenbau der Anwendung zuständig. Als Abdeckungsmetrik wurde dabei die Zeilenabdeckung (Line-Coverage) verwendet. Die einzelnen Tests sind nach Komponente und zugehöriger Klasse gruppiert. Eine ausführlichere Abdeckungsanalyse kann unter Codecov[1] eingesehen werden.
Die tatsächliche Testabdeckung liegt höher als angegeben, da Protokoll-Nachrichten nicht Teil der Testdurchführung waren, jedoch in die zu testenden Zeilen mit einfließen.

Komponente	Klasse	Tests	Line-Coverage
Scheduler		test_scheduling	**100.00%**
		test_double_start	
		test_not_running	
GraphQL		test_add_image	**90.98%**
		test_set_rating	
		test_image_votes	
		test_report_image	
		test_api_version	
		test_complete_request	
		test_frondend_query	
		test_get_specific_canteen	
		test_get_specific_meal	
		test_get_auth_info_null	
		test_recursive_line_canteen_ok	
		test_recursive_meal_line_ok	
		test_get_auth_info	
	GraphQLServer	test_graphql	
		test_playground	
		test_not_running	
		test_double_start	
	Util	test_auth_info_parsing	
		test_auth_info_parsing_client_only	
		test_read_static_header	
MealplanManagement		test_valid_start_update_parsing	**71.88%**
		test_valid_start_full_parsing	
	RelationResolver	test_resolve_empty_canteen	
		test_resolve_canteens	
		test_resolve_line_with_rand_dishes	
		test_is_side	
		test_average_calc	
ImageReview		test_start_image_review	**69.23%**
		test_full_review	
		test_review_image_ok	
		test_review_image_throws_error_when_checked	
		test_review_nonexistent_image	
		test_review_nonexistent_image_delete_error	

[1] `https://app.codecov.io/gh/kronos-et-al/MensaApp`

Komponente	Klasse	Tests	Line-Coverage
Command		test_new	**97.89%**
		test_report_image	
		test_add_image_upvote	
		test_add_image_downvote	
		test_remove_image_upvote	
		test_remove_image_downvote	
		test_add_image	
		test_set_meal_rating	
		test_will_be_hidden	
	Authenticator	test_auth_image_add_down	
		test_auth_image_add_up	
		test_auth_image_remove_up	
		test_auth_image_remove_down	
		test_auth_add_image	
		test_auth_rate_meal	
		test_auth_report_image	
SwKaParser		test_sort_and_parse_canteens_with_valid_urls	**95.82%**
		test_sort_and_parse_canteens_with_invalid_urls	
		test_parse	
		test_parse_all	
	SwKaLinkCreator	test_get_urls	
		test_get_all_urls	
	SwKaHtmlRequest	test_get_html_response_fail	
		test_get_html_response_no_fail	
		test_get_html_strings_response_fail	
		test_get_html_strings_response_no_fail	
	HTMLParser	test_1	
		test_normal	
		test_no_meal_data	
		test_no_mealplan_shown	
		test_mensa_moltke	
		test_not_a_canteen	
		test_not_a_canteen_de	
		test_canteen_closed_de	
		test_canteen_closed	
		test_invalid	

Komponente	Klasse	Tests	Line-Coverage
Database	RequestDataAccess	test_get_canteen	**100.00%**
		test_get_canteens	
		test_get_line	
		test_get_lines	
		test_get_meal	
		test_get_meals	
		test_get_sides	
		test_get_visible_images	
		test_get_personal_rating	
		test_get_personal_upvote	
		test_get_personal_downvote	
		test_get_additives	
		test_get_allergens	
	MealplanManagementDataAccess	test_dissolve_relations	
		test_get_similar_canteen	
		test_get_similar_line	
		test_get_similar_meal	
		test_get_similar_side	
		test_add_to_plan	
		test_insert_food	
		test_insert_canteen	
		test_insert_line	
		test_update_food	
		test_update_canteen	
		test_update_line	
		test_update_meal	
		test_update_side	
		test_insert_meal	
		test_insert_side	
		test_add_meal_to_plan	
		test_add_side_to_plan	
	ImageReviewDataAccess	test_get_images_for_date	
		test_get_unvalidated_images_for_next_week	
		test_get_old_images	
		test_delete_image	
		test_mark_as_checked	
	CommandDataAccess	test_get_image_info	
		test_hide_image	
		test_add_report	
		test_add_upvote	
		test_add_downvote	
		test_override_votes	
		test_remove_upvote	
		test_remove_downvote	
		test_link_image	
		test_add_rating	
		test_get_api_keys	
	DataAccessFactory	test_factory	

Komponente	Klasse	Tests	Line-Coverage
FlickrApi		test_valid_decode	**87.74%**
		test_invalid_decode	
		test_valid_determine_short_photo_id	
		test_valid_determine_long1_photo_id	
		test_valid_determine_long2_photo_id	
		test_invalid_determine_photo_id	
		test_empty_determine_photo_id	
		test_arc	
		test_validate_url	
		test_check_license	
		test_check_existence	
	ApiRequest	test_valid_request_sizes	
		test_invalid_request_sizes	
		test_valid_error_request	
		test_invalid_license_request	
		test_valid_check_license_request	
		test_error_check_license_invalid_photo	
	JsonParser	test_valid_get_size	
		test_fallback_get_size	
		test_invalid_get_size	
		test_valid_check_license	
		test_valid_check_old_license	
		test_invalid_check_old_license	
		test_valid_parse_error	
		test_invalid_parse_error	
Mail		test_get_report	**97.62%**
		test_try_notify_admin_image_report	
		test_notify_admin_iamge_report	

3 Codequalität

3.1 Qualitätsmerkmale des Frontends

3.1.1 Linting

Um auf hohe Code-Qualität zu achten, wurden unter anderem Linter-Regeln eingehalten. Dazu wurde „Dart Analysis“ mit `flutter_lints`[2] genutzt. `flutter_lints` nutzt die Dart Regelgruppen[3] `core` und `recommended`. Dabei wurde automatisch generierter Code nicht betrachtet bzw. analysiert. Auch wurden einige Regeln ausgeschaltet. Dazu gehört die Regel, die besagt, dass Dart-Dateien in Snake-Case geschreiben sein sollen, was Dart-Konvention ist. Wir haben uns allerdings dagegen entschieden, die Dateinamen in Snake-Case zu schreiben, weshalb diese Regel ausgeschaltet wurde.
Die Linter-Regeln beinhalten:

- Aufzeigen ungenutzter Variablen und Methoden
- Aufzeigen, falls Code nie erreicht werden kann (also nicht eintretenden Null-Checks oder unerreichbare if-Bedingungen)
- Wenn möglich konstante Widgets oder Variablen benutzen

Vereinzelt wurden die Regeln beabsichtigt aus verschiedenen Gründen übergangen. Das konnte mehrere Gründe haben:

- unbenutzte Methoden: Es gibt einige Methoden, die für die Datenbank nicht benutzt wurden, im Zuge späterer Erweiterungen aber evtl. benötigt werden.
- unbenutzte Variablen: Für die Synchronisation von verschiedenen Providern ist es wichtig, dass die Provider initialisiert sind, auch wenn sie nicht genutzt werden.
- `context` wurde vereinzelt nach dem Warten auf eine asynchrone Methode genutzt. Dabei wurde in den durch den Linter aufgezeigten Stellen stehts geprüft, ob der Kontext noch aktuell ist

3.1.2 Code-Review

Außerdem wurde der Code vor dem Mergen von einer Person reviewed, die ein den Code nicht selbst geschrieben hat, aber trotzdem ein Verständnis von Dart, Flutter und der angedachten Architektur hatte.

[2] `https://pub.dev/packages/flutter_lints`
[3] `https://pub.dev/packages/lints`

3.2 Qualitätsmerkmale des Backends

3.2.1 Linting

Schon während der Entwicklung wurde auf eine hohe Code-Qualität geachtet. Dazu wurden zum einen sehr strenge Linter-Regeln umgesetzt. Hierzu wurde „Clippy“[4] mit den Regelgruppen `pendantic` und `nursery` verwendet. Nur in Ausnahmefällen wurden einzelne Linter-Regeln für einzelne Codeabschnitte deaktiviert, falls diese aufgrund von äußeren Gegebenheiten wie Eigenheiten mancher Bibliotheken oder anderen Gründen als akzeptabel erachtet wurden. In letzteren Fällen wurden die Gründe vor Ort im Code dokumentiert.
Die Linter-Regeln beinhalten:

- zu komplizierter Code, der auch einfacher geschrieben werden kann
- Code, der zwar kompiliert, höchstwahrscheinlich aber nicht den Intentionen des Programmierers entspricht
- Code, der nicht dem für Rust typischen Vorgehen und Stil entspricht
- Code, der offensichtliche Performance-Nachteile mit sich bringt und leicht effizienter geschrieben werden kann
- Kommentare zur Dokumentation müssen für alle öffentlichen Klassen, Methoden und Attribute vorhanden sein und mögliche Fehler als Rückgabewerte müssen beschreiben sein.

3.2.2 Code-Reviews

Darüber hinaus wurden Code-Reviews durchgeführt, bevor Änderungen auf den aktuellen Arbeitsbranch gemerged wurden. Dabei wurden alle Änderungen von einer Person durchgeschaut, welche nicht an den Änderungen beteiligt war, aber dennoch ein gutes Verständnis der Backend-Architektur und Programmierung in Rust hatte. Bei den Reviews wurde unter anderem auf folgendes geachtet:

- kein Werfen von Laufzeitfehlern nach der Startsequenz des Servers führt
- idiomatische Verwendung der Programmiersprache
- gute Lesbarkeit des Codes sprechende Bezeichner und Aufteilung komplexen Verhaltens in mehrere Funktionen
- kein Neuimplementieren von schon in der Standardbibliothek vorhandenen Verhaltens

3.2.3 CI

Um die Einhaltung dieser Qualitätsstandards zu gewährleisten, wurde Continous-Integration und Regeln für Pull-Requests verwendet. Dabei wurde folgendes automatisiert überprüft, bevor Änderungen auf den aktuellen Arbeitsbranch gemerged wurden:

- Einhaltung aller Linter-Regeln
- Korrekte Formatierung des Codes
- Compilierfähigkeit des Codes
- Erfolgreiches Ablaufen aller Tests
- Vorhandensein eines Code-Reviews

[4] `https://github.com/rust-lang/rust-clippy`

4 Integrations-Tests

Um das korrekte Zusammenspiel der einzelnen Komponenten sicherzustellen, werden im folgenden Integrationstests nach dem Bottom-Up-Prinzip protokolliert. Dabei wird jede Komponente (bis auf die der untersten Schicht) mit allen darunterliegenden Komponenten auf korrekte Zusammenarbeit getestet. Hierbei ist es nicht das Ziel jegliche Funktionalität abzudecken (dies ist Aufgabe der Unit-Tests), sondern lediglich eine grundlegende Zusammenarbeit zu überprüfen.
In einigen Fällen ist es zu aufwändig hierfür „vollautomatische“ Tests zu erstellen, welche lediglich aus einem eigenständigen Programmcode bestehen, da dies oftmals eine Gegenimplementierung im Frontend vorhandenen Codes im Backend benötigen würde. Daher werden diese Tests „halbautomatisch“ oder „manuell“ durchgeführt. Halbautomatische Tests bestehen zum Teil aus einem Stück Programmcode, jedoch geschehen etwaige Vorbereitungen und die Überprüfung der Erfolgsbedingung manuell. Um die Abläufe der Tests dennoch nachvollziehen und wiederholen zu können, werden diese halbautomatischen Tests entsprechend ausführlich protokolliert und dokumentiert.

4.1 Integrationen des Backends

Für die Integrationstests im Backend wurden überwiegend halbautomatische Tests verwendet. Nach den Tests wurde zum Beispiel der Datenbankinhalt per Hand überprüft, da es die Tests unnötig komplex gemacht hätte, dies zu automatisieren.

Integration: Command

Kurzbeschreibung: Testen der Zusammenarbeit der Command-Schnittstelle mit der darunterliegenden Schicht.
Alle betroffenen Komponenten:

- Code `tests/command.rs`
- Komponente `Command`
- Komponente `Mail`
- Komponente `Database`
- Komponente `FlickrApi`
- Tool `PostgreSQL-Datenbank + pgAdmin`

Getestete Schnittstellen:

+ **Betroffene Methode:** `add_image()`

  Vorbereitung:

  1. API-Schlüssel in der Datenbank hinterlegt
  2. Gericht in Datenbank hinterlegt
  3. Bild bei Bildhoster hinterlegt (mit CC0 Lizenz)
  4. Hash zur Authentifizierung berechnet

  Durchführung:

  1. `test_add_image()` ausführen
  2. In der Datenbank nach neuem Bild schauen

3. Link des Bildes auf Korrektheit überprüfen

Ergebnis: Der Test wurde erfolgreich ausgeführt. Neues Tupel für Bild wurde in Datenbank mit korrektem Link hinterlegt.

+ **Betroffene Methode:** `report_image()`

Vorbereitung:

1. API-Schlüssel in der Datenbank hinterlegt
2. Gericht mit Bild in Datenbank hinterlegt
3. Hash zur Authentifizierung berechnet

Durchführung:

1. `test_report_image()` ausführen
2. In der Datenbank die Existenz eines neuen Meldeantrags überprüfen
3. Empfang einer Administratoren-E-Mail überprüfen

Ergebnis: Der Test wurde erfolgreich ausgeführt. Ein Eintrag für den Meldeantrag ist in der Datenbank vorhanden. Eine E-Mail dem Administrator zugestellt.

+ **Betroffene Methode:** `set_meal_rating()`

Vorbereitung:

1. API-Schlüssel in der Datenbank hinterlegt
2. Gericht in Datenbank hinterlegt
3. Hash zur Authentifizierung berechnet

Durchführung:

1. `test_meal_rating()` ausführen
2. In der Datenbank die Existenz einer neuen Gerichtsbewertung überprüfen

Ergebnis: Der Test wurde erfolgreich ausgeführt. Eine Gerichtsbewertung ist in der Datenbank vorhanden.

+ **Betroffene Methode:** `add_image_{up,down}vote()`

Vorbereitung:

1. API-Schlüssel in der Datenbank hinterlegt
2. Gericht mit Bild in Datenbank hinterlegt
3. Hash zur Authentifizierung berechnet

Durchführung:

1. `test_add_image_{up,down}vote()` ausführen
2. In der Datenbank die Existenz einer neuen / geänderten Bildbewertung überprüfen

Ergebnis:

`up`: Eine korrekte Bewertung ist in der Datenbank vorhanden.

`down`: Eine korrekte Bewertung ist in der Datenbank vorhanden.

+ **Betroffene Methode:** `remove_image_{up,down}vote()`

Vorbereitung:

1. API-Schlüssel in der Datenbank hinterlegt
2. Es wurde ein Gericht mit Bild in der Datenbank hinterlegt
3. `{up,down}`-Vote zu Bild in Datenbank hinterlegen
4. Hash zur Authentifizierung berechnet

Durchführung:

1. `test_remove_image_{up,down}vote()` ausführen
2. In der Datenbank die Nichtexistenz des `{up,down}`-Votes prüfen.

Ergebnis:

`up`: Der Upvote ist nichtmehr vorhanden.

`down`: Der Downvote ist nichtmehr vorhanden.

Getestet auf `05ca496cc71455255fd3a30dd6d30b9417beaeb` durch *Jonatan Ziegler* ✔

Integration: MealplanManagement

Kurzbeschreibung: Die `MealplanManagement`-Komponente erlangt Rohdaten der SwKa-Website durch die `SwKaParser`-Komponente. Nach Aufbereitung und Abgleichung der Daten mit der `Datenbank`-Komponente werden die Daten in die Datenbank eingepflegt. Dieser Integrationstest, testet das Zusammenspiel der oben genannten Komponenten unter unterschiedlichen Bedingungen.

Alle betroffenen Komponenten:

- Code `tests/mealplan_management.rs`
- Komponente `MealplanManagement`
- Komponente `SwKaParser`
- Komponente `Datenbank`
- Tool `PostgreSQL-Datenbank + pgAdmin`

Getestete Schnittstellen:

+ **Betroffene Methode:** `test_start_full_parsing()`

  Vorbereitung: -

  Durchführung:

  1. `test_start_full_parsing()` ausführen
  2. Die Daten in der Datenbank mit pgAdmin überprüfen

  Ergebnis: Das Einfügen der Daten in die Datenbank wurde erfolgreich ausgeführt. Alle Daten der nächsten Wochen sind in der Datenbank zu finden. Datenkonflikte wurden erfolgreich gelöst.

+ **Betroffene Methode:** `test_start_update_parsing()`

  Vorbereitung: -

  Durchführung:

  1. `test_start_update_parsing()` ausführen
  2. Die Daten in der Datenbank mit pgAdmin überprüfen

  Ergebnis: Das Einfügen der Daten in die Datenbank wurde erfolgreich ausgeführt. Alle Daten der nächsten Wochen sind in der Datenbank zu finden. Datenkonflikte wurden erfolgreich gelöst. Falls Daten am selben Tag vorhanden waren, wurden diese erwartungsgemäß überschrieben.

Getestet auf `2cd006f8f23bb5786068c24034d755bd5305f3bf` durch *Alexander Albers* ✔

Integration: ImageReview

Kurzbeschreibung: Testen der Zusammenarbeit der ImageReview-Schnittstelle mit der darunterliegenden Schicht.

Alle betroffenen Komponenten:

- Code `tests/image_reviewer.rs`
- Komponente `ImageReview`
- Komponente `Database`
- Komponente `FlickrApi`
- Tool `PostgreSQL-Datenbank + pgAdmin`

Getestete Schnittstellen:

+ **Betroffene Methode:** `start_image_review()`

  Vorbereitung:

  1. Gericht in Datenbank hinterlegt
  2. Bild bei Bildhoster hinterlegt (mit CC0 Lizenz)
  3. Bilder in Datenbank hinterlegen. Davon sollte mindestens ein Bild
     - bei Flickr gelöscht sein
     - bei Flickr vorhanden sein

     Außerdem sollten die Bilder entweder:
     - mit einem Gericht verknüpft sein, welches am aktuellen Tag serviert wird
     - mit einem Gericht verknüpft sein, welches in der nächsten Woche serviert wird und `last_verified_date` sollte länger als eine Woche her sein
     - `last_verified_date` sollte länger als eine Woche her sein

     Dabei ist zu beachten, dass aus den jeweiligen Kategorien jeweils nur 500 Bilder getestet werden. Diese sind bei den ersten Beiden nach Bildrang bestimmt und bei dem letzten nach `last_verified_date`.

  Durchführung:

  1. `start_image_review()` ausführen
  2. in der Datenbank prüfen, ob genau die Bilder darin vorhanden sind, die auf Flickr noch existieren und vorher in der Datenbank schon existierten

  Ergebnis: Der Code wurde erfolgreich ausgeführt. In der Datenbank sind nur die Bilder vorhanden, die auf Flickr noch existieren und vorher in der Datenbank schon existierten.

Getestet auf `1e7be132973cce2e7466fb7fb41b7d57c6a8e0df` durch *Peer Booken* ✔

Integration: GraphQL-Queries

Kurzbeschreibung: Vollständiges Testen der GraphQL Query-Request mit allen darunterliegenden Schichten
Alle betroffenen Komponenten:

- Komponente `GraphQL`
- Komponente `Database`
- Tool `GraphQL Playground`

Getestete Schnittstellen:

+ **Betroffene Methode:** `getCanteens()`

  Vorbereitung:

  1. Mehrere Kantinen mit mehreren Linien sind in der Datenbank hinterlegt
  2. Mindestens alle Gerichte eine Kantine sind im Speiseplan des 21.08.2023 in der Datenbank hinterlegt
  3. Manche Kantinen sind am 21.08.2023 geschlossen und bieten keine Gerichte an

  Durchführung:

  1. Backend starten
  2. GraphQL-Playground im Browser öffnen
  3. `getCanteens`-Anfrage im Browser an die API senden

  Ergebnis: Anfrage wurde erfolgreich ausgeführt. Die Mensa am Adenauerring ist geöffnet und bietet Gerichte an mehreren Linien zum ausgewählten Tag an. Manche Mensen wie beispielsweise die Mensa Moltke haben geschlossen, da sie keine Gerichte an dem ausgewählten Tag anbieten. Die Daten wurden vollständig mit der SwKa-Website abgeglichen. Keine fehlenden oder fehlerhaften Daten wurden festgestellt.

+ **Betroffene Methode:** `getCanteens()`

  Vorbereitung:

  1. Mehrere Kantinen mit mehreren Linien sind in der Datenbank hinterlegt
  2. Mindestens alle Gerichte eine Kantine sind im Speiseplan des 20.08.2023 in der Datenbank hinterlegt
  3. Alle Kantinen sollten geschlossen sein, da es sich am 20.08.2023 um einen Sonntag handelt.

  Durchführung:

  1. Backend starten
  2. GraphQL-Playground im Browser öffnen
  3. `getCanteens`-Anfrage im Browser an die API senden

  Ergebnis: Anfrage wurde erfolgreich ausgeführt. Alle Kantinen sind geschlossen. Keine Gerichte werden an den Linien der jeweiligen Kantinen angeboten.

+ **Betroffene Methode:** `getCanteen(id: UUID)`

  Vorbereitung:

  1. Eine Kantine mit mehreren Linien ist in der Datenbank hinterlegt
  2. Mindestens alle Gerichte dieser Kantine sind im Speiseplan des 21.08.2023 in der Datenbank hinterlegt
  3. Die UUID der Kantine ist bekannt

  Durchführung:

1. Backend starten
2. GraphQL-Playground im Browser öffnen
3. `getCanteen`-Anfrage mit Parameter `id` im Browser an die API senden

Ergebnis: Anfrage wurde erfolgreich ausgeführt. Die ausgewählte Kantine (Mensa am Adenauerring) hat am 21.08.2023 geöffnet und bietet mehrere Gerichte an unterschiedlichen Linien. Die Daten wurden vollständig mit der SwKa-Website abgeglichen. Keine fehlenden oder fehlerhaften Daten wurden festgestellt.

+ **Betroffene Methode:** `getMeal(mealId: UUID, lineId: UUID, date: Date)`

Vorbereitung:

1. Eine Kantine mit mindestens einer Linie ist in der Datenbank hinterlegt.
2. Diese Linie hat mindestens ein Gericht mit mehreren Beilagen am 21.08.2023 in der Datenbank hinterlegt
3. Die UUID des Gerichts, sowie der Linie sind neben dem Datum auch bekannt.

Durchführung:

1. Backend starten
2.
3. GraphQL-Playground im Browser öffnen
4. `getMeal`-Anfrage mit Parameter `mealId, lineId, date` im Browser an die API senden

Ergebnis: Anfrage wurde erfolgreich ausgeführt. Das gesuchte Gericht (Geflügel - Cevapcici) wird im Verbund mit der Linie (Linie 1) am ausgewählten Tag (21.08.2023) angeboten. Das Gericht wurde vollständig mit der SwKa-Website abgeglichen. Keine fehlenden oder fehlerhaften Daten wurden festgestellt.

+ **Betroffene Methode:** `apiVersion()`

Vorbereitung:

1. Der GraphQL-Server des Backend ist eingerichtet und gestartet

Durchführung:

1. Backend starten
2. GraphQL-Playground im Browser öffnen
3. `apiVersion`-Anfrage im Browser an die API senden

Ergebnis: Anfrage wurde erfolgreich ausgeführt. Die aktuelle API-Version (1.0) wurde zurückgegeben.

Getestet auf `b1d51f2bc0cabb6eb238dbd895e0caef6c617a6e` durch *Alexander Albers* ✔

Integration: GraphQL-Mutations

Kurzbeschreibung: Vollständiges Testen der GraphQL Mutation-Requests mit allen darunterliegenden Schichten

Alle betroffenen Komponenten:

- Code `src/main.rs`
- Code `tests/mutation.graphql`
- Komponente `GraphQL`
- Komponente `Command`
- Komponente `Mail`
- Komponente `Database`
- Komponente `FlickrApi`
- Tool `PostgreSQL-Datenbank + pgAdmin`
- Tool `GraphQL Playground`

Getestete Schnittstellen:

+ **Betroffene Methode:** `addImage`

  Vorbereitung:

  1. API-Schlüssel in Datenbank hinterlegt
  2. Gericht in Datenbank hinterlegt
  3. Bild bei Bildhoster hinterlegt (mit CC0 Lizenz)
  4. Hash zur Authentifizierung berechnet
  5. `Authorization`-Header berechnet

  Durchführung:

  1. Backend starten
  2. `addImage`-Anfrage an API senden
  3. In Datenbank nach neuem Bild schauen
  4. Link des Bildes auf Korrektheit überprüfen

  Ergebnis: Anfrage wurde erfolgreich ausgeführt. Ein neues Tupel für das betroffene Bild wurde in der Datenbank mit korrektem Link hinterlegt.

+ **Betroffene Methode:** `reportImage`

  Vorbereitung:

  1. API-Schlüssel in Datenbank hinterlegt
  2. Gericht mit Bild in Datenbank hinterlegt
  3. Hash zur Authentifizierung berechnet
  4. `Authorization`-Header berechnet

  Durchführung:

  1. Backend starten
  2. `reportImage`-Anfrage an API senden
  3. In Datenbank nach neuem Meldeantrag überprüfen
  4. Empfang einer Administratoren-E-Mail überprüfen

  Ergebnis: Anfrage wurde erfolgreich ausgeführt. Ein Eintrag für den Meldeantrag ist in der Datenbank vorhanden. Eine E-Mail wurde dem Administrator zugestellt.

+ **Betroffene Methode:** `setRating`

  Vorbereitung:

  1. API-Schlüssel in Datenbank hinterlegt
  2. Gericht in Datenbank hinterlegt
  3. Hash zur Authentifizierung berechnet
  4. `Authorization`-Header berechnet

  Durchführung:

  1. Backend starten
  2. `setRating`-Anfrage an API senden
  3. In Datenbank nach neuer Gerichtsbewertung prüfen

  Ergebnis: Anfrage wurde erfolgreich ausgeführt. Gerichtsbewertung in Datenbank vorhanden.

+ **Betroffene Methode:** `add{Up,Down}vote`

  Vorbereitung:

  1. API-Schlüssel in Datenbank hinterlegt
  2. Gericht mit Bild in Datenbank hinterlegt
  3. Hash zur Authentifizierung berechnet
  4. `Authorization`-Header berechnet

  Durchführung:

  1. Backend starten
  2. `add{Up,Down}vote`-Anfrage an API senden
  3. In der Datenbank die Existenz einer neuen oder geänderten Bildbewertung überprüfen

  Ergebnis:

  `up`: Die Anfrage wurde erfolgreich ausgeführt. Eine korrekte Bewertung ist in der Datenbank vorhanden.

  `down`: Die Anfrage wurde erfolgreich ausgeführt. Eine korrekte Bewertung ist in der Datenbank vorhanden.

+ **Betroffene Methode:** `remove{Up,Down}vote`

  Vorbereitung:

  1. API-Schlüssel in Datenbank hinterlegt
  2. Gericht mit Bild in Datenbank hinterlegt
  3. `{up,down}`-Vote zu Bild in Datenbank hinterlegen
  4. Hash zur Authentifizierung berechnet
  5. `Authorization`-Header berechnet

  Durchführung:

  1. Backend starten
  2. `remove{Up,Down}vote`-Anfrage an API senden
  3. In Datenbank nach nicht mehr existierenden `{up,down}`-Vote prüfen.

  Ergebnis:

  `up`: Die Anfrage wurde erfolgreich ausgeführt. Der Upvote ist nichtmehr vorhanden.

  `down`: Die Anfrage wurde erfolgreich ausgeführt. Der Downvote ist nichtmehr vorhanden.

Getestet auf `5f1dd0687b520ab62eb8410e90da866a2d1ebd44` durch *Jonatan Ziegler* ✔

Integration: Scheduler mit ImageReview

Kurzbeschreibung: Testen der Zusammenarbeit der Scheduler-Schnittstelle mit allen darunterliegenden Schichten.

Alle betroffenen Komponenten:

- Code `tests/image_review_scheduler.rs`
- Komponente `Scheduling`
- Komponente `ImageReview`
- Komponente `FlickrApi`
- Komponente `Database`
- Tool `PostgreSQL-Datenbank + pgAdmin`

Getestete Schnittstellen:

+ **Betroffene Methode:** `test_image_scheduling()`

  Vorbereitung:

  1. Gericht in Datenbank hinterlegt
  2. Bild bei Bildhoster hinterlegt (mit CC0 Lizenz)
  3. Bilder in Datenbank hinterlegen. Davon sollte mindestens ein Bild
     - bei Flickr gelöscht sein
     - bei Flickr vorhanden sein

     Außerdem sollten die Bilder entweder:
     - zu einem Gericht sein, welches am aktuellen Tag serviert wird
     - zu einem Gericht sein, welches in der nächsten Woche serviert wird und last_verified_date sollte länger als eine Woche her sein
     - last_verified_date sollte länger als eine Woche her sein

     Dabei ist zu beachten, dass aus den jeweiligen Kategorien jeweils nur 500 Bilder getestet werden. Diese sind bei den ersten Beiden nach Bildrang bestimmt und bei dem letzten nach last_verified_date.

  Durchführung:

  1. `test_image_scheduling()` starten
  2. Warten bis ImageReview zum ersten Mal gestartet wird
  3. Datenbankinhalt auf Bedingung 2 des Ergebnisses überprüfen
  4. Bilder wie im zweiten Punkt der Vorbereitung beschrieben in die DB einfügen
  5. Wiederhole Schritt 3-6 beliebig oft
  6. Beende den Test mit Strg+C

  Ergebnis: Alle 5 Minuten werden die nicht mehr vorhandenen Bilder aus der Datenbank gelöscht In der Datenbank sind genau die Bilder, die auf Flickr noch existieren und vorher in der Datenbank waren

Getestet auf `5f3dac82f64a4ae7757499840ec68a876b8d0b5e` durch *Peer Booken* ✔

Integration: Scheduler mit MealplanManagement

Kurzbeschreibung: Testen der Zusammenarbeit der Scheduler-Schnittstelle mit allen darunterliegenden Schichten.

Alle betroffenen Komponenten:

- Code `tests/mensa_parse_scheduling.rs`
- Komponente `Scheduling`
- Komponente `MealplanManagement`
- Komponente `Database`
- Komponente `SwKaParser`
- Tool `PostgreSQL-Datenbank + pgAdmin`

Getestete Schnittstellen:

+ **Betroffene Methode:** `test_full_mensa_parse_scheduling()`

  Vorbereitung:

  1. Datenbank muss leer sein

  Durchführung:

  1. `test_full_mensa_parse_scheduling()` starten
  2. Warten bis der Parser zum ersten Mal gestartet wird
  3. Datenbankinhalt überprüfen
  4. Zur Wiederholung Datenbank leeren
  5. Wiederhole Schritt 3-6 beliebig oft
  6. Beende Test mit Strg+C

  Ergebnis: Alle 5 Minuten werden die Gerichte für die nächsten 4 Wochen in die Datenbank eingefügt.

+ **Betroffene Methode:** `test_update_mensa_parse_scheduling()`

  Vorbereitung:

  1. Datenbank muss leer sein

  Durchführung:

  1. `test_update_mensa_parse_scheduling()` starten
  2. Warten bis der Parser zum ersten Mal gestartet wird
  3. Datenbankinhalt überprüfen
  4. Zur Wiederholung Datenbank leeren
  5. Wiederhole Schritt 3-6 beliebig oft
  6. Beende Test mit Strg+C

  Ergebnis: Alle 5 Minuten werden alle Gerichte für den heutigen Tag in die Datenbank eingefügt.

Getestet auf `5ccb8dc880d36153048f166822d298719eeb8e19` durch *Peer Booken* ✔

4.2 Integrationen des Frontends

Für die automatischen Integrationstests wurde mit `mocktail` gearbeitet. Hier wurden erst `model` und `view_model` gemeinsam automatisch gegen Test-Backend Daten getestet. Anschließend wurde die App in dem Zustand, in dem sie veröffentlicht werden soll, manuell getestet.
Bei den manuellen Test werden bei den Komponenten nur jene aufgelistet, auf welche die Integration abzielt. Bei einer normalen Nutzung der App werden immer mindestens MealData, Database und APIServer verwendet.

Integration: Preferences

Kurzbeschreibung: Bei diesem Test wird getestet, ob die Klassen PreferenceAccessTest und SharedPreferenceAccess miteinander funktionieren. Dabei wurde die Initialisierung mit und ohne Werten in den SharedPreferences und die Getter bzw. Setter getestet.
Alle betroffenen Komponenten:

- Code `test/integrity_test/PreferenceTest.dart`
- Komponente `Preferences`
- Komponente `LocalStorage`

Automatisierte Tests:

+ **Test:** Initialisierung
  Beschreibung: Initialisierung ohne Werte in Shared Preferences. Dieser Test verifiziert, dass der `clientIdentifier` gesetzt wurde und dass für `colorScheme`, `mealPlanFormat` und `priceCategory` Standardwerte zurückgegeben werden.
+ **Test:** Initialisierung with values
  Beschreibung: Initialisierung mit nicht-Standardwerten in Shared Preferences. Dieser Test verifiziert, die in den SharedPreferences gespeicherten Werte für `clientIdentifier`, `colorScheme`, `mealPlanFormat` und `priceCategory` zurückgegeben werden.
+ **Test:** test setter
  Beschreibung: Setzen anderer Werte für `clientIdentifier`, `colorScheme`, `mealPlanFormat` und `priceCategory`. Dieser Test verifiziert, dass die neuen Werte in den SharedPreferences gespeichert werden und die Getter die neuen Werte zurückgeben.

Getestet auf `5cefab3a00427cda0e09f7a2c948ff17c6e3d2f2` durch *Elena Häußler* ✔

Integration: Favorite

Kurzbeschreibung: Es wird getestet, ob ein Favorit in der Datenbank richtig hinzugefügt bzw. entfernt wird.
Alle betroffenen Komponenten:

- Code `test/integrity_test/FavoriteTest.dart`
- Komponente `Favorite`
- Komponente `Database`
- Komponente `APIServer`

Automatisierte Tests:

+ **Test:** test add favorite meal
  Beschreibung: Ein Gericht wird zu den Favoriten hinzugefügt und anschließend überprüft, ob die Favorite-Komponente danach dies auch richtig ausgibt (in Form der `isFavorite()` und `getFavorites()` Methoden) und ob die Datenbank dies auch gespeichert hat.
+ **Test:** test delete favorite meal
  Beschreibung: Ein Gericht wird zu den Favoriten entfernt und anschließend überprüft, ob die Favorite-Komponente danach dies auch richtig ausgibt (in Form der `isFavorite()` und `getFavorites()` Methoden) und ob die Datenbank dies auch gespeichert hat.

Getestet auf `5cefab3a00427cda0e09f7a2c948ff17c6e3d2f2` durch *Elena Häußler* ✔

Integration: Image

Kurzbeschreibung: Es wird getestet, ob die Komponenten beim Bewerten, Hinzufügen und Melden von Bildern richtig funktionieren. Dabei wurde gegen ein Test-GraphQL-Server getestet, der keine Daten geändert hat, weshalb nicht getestet wurde, ob die Änderungen beim Server übernommen wurden. Der Server gibt immer zurück, dass die Aktion erfolgreich war.
Alle betroffenen Komponenten:

- Code `test/integrity_test/ImageTest.dart`
- Komponente `Image`
- Komponente `API Server`
- Komponente `Database`

Automatisierte Tests:

+ **Test:** delete downvote
  Beschreibung: Es wird geprüft, ob die Nachricht bei einem erfolgreichen Übermitteln die richtige ist.
+ **Test:** delete upvote
  Beschreibung: Es wird geprüft, ob die Nachricht bei einem erfolgreichen Übermitteln die richtige ist.
+ **Test:** add upvote
  Beschreibung: Es wird geprüft, ob die Nachricht bei einem erfolgreichen Übermitteln die richtige ist.
+ **Test:** add downvote
  Beschreibung: Es wird geprüft, ob die Nachricht bei einem erfolgreichen Übermitteln die richtige ist.

+ **Test:** report image
  Beschreibung: Es wird geprüft, ob die Nachricht bei einem erfolgreichen Übermitteln die richtige ist. Zusätzlich wird getestet, ob das Bild aus der Datenbank gelöscht wurde.
+ **Test:** link image
  Beschreibung: Es wird geprüft, ob die Nachricht bei einem erfolgreichen Übermitteln die richtige ist.

Getestet auf `5cefab3a00427cda0e09f7a2c948ff17c6e3d2f2` durch *Elena Häußler* ✔

Integration: MealData

Kurzbeschreibung: Es wird getestet, ob die Komponenten beim Bewerten, Hinzufügen und Melden von Bildern richtig funktionieren. Dabei wurde gegen ein Test-GraphQL-Server getestet, der keine Daten geändert hat, weshalb nicht getestet wurde, ob die Änderungen beim Server übernommen wurden. Der Server gibt immer zurück, dass die Aktion erfolgreich war.
Alle betroffenen Komponenten:

- Code `test/integrity_test/ImageTest.dart`
- Komponente `Image`
- Komponente `API Server`
- Komponente `Database`

Automatisierte Tests:

+ **Test:** get meal plan
  Beschreibung: Es wird geprüft, ob die Methode einen Speiseplan zurückgibt.
+ **Test:** get meal
  Beschreibung: Es wird geprüft, ob die Methode das richtige Gericht zurückgibt.
+ **Test:** get available canteens
  Beschreibung: Es wird geprüft, ob die Methode kein Fehler wirft.
+ **Test:** refresh meal plan
  Beschreibung: Es wird geprüft, der Speiseplan erfolgreich aktualisiert wird.

Getestet auf `6139f782f2788feaf5757287f73bc761a12585b3` durch *Alexander Kutschera* ✔

Integration: Speisepläne anzeigen

Kurzbeschreibung: Bei diesem Test wird getestet, ob die Gerichte richtig angezeigt werden und die Fehlermeldungen für die verschiedenen Tage funktionieren. Dabei wird insbesondere getestet, ob die Daten, die in der Backend-Datenbank sind, in die App korrekt übertragen werden.
Alle betroffenen Komponenten:

- View `Speiseplanansicht`
- View `Detailansicht`
- Komponente `View`
- Komponente `MealData`
- Komponente `APIServer`
- Komponente `Database`
- Tool `PostgreSQL-Datenbank + pgAdmin`

Getestete Schnittstellen:

+ **Test:** Gerichte werden richtig angezeigt
  Vorbereitung: Die App wird gestartet.
  Durchführung: Abgleich der Gerichte in der App mit denen in der Backend-Datenbank.
  Ergebnis: Die Daten sind gleich.
+ **Test:** Tag in ca. zwei Monaten auswählen
  Vorbereitung: Die App wird gestartet.
  Durchführung: Es wird zu einem Tag in zwei Monaten navigiert.
  Ergebnis: Es wird angezeigt, dass noch keine Daten vorliegen.
+ **Test:** Tag in max. einem Monat auswählen
  Vorbereitung: Die App wird gestartet.
  Durchführung: Es wird zu einem Tag in max. einem Monat navigiert.
  Ergebnis: Es wird ein Speiseplan angezeigt oder, dass die ausgewählte Mensa geschlossen hat.
+ **Test:** Sonn- oder Feiertag auswählen
  Vorbereitung: Die App wird gestartet.
  Durchführung: Es wird zu einem Tag Sonn- oder Feiertag navigiert.
  Ergebnis: Es wird angezeigt, dass die ausgewählte Mensa an dem Tag geschlossen hat.

Getestet auf `2af9dee7503ed41fb90f6c0572edacb14a48b43b` durch *Elena Häußler* ✔

Integration: Favoriten hinzufügen und entfernen

Kurzbeschreibung: Bei diesem Test wird getestet, ob die Favoriten in der App richtig angezeigt, hinzugefügt und entfernt werden.

Alle betroffenen Komponenten:

- View `Favoritenansicht`
- View `Detailansicht`
- Komponente `View`
- Komponente `FavoriteAccess`
- Komponente `MealDataAccess`
- Komponente `Database`

Getestete Schnittstellen:

+ **Test:** Favoriten hinzufügen

  Vorbereitung: App starten und in die Detailansicht eines beliebigen Gerichts navigieren.

  Durchführung:

  1. Gericht favorisieren

  Ergebnis: Gericht wird als favorisiert angezeigt und in der Favoritenansicht angezeigt.

+ **Test:** Favoriten entfernen

  Vorbereitung: App starten und in Detailansicht eines Favorisierten Gerichts navigieren.

  Durchführung:

  1. Gericht defavorisieren

  Ergebnis: Gericht wird nicht mehr als favorisiert angezeigt und wird nicht in der Favoritenansicht angezeigt.

+ **Test:** keine Favoriten vorhanden

  Vorbereitung: App starten

  Durchführung:

  1. Alle Favoriten defavorisieren.

  Ergebnis: Es wird angezeigt, dass keine Favoriten vorhanden sind.

Getestet auf `6ab8cce47c424f594bf60b254d77c162eef2e736` durch *Elena Häußler* ✔

Integration: Light- / Darkmode

Kurzbeschreibung: Bei diesem Test wird getestet, ob der Light- und Darkmode in allen Ansichten, Dialogen und Bannern richtig angezeigt wird.

Alle betroffenen Komponenten:

- View `Speiseplanansicht (in beiden Darstellungsformen)`
- View `Favoritenansicht`
- View `Einstellungsansicht`
- View `Detailansicht`
- View `Filterdialog`
- View `Bilddialog`
- View `Hochladedialog`
- View `Bewertungsdialog`
- View `Meldedialog`
- Komponente `View`
- Komponente `Preference`
- Komponente `LocalStorage`

Getestete Schnittstellen:

+ **Test:** Ansichten und Dialoge

  Vorbereitung: App starten

  Durchführung: Navigation zu allen Ansichten und Dialogen, die oben genannte sind.

  Ergebnis: Im Darkmode werden diese dunkel und im Lightmode hell dargestellt.

+ **Test:** Banner

  Vorbereitung: App starten

  Durchführung: Aktivierung verschiedener Banner. Dazu gehören:

  1. Fehlermeldung Speiseplan aktualisieren
  2. Fehlermeldung Gericht bewerten
  3. Erfolgs- und Fehlermeldung Bild hinzufügen
  4. Erfolgs- und Fehlermeldung Bild melden

  Ergebnis: Im Dark- bzw. Lightmode werden die Banner für Fehlermeldungen mit rotem Hintergrund und die Banner für Erfolgsmeldungen mit grünem Hintergrund dargestellt.

Getestet auf `b41da769313979bc9a317bbcc6d8b7870c56fb8d` durch *Elena Häußler* ✔

Integration: Preiskategorie ändern

Kurzbeschreibung: Bei diesem Test wird getestet, ob die Preise entsprechend der eingestellten Preiskategorie richtig angezeigt werden und jenen aus der Datenbank des Servers entsprechen.
Alle betroffenen Komponenten:

- View `Speiseplanansicht (in beiden Darstellungsformen)`
- View `Einstellungsansicht`
- View `Detailansicht`
- Komponente `View`
- Komponente `Preference`
- Komponente `MealData`
- Komponente `LocalStorage`
- Tool `PostgreSQL-Datenbank + pgAdmin`

Getestete Schnittstellen:

+ **Test:** Preiskategorie ändern
  Vorbereitung:
  1. App starten
  2. In die Einstellungsansicht navigieren

  Durchführung: Preiskategorie in Mitarbeitende ändern
  Ergebnis: In der Speiseplanansicht werden die Preise der Mitarbeitende aus der Backend-Datenbank angezeigt.

Getestet auf `6ab8cce47c424f594bf60b254d77c162eef2e736` durch *Elena Häußler* ✔

Integration: Gerichte bewerten

Kurzbeschreibung: Es wird getestet, ob die App das gewünschte Verhalten beim Hinzufügen bzw. Ändern einer Bewertung zu einem Gericht vorweist.
Alle betroffenen Komponenten:

- View `Speiseplanansicht`
- View `Detailansicht`
- View `Bewertungsdialog`
- Komponente `View`
- Komponente `MealData`
- Komponente `APIServer`
- Tool `PostgreSQL-Datenbank + pgAdmin`

Getestete Schnittstellen:

+ **Test:** neue Bewertung hinzufügen
  Vorbereitung:
  1. App starten
  2. In Detailansicht zu Gericht navigieren, das noch nicht bewertet wurde

Durchführung: Gericht bewerten

Ergebnis:

1. Bewertung wird angezeigt
2. Bewertungseintrag wurde in die Backend-Datenbank hinzugefügt

+ **Test:** Bewertung aktualisieren

Vorbereitung:

1. App starten
2. In Detailansicht zu Gericht navigieren, das noch nicht bewertet wurde

Durchführung: Gericht bewerten

Ergebnis:

1. Bewertung wird angezeigt
2. Bewertungseintrag wurde in die Backend-Datenbank aktualisiert

+ **Test:** Bewertung aktualisierten gescheitert

Vorbereitung:

1. App starten
2. Flugmodus aktivieren
3. In Detailansicht zu Gericht navigieren, das noch nicht bewertet wurde

Durchführung: Gericht bewerten

Ergebnis:

1. Banner mit Fehlermeldung wird angezeigt.
2. Bewertungseintrag wurde in die Backend-Datenbank hat sich nicht geändert.

Getestet auf `ac80996a067e14c75b00f39cdd3cf0c7d4f979da` durch *Elena Häußler* ✔

Integration: Bild

Kurzbeschreibung: Es wird getestet, ob die App das gewünschte Verhalten beim Verlinken, Bewerten und Melden eines Bildes vorweist und ob die angezeigten Bewertungen mit denen des Servers übereinstimmen.
Alle betroffenen Komponenten:

- View `Speiseplanansicht`
- View `Detailansicht`
- View `Bilddialog`
- View `Hochladedialog`
- View `Meldedialog`
- Komponente `View`
- Komponente `MealData`
- Komponente `APIServer`
- Tool `PostgreSQL-Datenbank + pgAdmin`

Getestete Schnittstellen:

+ **Test:** Bild hinzufügen

Vorbereitung:

1. App starten

2. In die Detailansicht eines Gerichts navigieren

Durchführung: Link zu Bild mit akzeptierender Lizenz über den Hochladedialog einfügen.

Ergebnis:

1. Banner mit Erfolgsmeldung wird angezeigt
2. Das Bild wird nach aktualisieren des Speiseplans angezeigt.
3. Das Bild ist in der Backend-Datenbank mit dem Gericht verknüpft.

+ **Test:** Bild bewerten

Vorbereitung:

1. App starten
2. In den Bilddialog eines Bildes navigieren

Durchführung: Beliebige Bewertung hinzufügen

Ergebnis:

1. Bewertung wird inkl. Veränderung der Anzahl der Bewertungen geändert.
2. Die Bewertung des Bildes wird in der Backend-Datenbank aktualisiert und bei keiner Bewertung wird der Eintrag gelöscht.

+ **Test:** Bild ohne Verbindung zum Server bewerten

Vorbereitung:

1. App starten
2. In den Bilddialog eines Bildes navigieren

Durchführung: Beliebige Bewertung hinzufügen

Ergebnis:

1. Bewertung wird inkl. Veränderung der Anzahl der Bewertungen bliebt gleich.
2. Die Bewertung des Bildes wird in der Backend-Datenbank nicht aktualisiert.

+ **Test:** Bild melden

Vorbereitung:

1. App starten
2. In den Bilddialog eines Bildes navigieren

Durchführung: Bild mit der Kategorie "falsches Gericht"melden.

Ergebnis:

1. Ein Banner mit einer Erfolgsmeldung wird angezeigt und das Bild wird ausgeblendet.
2. Die Meldung des Bildes ist mit entsprechender Kategorie in der Backend-Datenbank eingetragen.
3. Der Administrator bekommt eine entsprechende E-Mail, die über die Meldung des Bildes informiert.

Getestet auf `ac80996a067e14c75b00f39cdd3cf0c7d4f979da` durch *Elena Häußler* ✔

Integration: Filtern

Kurzbeschreibung: Es wird getestet, ob die App das gewünschte Verhalten beim Filtern nach Gerichtstyp, Allergenen, Preis, Bewertung, Favoriten und Häufigkeit vorweist. Dabei wurde die Backend-Datenbank zur Überprüfung genutzt.

Alle betroffenen Komponenten:

- View `Speiseplanansicht`
- View `Filteransicht`
- Komponente `View`
- Komponente `MealData`
- Tool `PostgreSQL-Datenbank + pgAdmin`

Getestete Schnittstellen:

+ **Test:** nach veganen Gerichten filtern

  Vorbereitung:

  1. App starten
  2. Filter zurücksetzen

  Durchführung: Veganen Filter aktivieren

  Ergebnis: Es werden nur noch Vegane oder unspezifizierte Gerichte angezeigt.

+ **Test:** Rindfleisch heraus filtern

  Vorbereitung:

  1. App starten
  2. Filter zurücksetzen

  Durchführung: Rind Filter aktivieren

  Ergebnis: Es werden nur noch Gerichte ohne Rind angezeigt.

+ **Test:** Milch/Laktose heraus filtern

  Vorbereitung:

  1. App starten
  2. Filter zurücksetzen

  Durchführung: Milch/Laktose Filter aktivieren

  Ergebnis: Es werden nur noch Gerichte ohne Milch bzw. Laktose angezeigt.

+ **Test:** nach max. Preis filtern

  Vorbereitung:

  1. App starten
  2. Filter zurücksetzen

  Durchführung: Preis Regler auf ca. drei Euro setzen

  Ergebnis: Es werden nur noch Gerichte angezeigt, die weniger kosten als der eingestellte Preis.

+ **Test:** nach min. Bewertung filtern

  Vorbereitung:

  1. App starten
  2. Filter zurücksetzen

  Durchführung: Bewertungsregler auf ca. drei Sterne setzen

Ergebnis: Es werden nur noch Gerichte angezeigt, mindestens drei Sterne haben, oder noch nicht bewertet wurden.

+ **Test:** nur Favoriten anzeigen

  Vorbereitung:

  1. App starten
  2. einige Gerichte favorisieren
  3. Filter zurücksetzen

  Durchführung: Nur Favoriten aktivieren

  Ergebnis: Es werden nur noch Favoriten angezeigt.

+ **Test:** nur Neue Gerichte anzeigen

  Vorbereitung:

  1. App starten
  2. Filter zurücksetzen

  Durchführung: „Neue Gerichte“-Filter aktivieren

  Ergebnis: Es werden nur noch Gerichte angezeigt, die seit dem Zeitpunkt der Existenz der Backend-Datenbank das erste Mal gab.

Getestet auf `2af9dee7503ed41fb90f6c0572edacb14a48b43b` durch *Elena Häußler* ✔

Integration: Sortieren

Kurzbeschreibung: Es wird getestet, ob die App das gewünschte Verhalten beim Sortieren nach Linie, Preis, Bewertung und Häufigkeit zeigt. Dabei wurde die Backend-Datenbank zur Überprüfung der Reihenfolge genutzt.

Alle betroffenen Komponenten:

- View `Speiseplanansicht`
- View `Filteransicht`
- Komponente `View`
- Komponente `MealData`
- Tool `PostgreSQL-Datenbank + pgAdmin`

Getestete Schnittstellen:

+ **Test:** nach Linie aufsteigend sortieren

  Vorbereitung:

  1. App starten

  Durchführung: Filtereinstellungen zurücksetzen

  Ergebnis: Gerichte sind aufsteigend nach Linie sortiert, wobei hierbei die Positionen der Linien genutzt werden

+ **Test:** nach Linie absteigend sortieren

  Vorbereitung:

  1. App starten
  2. Filter zurücksetzen

  Durchführung: Sortierreihenfolge ändern

Ergebnis: Gerichte sind absteigend nach Linie sortiert, wobei hierbei die Positionen der Linien genutzt werden

+ **Test:** nach Bewertung aufsteigend sortieren

  Vorbereitung:

  1. App starten
  2. Filter zurücksetzen

  Durchführung: Sortierung nach Bewertung auswählen

  Ergebnis: Gerichte sind aufsteigend nach Bewertung sortiert

+ **Test:** nach Bewertung absteigend sortieren

  Vorbereitung:

  1. App starten
  2. Filter zurücksetzen

  Durchführung: Sortierung nach Bewertung auswählen und Sortierreihenfolge ändern

  Ergebnis: Gerichte sind absteigend nach Bewertung sortiert.

+ **Test:** nach Preis aufsteigend sortieren

  Vorbereitung:

  1. App starten
  2. Filter zurücksetzen

  Durchführung: Sortierung nach Preis auswählen

  Ergebnis: Gerichte sind aufsteigend nach Preis sortiert.

+ **Test:** nach Preis absteigend sortieren

  Vorbereitung:

  1. App starten
  2. Filter zurücksetzen

  Durchführung: Sortierung nach Preis auswählen und Sortierreihenfolge ändern

  Ergebnis: Gerichte sind absteigend nach Preis sortiert.

+ **Test:** nach Häufigkeit aufsteigend sortieren

  Vorbereitung:

  1. App starten
  2. Filter zurücksetzen

  Durchführung: Sortierung nach Häufigkeit auswählen

  Ergebnis: Gerichte sind aufsteigend nach Häufigkeit sortiert.

+ **Test:** nach Häufigkeit absteigend sortieren

  Vorbereitung:

  1. App starten
  2. Filter zurücksetzen

  Durchführung: Sortierung nach Häufigkeit auswählen und Sortierreihenfolge ändern

  Ergebnis: Gerichte sind absteigend nach Häufigkeit sortiert.

Getestet auf `6ca40878b292807571f7f84c0fe1c95261e21f30` durch *Elena Häußler* ✔

5 Abnahmetest

Ziel des Abnahmetests ist es sicherzustellen, dass alle Funktionen so umgesetzt wurden wie im Pflichtenheft definiert. Dazu wurden teilweise schon im Pflichtenheft Tests (Funktionstests und Sequenztests) definiert, um dies sicherzustellen. Diese decken jedoch lediglich die Musskriterien ab. Daher wurden für alle umgesetzten Kannkriterien weitere Tests anhand der Funktionsdefinitionen im Pflichtenheft entworfen.

5.1 Funktionstests

[TF01X] App starten

Betroffene Funktionen: `[F101]`
Ausführung: Die App wird gestartet
Varianten:

- **`[TF011]`** **Vorbedingung:** Die App wird gestartet
  Erwartung: Die App zeigt die Speiseplanansicht und der Speiseplan wird aktualisiert.

Erfolgreich durchgeführt von *Alexander Albers* ✔

[TF02X] UPDATED Server starten

Betroffene Funktionen: `[F801]`
Ausführung: Der Administrator startet den Server
Varianten:

- **`[TF021]`** **Vorbedingung:** Umgebungsvariablen wurden nicht definiert
- **`[TF022]`** **Vorbedingung:** Die Datenbank-Initialisierung schlägt fehl
- **`[TF023]`** **Vorbedingung:** Der `GraphQL-Server` konnte nicht gestartet werden
  Erwartung: Der Server konnte nicht gestartet werden und die jeweiligen Fehler werden protokolliert
- **`[TF024]`** **Vorbedingung:** Der Server startet ohne Probleme (Keiner der obigen Fälle trifft zu)
  Erwartung: Der Server protokolliert einen erfolgreichen Start

Erfolgreich durchgeführt von *Alexander Albers* ✔

[TF03X] Speiseplanansicht anzeigen

Betroffene Funktionen: `[F102]`
Ausführung: Der Nutzer befindet sich in der Favoriten-, oder Einstellungsansicht
Varianten:

- **`[TF031]`** **Vorbedingung:** Die App zeigt die Einstellungsansicht an
- **`[TF032]`** **Vorbedingung:** Die App zeigt die Favoritenansicht an
  Erwartung: Die Speiseplanansicht wird angezeigt

Erfolgreich durchgeführt von *Alexander Albers* ✔

[TF04X] Speiseplan aktualisieren

Betroffene Funktionen: `[F103], [F802], [F804]`
Ausführung: Der Nutzer wischt den Speiseplan nach unten
Varianten:

- **[TF041]** **Vorbedingung:** Die App hat eine Verbindung zum Server
  Erwartung: Die App zeigt die Speiseplanansicht an. Es werden alle Gerichte des ausgewählten Tages angezeigt
- **[TF042]** **Vorbedingung:** Die App hat keine Serververbindung und die App hat lokale Daten für den Tag gespeichert
  Erwartung: Die App zeigt die Speiseplanansicht an. Es werden alle Gerichte des ausgewählten Tages angezeigt
- **[TF043]** **Vorbedingung:** Die App hat keine Serververbindung und die App hat keine lokalen Daten für den Tag gespeichert
  Erwartung: Es wird eine Fehlermeldung angezeigt

Erfolgreich durchgeführt von *Alexander Albers* ✔

[TF05X] Detailansicht anzeigen

Betroffene Funktionen: `[F201]`
Ausführung: Der Nutzer tippt auf ein Gericht
Varianten:

- **[TF051]** **Vorbedingung:** Die App zeigt die Speiseplanansicht an
- **[TF052]** **Vorbedingung:** Die App zeigt die Favoritenansicht an
  Erwartung: Die Detailansicht wird angezeigt

Erfolgreich durchgeführt von *Alexander Albers* ✔

[TF06X] Einstellungen anzeigen

Betroffene Funktionen: `[F601]`
Ausführung: Der Nutzer tippt auf das Einstellungssymbol in der Navigationsleiste
Varianten:

- **[TF061]** **Vorbedingung:** Die App zeigt die Speiseplanansicht an
- **[TF062]** **Vorbedingung:** Die App zeigt die Favoritenansicht an
  Erwartung: Die Einstellungsansicht wird angezeigt

Erfolgreich durchgeführt von *Alexander Albers* ✔

[TF07X] Preisklasse einstellen

Betroffene Funktionen: `[F603]`
Ausführung: Der Nutzer wählt eine Preisklasse
Varianten:

- **[TF071]** **Vorbedingung:** Die App zeigt die Einstellungsansicht an
  Erwartung: Die Preise der Gerichte entsprechen der ausgewählten Preisklasse

Erfolgreich durchgeführt von *Jonatan Ziegler* ✔

[TF08X] Mensa wechseln

Betroffene Funktionen: `[F104]`

Ausführung:

- Der Nutzer tippt auf die aktuell ausgewählte Mensa
- Der Nutzer wählt eine andere Mensa aus

Varianten:

`[TF081]` **Vorbedingung:** Die App zeigt die Speiseplananansicht an
Erwartung: Der Speiseplan der anderen Mensa wird angezeigt

Erfolgreich durchgeführt von *Jonatan Ziegler* ✔

[TF09X] Tag ändern

Betroffene Funktionen: `[F105]`, `[F803]`

Ausführung:

- Der Nutzer tippt auf das Datum
- Der Nutzer wählt einen Tag aus

Varianten:

`[TF091]` **Vorbedingung:** Die App zeigt die Speiseplananansicht an
Erwartung: Der Speiseplan für den neu ausgewählten Tag wird angezeigt

Erfolgreich durchgeführt von *Jonatan Ziegler* ✔

[TF10X] Filterdialog öffnen

Betroffene Funktionen: `[F401]`, `[F402]`

Ausführung: Der Nutzer tippt auf das Filtersymbol

Varianten:

`[TF101]` **Vorbedingung:** Die App zeigt die Speiseplananansicht an und es wurde noch kein Filter konfiguriert
Erwartung: Der Filterdialog wird angezeigt und alle Standardwerte sind ausgewählt

`[TF102]` **Vorbedingung:** Die App zeigt die Speiseplananansicht an und es wurde ein Filter konfiguriert
Erwartung: Der Filterdialog wird angezeigt mit den zuletzt eingestellten Filtermöglichkeiten

Erfolgreich durchgeführt von *Jonatan Ziegler* ✔

[TF11X] Filterdialog schließen

Betroffene Funktionen: `[F401]`, `[F402]`

Ausführung: Der Nutzer tippt auf die „Schließen“-Schaltfläche

Varianten:

`[TF111]` **Vorbedingung:** Der Filterdialog ist geöffnet
Erwartung: Der Filterdialog wird geschlossen und die App zeigt die Speiseplananansicht

Erfolgreich durchgeführt von *Jonatan Ziegler* ✔

[TF12X] UPDATED Filteroptionen auswählen

Betroffene Funktionen: `[F403]`
Ausführung: Der Nutzer wählt Filteroptionen aus
Varianten:

- **[TF121]** **Vorbedingung:** Es wird nach vegetarischen Gerichten gefiltert
  Erwartung: Alle Gerichte, die angezeigt werden, sind vegetarisch
- **[TF122]** **Vorbedingung:** Gerichte mit Schweinefleisch werden herausgefiltert.
  Erwartung: Alle Gerichte, die angezeigt werden, enthalten kein Schweinefleisch
- **[TF123]** **Vorbedingung:** Es wird nach Gerichten gefiltert, die keine Cashewnüsse enthalten
  Erwartung: Alle Gerichte, die angezeigt werden, enthalten keine Cashewnüsse
- **[TF124]** **Vorbedingung:** Es wird nach Gerichten gefiltert, die unter 5€ kosten
  Erwartung: Alle Gerichte, die angezeigt werden, kosten weniger als 5€
- **[TF125]** **Vorbedingung:** Es wird nach Gerichten gefiltert, die eine Bewertung von mindestens vier Sternen haben
  Erwartung: Alle Gerichte, die angezeigt werden, haben eine Bewertung von mindestens vier Sternen
- **[TF126]** **Vorbedingung:** Es wird nach favorisierten Gerichten gefiltert
  Erwartung: Alle Gerichte, die angezeigt werden, sind favorisiert
- **[TF127]** **Vorbedingung:** Es wird nach neuen Gerichten gefiltert
  Erwartung: Alle Gerichte, die angezeigt werden, gab es noch nicht

Erfolgreich durchgeführt von *Jonatan Ziegler & Alexander Albers* ✔

[TF13X] Filter speichern

Betroffene Funktionen: `[F404]`
Ausführung: Der Nutzer tippt auf die „Speichern“-Schaltfläche
Varianten:

- **[TF131]** **Vorbedingung:** Keine Filtermöglichkeit hat sich geändert
- **[TF132]** **Vorbedingung:** Eine Filtermöglichkeit hat sich geändert
  Erwartung: Die Filterkonfiguration wird lokal gespeichert und der Filterdialog schließt sich

Erfolgreich durchgeführt von *Peer Booken* ✔

[TF14X] UPDATED Filter zurücksetzten

Betroffene Funktionen: `[F405]`
Ausführung: Der Nutzer tippt auf die „Zurücksetzen“-Schaltfläche
Varianten:

- **[TF141]** **Vorbedingung:** Filter wird zurückgesetzt
  Erwartung: Alle Filtermöglichkeiten werden auf ihre Standardwerte zurückgesetzt. Es werden alle Gerichte für den Tag angezeigt

Erfolgreich durchgeführt von *Peer Booken* ✔

[TF15X] UPDATED Filter de-/aktivieren

Betroffene Funktionen: `[F406]`, `[F407]`

Ausführung: Der Nutzer hält das Filtersymbol gedrückt

Varianten:

`[TF151]` **Vorbedingung:** Der Filter ist aktiviert
Erwartung: Der Filter ist deaktiviert, alle Gerichte des ausgewählten Tages der jeweiligen Mensa werden angezeigt und das Filtersymbol ändert sich

`[TF152]` **Vorbedingung:** Der Filter ist deaktiviert
Erwartung: Der Filter ist aktiviert, alle Gerichte die dem Filter entsprechen werden angezeigt und das Filtersymbol ändert sich

Erfolgreich durchgeführt von *Peer Booken* ✔

[TF16X] UPDATED Galeriedialog ein-/ausblenden

Betroffene Funktionen: `[F301]`, `[F302]`, `[F806]`

Ausführung:

- Galeriedialog ist ausgeblendet: Der Nutzer tippt auf das Bild in der Detailansicht
- Galeriedialog ist eingeblendet: Der Nutzer tippt auf die „Schließen“-Schaltfläche

Varianten:

`[TF161]` **Vorbedingung:** Der Galeriedialog ist eingeblendet
Erwartung: Die App zeigt die Detailansicht des jeweiligen Gerichts

`[TF162]` **Vorbedingung:** Der Galeriedialog ist ausgeblendet und zu dem ausgewählten Gericht wurde ein Bild hochgeladen
Erwartung: Der Galeriedialog öffnet sich und zeigt das Bild an

`[TF163]` **Vorbedingung:** Der Galeriedialog ist ausgeblendet und zu dem ausgewählten Gericht wurde noch kein Bild hochgeladen
Erwartung: Der Galeriedialog lässt sich nicht öffnen

Erfolgreich durchgeführt von *Peer Booken* ✔

[TF17X] Bild melden

Betroffene Funktionen: `[F303]`, `[F807]`

Ausführung:

- Der Nutzer tippt auf die „Melden“-Schaltfläche
- Der Nutzer gibt einen Grund zum Melden an
- Der Nutzer bestätigt das Melden

Varianten:

`[TF171]` **Vorbedingung:** Der Nutzer meldet Bild wie oben beschrieben mit einer Verbindung zum Server
Erwartung: Das Bild wird dem Nutzer nicht mehr angezeigt und eine temporäre Erfolgsmeldung wird angezeigt

`[TF172]` **Vorbedingung:** Der Nutzer meldet Bild wie oben beschrieben ohne eine Verbindung zum Server
Erwartung: Eine temporäre Fehlermeldung wird angezeigt

`[TF173]` **Vorbedingung:** Der Nutzer bricht das Melden ab
Erwartung: Es passiert nichts

Erfolgreich durchgeführt von *Alexander Albers & Peer Booken* ✔

[TF18X] Bild hochladen

Betroffene Funktionen: `[F205]`, `[F805]`

Ausführung:

- Der Nutzer navigiert zur Detailansicht
- Der Nutzer tippt auf die „Hochladen“-Schaltfläche.
- Der Nutzer fügt einen Link zu dem Bild ein.
- Der Nutzer bestätigt das Einbinden des Bildes.

Varianten:

`[TF181]` **Vorbedingung:** Die App hat eine Verbindung zum Server und der eingefügte Link ist valide
Erwartung: Das Bild wird eingebunden und angezeigt

`[TF182]` **Vorbedingung:** Die App hat eine Verbindung zum Server und der eingefügte Link ist nicht valide

`[TF183]` **Vorbedingung:** Die App hat keine Verbindung zum Server
Erwartung: Eine temporäre Fehlermeldung wird angezeigt

Erfolgreich durchgeführt von *Peer Booken* ✔

[TF19X] ⭐ NEW Zwischen Bildern durchwechseln

Betroffene Funktionen: `[F304]`, `[F305]`

Ausführung:

- Der Nutzer navigiert zum Galeriedialog
- Der Nutzer wischt nach links/rechts.

Varianten:

`[TF191]` **Vorbedingung:** rechts
Erwartung: Falls vorhanden, wird das Bild mit nächst höherem Bildrang zum aktuellen Gericht angezeigt.

`[TF192]` **Vorbedingung:** links
Erwartung: Falls vorhanden wird das Bild mit den nächstniedrigeren Bildrang zum aktuellen Gericht angezeigt

Erfolgreich durchgeführt von *Jonatan Ziegler* ✔

[TF20X] ⭐ NEW Bilder bewerten

Betroffene Funktionen: `[F306]`, `[F307]`, `[F308]`, `[F309]`

Ausführung:

- Der Nutzer navigiert zum Galeriedialog
- Der Nutzer betätigt die Schaltfläche zum Auf-/Abwerten.

Varianten:

`[TF201]` **Vorbedingung:** Auf-/Abwerten: Es besteht eine Internetverbindung und das Bild wurde vom Nutzer noch nicht bewertet
Erwartung: Die Bewertungsschaltfläche stellt dar, dass das Bild entsprechend bewertet wurde.

`[TF202]` **Vorbedingung:** Aufwerten: Es besteht eine Internetverbindung und das Bild wurde vom Nutzer zuletzt abgewertet
Erwartung: Die Bewertungsschaltfläche stellt dar, dass das Bild als hilfreich bewertet wurde.

`[TF203]` **Vorbedingung:** Abwerten: Es besteht eine Internetverbindung und das Bild wurde vom Nutzer zuletzt aufgewertet
Erwartung: Die Bewertungsschaltfläche stellt dar, dass das Bild als nicht hilfreich bewertet wurde.

`[TF204]` **Vorbedingung:** Auf-/Abwerten: Es besteht eine Internetverbindung und das Bild wurde vom Nutzer schon auf-/abgewertet.
Erwartung: Die Bewertungsschaltfläche stellt dar, dass das Bild als keiner Bewertung mehr obliegt.

`[TF205]` **Vorbedingung:** Es besteht keine Internetverbindung
Erwartung: Es wird eine temporäre Fehlermeldung angezeigt und die Auf-/Abwertung verändert sich nicht.

Erfolgreich durchgeführt von *Jonatan Ziegler* ✔

[TF21X] ★ NEW Sortieren von Gerichten

Betroffene Funktionen: `[F403]`

Ausführung:

- Der Nutzer navigiert zum Filterdialog
- Der Nutzer wählt eine Sortierung.

Varianten:

`[TF211]` **Vorbedingung:** Es wird nach Linie absteigend sortiert.
Erwartung: Die Gerichte werden in der Speiseplanansicht nach ihrer zugehörigen Linie sortiert angezeigt. Dabei ist die übliche Reihenfolge umgekehrt.

`[TF212]` **Vorbedingung:** Es wird nach Preis aufsteigend sortiert.
Erwartung: Die Gerichte werden in der Speiseplanansicht nach Preis sortiert angezeigt. Dabei werden Gerichte mit dem kleinsten Preis zuerst angezeigt.

`[TF213]` **Vorbedingung:** Es wird nach Bewertung absteigend sortiert.
Erwartung: Die Gerichte werden in der Speiseplanansicht nach der Bewertung sortiert angezeigt. Dabei werden Gerichte mit der größten Bewertung zuerst angezeigt.

`[TF214]` **Vorbedingung:** Es wird nach Häufigkeit aufsteigend sortiert.
Erwartung: Die Gerichte werden in der Speiseplanansicht nach der Häufigkeit sortiert angezeigt. Dabei werden neue Gerichte zuerst angezeigt.

Erfolgreich durchgeführt von *Jonatan Ziegler* ✔

[TF22X] ★ NEW Nachricht bei zu strengem Filter

Betroffene Funktionen: `[F406]`

Ausführung:

- Der Nutzer navigiert zum Filterdialog
- Nutzer wählt Filter übernimmt diesen

Varianten:

`[TF221]` **Vorbedingung:** kein Gericht erfüllt die Filtereinstellungen
Erwartung: Es wird eine Nachricht angezeigt, dass kein Gericht den Filtereinstellungen entspricht

Erfolgreich durchgeführt von *Jonatan Ziegler* ✔

[TF23X] ★ NEW Favoriten anzeigen

Betroffene Funktionen: `[F501]`

Ausführung: Nutzer navigiert zur Favoritenansicht

Varianten:

`[TF231]` **Vorbedingung:** Internetverbindung vorhanden
Erwartung: Es werden alle vom Nutzer favorisierten Gerichte angezeigt

`[TF232]` **Vorbedingung:** keine Internetverbindung vorhanden
Erwartung: Es wird eine temporäre Fehlermeldung angezeigt.

Erfolgreich durchgeführt von *Jonatan Ziegler & Alexander Kutschera* ✔

[TF24X] ★ NEW Gericht favorisieren und defavorisieren

Betroffene Funktionen: `[F502], [F503]`

Ausführung:

- Nutzer befindet sich in Detailansicht
- Nutzer betätigt Schaltfläche zum Favorisieren

Varianten:

`[TF241]` **Vorbedingung:** Gericht ist noch nicht favorisiert
Erwartung: Gericht ist nun in der Favoritenansicht sichtbar

`[TF242]` **Vorbedingung:** Gericht ist schon favorisiert
Erwartung: Gericht ist nun nicht mehr in der Favoritenansicht sichtbar

Erfolgreich durchgeführt von *Jonatan Ziegler* ✔

[TF25X] ★ NEW Ereignisse des Servers werden protokolliert

Betroffene Funktionen: `[F808]`

Ausführung: Der Server läuft oder wird gestartet

Varianten:

`[TF251]` **Vorbedingung:** Starten des Servers und erfolgreicher Start

`[TF252]` **Vorbedingung:** Fehler beim Starten des Servers

`[TF253]` **Vorbedingung:** Starten und beenden einer Speiseplanaktualisierung

`[TF254]` **Vorbedingung:** Auflösen und einfügen von Speiseplänen

`[TF255]` **Vorbedingung:** Senden einer Web-Anfrage an die Mensa-Webseite

`[TF256]` **Vorbedingung:** Eingehende API-Anfragen und deren Fehler

`[TF257]` **Vorbedingung:** Löschen von beim Bildhoster nicht mehr existierenden Bildern

`[TF258]` **Vorbedingung:** Starten und Beenden des Vorgangs zum automatischen Ausblenden von Bildern

`[TF259]` **Vorbedingung:** Senden einer Anfrage an die externe Flickr-API

`[TF2510]` **Vorbedingung:** Laden von Konfigurationen aus den Umgebungsvariablen
Erwartung: Das Ereignis wird protokolliert

Erfolgreich durchgeführt von *Alexander Albers* ✔

5.2 Sequenztests

In dieser Sektion werden alle Sequenztests aus dem Pflichtenheft aufgelistet. Diese wurden manuell getestet.

Sequenz: „Gewöhnliche Abfolge bei Gerichtsauswahl“

Betroffenen Funktionen: `[F101]`, `[F102]`, `[F301]`, `[F302]`
Alle betroffenen Komponenten:

- View `Speiseplanansicht`
- View `Detailansicht`

Ablauf:

1. Die App wird gestartet `[TF011]`
2. *Der Nutzer kann den Speiseplan einsehen*
3. Ein Gericht auswählen
4. *Der Nutzer kann die Informationen zu einem Gericht anschauen*
5. Zurück zur Speiseplanansicht navigieren
6. *Die letzten drei Schritte beliebig oft wiederholen*
7. *Die App zeigt die Speiseplanansicht an*

Erfolgreich durchgeführt von *Alexander Albers* ✔

Sequenz: „Filter auf Speiseplanansicht anwenden“

Betroffenen Funktionen: `[F101]`, `[F102]`, `[F401]`, `[F403]`, `[F404]`
Alle betroffenen Komponenten:

- View `Speiseplanansicht`
- View `Filterdialog`

Ablauf:

1. Die App wird gestartet `[TF011]`
2. *Der Nutzer kann den Speiseplan einsehen*
3. Filterdialog öffnen `[TF101]`
4. Wählen von beliebiger Filterkombination `[TF121]`
5. Übernehmen der Filterkonfiguration `[TF132]`
6. *Der Nutzer kann den gefilterten Speiseplan einsehen*
7. *Es werden nur Gerichte angezeigt, die dem Filter entsprechen*

Erfolgreich durchgeführt von *Alexander Albers* ✔

Sequenz: „Einstellen des Filters wird abgebrochen“

Betroffenen Funktionen: `[F101]`, `[F102]`, `[F401]`, `[F402]`, `[F403]`
Alle betroffenen Komponenten:

- View `Speiseplanansicht`
- View `Filterdialog`

Ablauf:

1. Die App wird gestartet `[TF011]`
2. *Der Nutzer kann den Speiseplan einsehen*
3. Filterdialog öffnen `[TF101]`
4. Auswahl einer beliebigen Filterkombination `[TF121]`
5. Verlassen des Filterdialogs
6. *Keine Änderung in der Speiseplananansicht kann festgestellt werden*

Erfolgreich durchgeführt von *Alexander Albers* ✔

Sequenz: „★ NEW Tag im Speiseplan wechseln“

Betroffenen Funktionen: `[F101]`, `[F102]`, `[F108]`, `[F109]`
Alle betroffenen Komponenten:

- View `Speiseplanansicht`
- View `Kalenderdialog`

Ablauf:

1. Die App wird gestartet `[TF011]`
2. *Der Nutzer kann den Speiseplan einsehen*
3. Wechsel ein Tag in die Vergangenheit über die Pfeil-Schaltfläche
4. *Der Nutzer kann den Speiseplan von gestern einsehen*
5. Wechsel zwei Tage in die Zukunft über die Pfeil-Schaltfläche
6. *Der Nutzer kann den Speiseplan von morgen einsehen*
7. Öffnen des Kalenderdialogs durch tippen auf das Datum
8. Wahl eines Tages, der über einem Monat in der Zukunft liegt
9. *Dem Nutzer wird angezeigt, dass keine Daten verfügbar sind*

Erfolgreich durchgeführt von *Alexander Albers* ✔

Sequenz: „⭐ NEW Wechseln der Ansicht“

Betroffenen Funktionen: [F101], [F102], [F106], [F107]
Alle betroffenen Komponenten:

- View `Speiseplanansicht`

Ablauf:

1. Die App wird gestartet [TF011]
2. *Der Nutzer kann den Speiseplan einsehen*
3. Wechsel in die Listenansicht
4. *Der Nutzer kann denselben Speiseplan einsehen*
5. Wechsel in die Galerieansicht
6. *Der Nutzer kann denselben Speiseplan einsehen*

Erfolgreich durchgeführt von *Alexander Albers* ✔

Sequenz: „⭐ NEW Allergene und Zusatzstoffe einsehen“

Betroffenen Funktionen: [F101], [F102], [F201], [F206], [F207]
Alle betroffenen Komponenten:

- View `Speiseplanansicht`
- View `Detailansicht`

Ablauf:

1. Die App wird gestartet [TF011]
2. *Der Nutzer kann den Speiseplan einsehen*
3. Auswahl eines Gerichts [TF051]
4. *Der Nutzer befindet sich in der Detailansicht des ausgewählten Gerichtes*
5. *Alle Zusatzstoffe werden angezeigt*
6. Allergene einblenden
7. *Allergene werden dem Nutzer angezeigt*
8. Allergene ausblenden
9. *Allergene werden dem Nutzer nicht mehr angezeigt*

Erfolgreich durchgeführt von *Alexander Albers* ✔

Sequenz: „⭐ NEW Farbschema wechseln“

Betroffenen Funktionen: [F101], [F102], [F207], [F501], [F601], [F602], [F207]
Alle betroffenen Komponenten:

- View Speiseplanansicht
- View Favoritenansicht
- View Einstellungsansicht

Ablauf:

1. Die App wird gestartet [TF011]
2. *Der Nutzer kann den Speiseplan einsehen*
3. Wechsel in die Einstellungsansicht [TF061]
4. *Der Nutzer wechselt das Farbschema*
5. Wechsel in die Speiseplanansicht [TF031]
6. *Das Farbschema wurde angewendet*
7. Wechsel in die Favoritenansicht [TF231]
8. *Das Farbschema wurde angewendet*

Erfolgreich durchgeführt von *Alexander Albers & Peer Booken* ✔

Sequenz: „Filter deaktivieren und aktivieren“

Betroffenen Funktionen: [F101], [F102], [F406], [F407]
Alle betroffenen Komponenten:

- View Speiseplanansicht

Ablauf:

1. Die App wird gestartet [TF011]
2. *Der Nutzer kann den Speiseplan einsehen*
3. *Filter auf Speiseplananansicht anwenden [TS02]*
4. Filter deaktivieren [TF151]
5. *Alle Gerichte des ausgewählten Tages der jeweiligen Mensa werden angezeigt*
6. Filter aktivieren [TF152]
7. *Es werden nur Gerichte angezeigt, die dem Filter entsprechen*

Erfolgreich durchgeführt von *Jonatan Ziegler* ✔

Sequenz: „Lokale Daten, Preisklasse und Filter bleiben gespeichert“

Betroffenen Funktionen: `[F101]`, `[F102]`, `[F601]`, `[F603]`
Alle betroffenen Komponenten:

- View `Speiseplanansicht`
- View `Einstellungsansicht`

Ablauf:

1. Die App wird gestartet `[TF011]`
2. *Der Nutzer kann den Speiseplan einsehen*
3. *Filter auf Speiseplanansicht anwenden `[TS02]`*
4. Wechseln in die Einstellungsansicht `[TF061]`
5. Ändern der Preisklasse `[TF071]`
6. App schließen
7. App starten `[TF011]`
8. *Die Lokalen Daten, die Preisklasse und die Filterkonfiguration sind dieselben wie vor dem Schließen der App*

Erfolgreich durchgeführt von *Jonatan Ziegler* ✔

Sequenz: „Der Nutzer meldet ein Bild“

Betroffenen Funktionen: `[F101]`, `[F102]`, `[F201]`, `[F301]`, `[F303]`
Alle betroffenen Komponenten:

- View `Speiseplanansicht`
- View `Detailansicht`
- View `Galeriedialog`
- View `Meldedialog`

Ablauf:

1. Die App wird gestartet `[TF011]`
2. *Der Nutzer kann den Speiseplan einsehen*
3. Ein Gericht auswählen `[TF051]`
4. *Es wird die Detailansicht mit eingebundenem Bild angezeigt*
5. Bild auswählen `[TF162]`
6. *Der Galeriedialog öffnet sich*
7. „Melde“-Schaltfläche betätigen `[TF171]`
8. *Der Meldedialog öffnet sich*
9. Kategorie auswählen
10. Meldedialog bestätigen
11. *Das Bild wird ausgeblendet*
12. *Der Administrator bekommt eine Mail zugesendet*

Erfolgreich durchgeführt von *Jonatan Ziegler* ✔

Sequenz: „Der Nutzer verlinkt ein Bild zu einem Gericht“

Betroffenen Funktionen: `[F101]`, `[F102]`, `[F201]`, `[F205]`
Alle betroffenen Komponenten:

- View `Speiseplanansicht`
- View `Detailansicht`
- View `Hochladedialog`

Ablauf:

1. Die App wird gestartet `[TF011]`
2. *Der Nutzer kann den Speiseplan einsehen*
3. Ein Gericht auswählen `[TF051]`
4. Auf das „Hochladen“-Symbol tippen `[TF181]`
5. *Ein Dialog öffnet sich*
6. Link zum Bild eines Bildhosters einfügen
7. Dialog wird bestätigt
8. *Server validiert die Eingabe*

Erfolgreich durchgeführt von *Peer Booken* ✔

Sequenz: „Mensa hat zum ausgewählten Tag keine Gerichte“

Betroffenen Funktionen: `[F101]`, `[F102]`, `[F105]`
Alle betroffenen Komponenten:

- View `Speiseplanansicht`
- View `Kalenderdialog`

Ablauf:

1. Die App wird gestartet `[TF011]`
2. *Der Nutzer kann den Speiseplan einsehen*
3. Tag ändern auf einen Tag, an dem die ausgewählte Mensa geschlossen ist `[TF091]`
4. *Es wird angezeigt, dass die Mensa geschlossen ist*

Erfolgreich durchgeführt von *Peer Booken* ✔

Sequenz: „Speiseplan aktualisieren ohne Serververbindung“

Betroffenen Funktionen: `[F101]`, `[F102]`, `[F103]`, `[F201]`
Alle betroffenen Komponenten:

- View `Speiseplanansicht`
- View `Detailansicht`

Ablauf:

1. Die App wird gestartet `[TF011]`
2. *Der Nutzer kann den Speiseplan einsehen*
3. Internetzugang auf dem Gerät deaktivieren
4. Speiseplan aktualisieren `[TF041]`
5. *Speiseplan wird mit lokalen Daten geladen*
6. *Der Nutzer kann den Speiseplan einsehen*
7. Ein Gericht auswählen `[TF051]`
8. *Die Detailansicht öffnet sich mit lokalen Daten*

Erfolgreich durchgeführt von *Peer Booken* ✔

Sequenz: „Filter schließt alle Gerichte aus“

Betroffenen Funktionen: `[F101]`, `[F102]`, `[F401]`, `[F403]`, `[F404]`
Alle betroffenen Komponenten:

- View `Speiseplanansicht`
- View `Filterdialog`
- View `Keine Filterergebnisse`

Ablauf:

1. Die App wird gestartet `[TF011]`
2. *Der Nutzer kann den Speiseplan einsehen*
3. Filterdialog öffnen `[TF101]`
4. Filter anpassen, sodass kein Gericht die Filterkriterien erfüllt `[TF126]`
5. Filter speichern `[TF132]`
6. *Es werden keine Gerichte angezeigt, da alle durch den Filter ausgeschlossen sind*

Erfolgreich durchgeführt von *Peer Booken* ✔

5.3 Nichtfunktionale Abnahme

Leistung

> Bis zu 1000 Nutzer müssen im ordnungsgemäßen Betrieb (vgl. Produktumgebung) gleichzeitig durch die App navigieren können und infolgedessen Anfragen an den Server senden, welche beantwortet werden.

Die wesentlichen Anfragen beim Navigieren durch die App passieren beim Wählen eines Tages auserhalb der nächsten sieben oder beim Aktualisieren durch nach unten wischen. Hierbei kann der Speiseplan für den gewählten Tag und die gewählte Mensa vom Server abgefragt.

Abbildung 1 zeigt die Antwortzeit einer solchen Anfrage abhängig von den nebenher gleichzeitig gestellten Anfragen. Dabei wurde der Server nach den Minimalbedingungen des ordnungsgemäßen Betriebs ausgeführt. Die Antwortzeit reicht von ca. 10 bis 20 ms bei Einzelanfragen. Bei 1000 gleichzeitigen Anfragen sind es 4,5 bis 9 Sekunden.

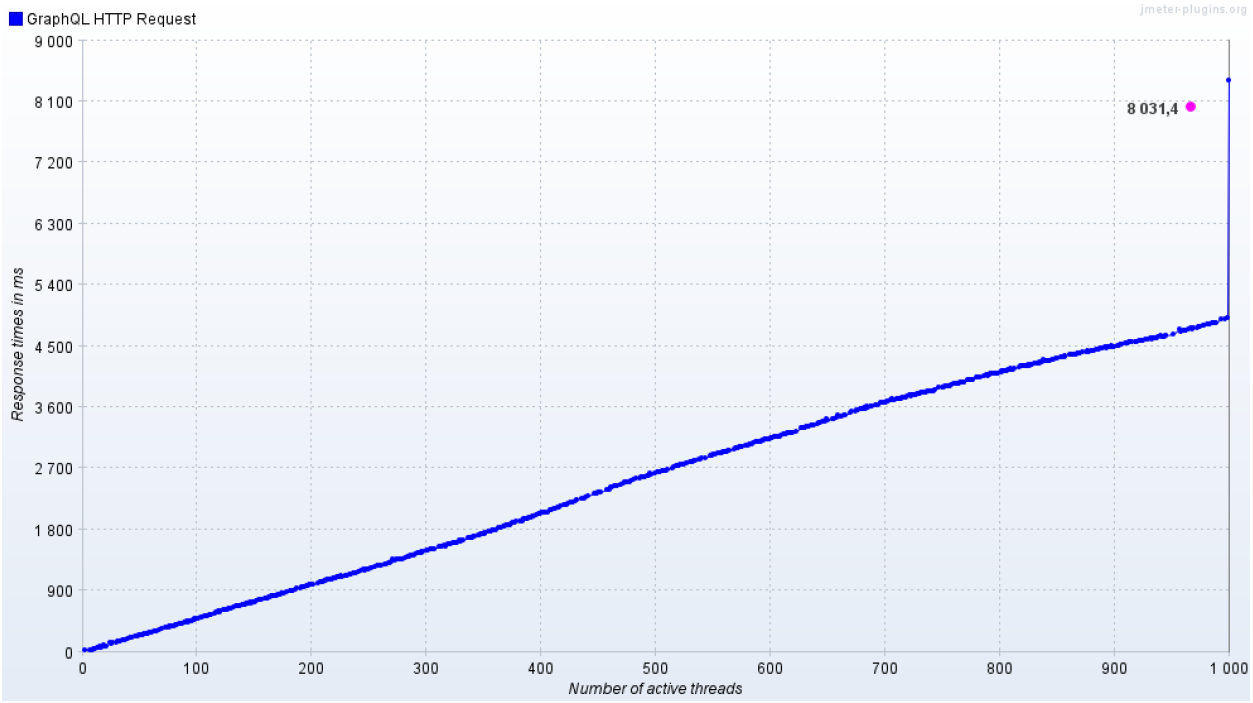

Abbildung 1: Antwortzeit pro gleichzeitige Nutzer

Da jedoch bei normaler Nutzung und Navigation der App eine Anfrage unter den oben genannten Bedingungen eher selten Eintritt, ist hier nur mit ca. 100 gleichzeitigen Anfragen bei 1000 Nutzern zu rechnen. Dabei liegt die Antwortzeit unter einer Sekunde (siehe Abbildung 2) und damit in einem akzeptablen Rahmen. Darüber hinaus wurden alle Anfragen, auch bei hoher Last erfolgreich beantwortet und es kam zu keinen Verbindungsabbrüchen durch Überlastung.

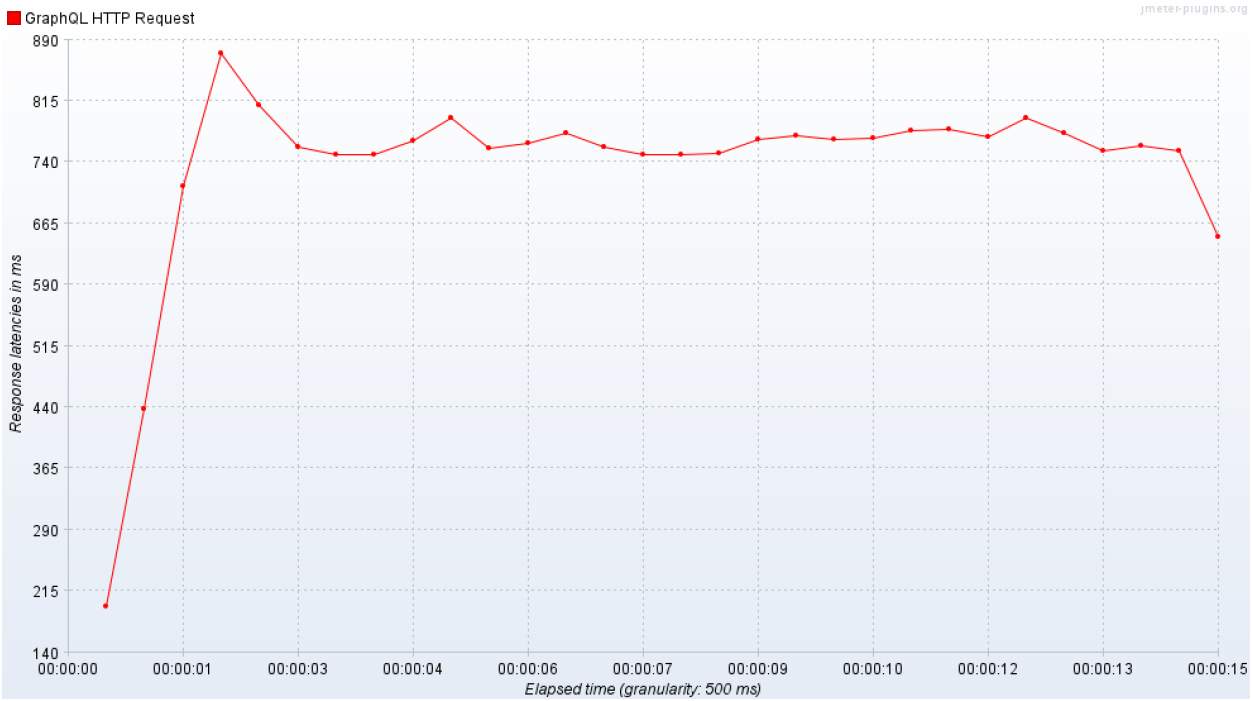

Abbildung 2: Antwortzeit bei 100 gleichzeitigen Anfragen

✔ Somit kann die Anforderung von beantworteten Anfragen bei 1000 gleichzeitigen Nutzern als erfüllt betrachtet werden.

Sicherheit

API-Anfragen an den Server, welche Änderungen in der Datenbank auslösen, müssen durch einen API-Key abgesichert sein.

✔ Erfüllt, siehe `https://github.com/kronos-et-al/MensaApp/blob/main/doc/ApiAuth.md`

Benutzbarkeit

Die Nutzerführung der Anwendung muss intuitiv gestaltet sein. Dadurch soll eine Navigation durch die App ohne externe Anweisungen möglich sein.

Die App wurde Testpersonen aus verschiedenen Altersgruppen zur Verfügung gestellt. Unter Beobachtung der Testpersonen wurde folgendes festgestellt:

- Alle Tester haben initial sich intuitiv durch den Speiseplan bewegt, Gerichte geöffnet und wieder geschlossen. Das Wechseln der Ansicht über die Navigationsleiste passierte intuitiv.
- Die meisten Teilnehmenden haben die Bewerten-Schaltfläche selbstständig gefunden.
- Der Dialog zum Bewerten eines Gerichts konnte ohne Fragen ausgefüllt werden.
- Nachdem die Information verbreitet wurde, dass mehrere Bilder zu Gerichten existieren, haben die Teilnehmenden die Galerieansicht selbstständig gefunden.
- Das Auf- und Abwerten von Bildern wurde intuitiv ausgeführt, genauso wie das Melden eines Bildes oder das Wechseln zwischen Bildern in der Galerieansicht.
- Zuletzt wurden die Testpersonen aufgefordert, selbst ein Bild zu einem Gericht hochzuladen. Nahezu alle Tester fanden die Hochladen-Schaltfläche selbstständig.
- Das Hochladen eines Bildes
  - konnte von älteren Personen oder Kindern nicht selbstständig ausgeführt werden. Außerdem besaßen die betroffenen Personen keinen Flickr-Account.
  - konnte von jüngeren Personen selbstständig ausgeführt werden. Selbst Personen ohne Flickr-Account haben diesen selbstständig erstellen können. Das Verlinken eines Bildes mit valider Lizenz ist durch den Leitfaden des Dialoges ohne externe Hilfe gelungen.

✔ Alle Funktionen konnten von unserer Zielgruppe intuitiv durchgeführt werden. Die Navigation durch die App konnte von allen Testpersonen selbstständig durchgeführt werden.

Praktikabilität

Alle im Pflichtenheft spezifizierten Ansichten müssen innerhalb von maximal fünf Interaktionen nach Starten der App erreichbar sein.

Es sind im Pflichtenheft vier Ansichten definiert:

- Speiseplanansicht → 1 Klick
  - Starten der App
- Favoritenansicht → 2 Klicks
  - Starten der App
  - Klick auf das Favoritenicon in der Navigationsleiste
- Einstellungsansicht → 2 Klicks
  - Starten der App
  - Klick auf das Einstellungsicon in der Navigationsleiste
- Detailansicht → 2 – 5 Klicks
  - Starten der App
  - Wechseln des Tages, falls kein Gericht vorhanden (max. 3 Klicks (Klick auf die Datumsauswahl, Klick auf Monat, Klick auf Tag)

- Klick auf das Favoritenicon in der Navigationsleiste

✔ Erfüllt, alle Ansichten sind in zwei bis fünf Klicks inkl. Starten der App zu erreichen. Dialoge werden hier nicht betrachtet.

Zuverlässigkeit

> Die App kommuniziert eigene Fehler durch Fehlermeldungen an den Nutzer und wird dabei aufgrund von unbehandelten Fehlern nicht vom Betriebssystem beendet, soweit dies nicht auf Probleme des Geräts, wie zum Beispiel die Überlastung des Arbeitsspeichers zurückzuführen ist.

Im Testbetrieb konnten keine Abstürze festgestellt werden. Auch werden bei eigenen Fehlern (also meist eine fehlende Verbindung zum Server) durch Fehlermeldungen dem Nutzer kommuniziert.
✔ Erfüllt.

Administrierbarkeit

> Der Server muss Ereignisse und Fehler in unterschiedlichen Protokollierungsstufen protokollieren.

Es existieren die folgenden Protokollierungsstrufen:

Stufe	Verwendung
TRACE	Informationen über Hintergrundaktivitäten wie API- oder Web-Anfragen
DEBUG	Ereignisse wichtiger als TRACE aber nicht unbedingt relevant, z.B. fehlgeschlagene API-Anfragen
INFO	Relevante Informationen über den Zustand des Systems
WARN	Auftreten von Fehlern, welche jedoch nicht zum Abbruch des aktuellen Vorgangs führen.
ERROR	Fatale Fehler, die vielleicht nicht direkt zum Programmabbruch, aber zum Abbruch des aktuellen Vorgangs führen

Folgende Ereignisse werden protokolliert:

INFO	Starten des Servers und erfolgreicher Start
ALWAYS	Fehler beim Starten des Servers
INFO	Starten und beenden einer Speiseplanaktualisierung (heute und nächste vier Wochen) inklusive möglichen Fehlern
TRACE	Auflösen und einfügen von Speiseplänen (Mensen, Linien, Gerichten) Gerichten in die Datenbank
WARN	Fehler dabei
TRACE	Senden einer Web-Anfrage an die Mensa-Webseite
TRACE	Eingehende API-Anfragen
DEBUG	Fehler bei API-Anfragen
INFO	Löschen von beim Bildhoster nicht mehr existierenden Bildern

INFO Starten und Beenden des Vorgangs zum automatischen Ausblenden von zu oft gemeldeten Bildern inklusive möglichen Fehlern

TRACE Senden einer Anfrage an die externe Flickr-API

INFO Laden von Konfigurationen aus den Umgebungsvariablen

✔ Damit sind alle im Pflichtenheft spezifizierten Ereignisse abgedeckt.

Übertragbarkeit

Die App muss auf Android-Geräten, mit einer Android SDK Version von mindestens 19, lauffähig sein.

✔ Die App kann auf Android Geräten mit SDK Version 19 oder höher ausgeführt werden.

Transparenz

Der Quellcode des Produkts muss unter der MIT-Lizenz [5] veröffentlicht werden.

✔ Siehe `https://github.com/kronos-et-al/MensaApp/blob/main/LICENSE`

Datenschutz

Das Produkt muss sich an die Datenschutzgrundverordnung (DSGVO) halten. Dies ist insbesondere für die Bewertungen und Verlinkung von Bildern relevant, welche serverseitig gespeichert werden. Außerdem muss dem Nutzer eine Anlaufstelle mitgeteilt werden, um alle seine gespeicherten personenbezogenen Daten abfragen oder löschen zu können.

✔ Siehe `https://mensa-ka.de/privacy.html`

Barrierefreiheit

Die App muss die vom Flutter-Framework empfohlenen Richtlinien zur Barrierefreiheit[6] einhalten.

Dabei werden folgende Punkte eingehalten:

- Interaktionen: Nach jeder Interaktion wird eine Änderung oder ein Banner, der über eine Änderung informiert, angezeigt.
- Screenreader: Alle Buttons lassen sich vorlesen.
- Kontrast: Die Richtlinien werden sowohl im Dark- als auch im Lightmode eingehalten. (Bei den Farben Schwarz und Weiß stimmt das trivialerweise. Die Grüntöne wurden so gewählt, dass das geforderte Verhältnis von 4.5 : 1 eingehalten werden konnte.)
- Kontextwechsel und Error: Im Allgemeinen können alle Änderungen, die lokal gespeichert werden, ohne Probleme wieder geändert werden. Bei Änderungen mit Serverinteraktion wird entweder auf eine Bestätigung des Nutzers gewartet oder sie lassen sich durch einen Klick rückgängig machen (eine funktionierende Verbindung wird vorausgesetzt).
- Schaltflächen: Alle Schaltflächen haben min. die Größe 48x48px.

[5] `https://mit-license.org/`
[6] `https://docs.flutter.dev/accessibility-and-localization/accessibility`

- Farbsehen: Die App ist auch in Schwarz/Weiß benutzbar.
- Schriftgröße: Die App wird auch bei großen Schriftgrößen richtig angezeigt.

✔ Damit hält die App die Richtlinien in großen Teilen ein.

Übersetzbarkeit

Die Nutzeroberfläche der App muss ohne Änderungen im Programmcode in andere natürliche Sprachen übersetzt werden können.

Alle Texte, die von der App angezeigt werden (und nicht vom Server kommen, wie zum Beispiel Namen von Gerichten) sind an einem zentralen Ort gespeichert. Will man eine Sprache hinzufügen, so muss man an diesem Ort alle Texte für diese Sprache und die Sprache zur Auswahl der Sprachen hinzufügen. Dabei wird kein Programmcode geändert.

✔ Erfüllt

Sprache

Die Nutzeroberfläche muss auf Deutsch gestaltet sein.

Durch die Gewährleistung der Übersetzbarkeit werden alle Texte (die nicht vom Server und somit vom Studierendenwerk kommen) an einem zentralen Ort pro Sprache abgespeichert. Dabei wurde nur die deutsche Sprache in der Entwicklung berücksichtigt.

✔ Erfüllt

Änderbarkeit

Das Produkt muss modular aufgebaut sein, um zukünftige Erweiterungen zu ermöglichen. Dafür sind die SOLID-Designprinzipien einzuhalten.

Die Designprinzipien sind aus folgenden Gründen erfüllt:

- ✔ **Single-Responsibility-Prinzip**
  Wir haben dieses Prinzip dadurch erfüllt, dass wir die Gesamtfunktionalität in einzelne Module, Klassen und Funktionen aufgeteilt haben, die jeweils nur einen Teil der Funktionalität der übergeordneten Klasse bzw. des übergeordneten Moduls oder des Produktes implementieren.
- ✔ **Open-Closed-Prinzip**
  Dieses Prinzip haben wir durch die Verwendung von Interfaces in Front- und Backend erfüllt.
- ✔ **Liskovsches Substitutionsprinzip**
  Im Backend ist dieses Prinzip automatisch erfüllt, da Rust keine Vererbungsfunktionalität im klassischen Sinne bietet. Im Frontend wird nur von abstrakten Klassen geerbt, sodass das LSP dort automatisch erfüllt ist.
- ✔ **Interface-Segregation-Prinzip**
  Im Backend haben wir unsere Interfaces so aufgeteilt, sodass diese nur die Funktionen an die Schichten darüber freigeben, die diese auch benötigen. Im Frontend wurde dies an den Stellen getan, an denen es sinnvoll war, allerdings wurden an einigen Stellen auch größere Interfaces verwendet.
- ✔ **Dependency-Inversion-Prinzip**
  Dieses Prinzip haben wir durch die Verwendung von Interfaces zur Trennung zwischen den Schichten

in Front- und Backend erfüllt, wobei eine Komponente immer das Interface und niemals von einer konkreten Implementierung abhängt.

Portierbarkeit

Die App muss einfach auf weitere Zielplattformen wie iOS oder das Web übertragbar sein. Konkret muss der Aufwand hierfür geringer als der einer Neuentwicklung für diese Plattformen sein.

Durch die Nutzung von Flutter kann die App mit wenig Mehraufwand für weitere Zielplattformen zur Verfügung gestellt werden. Sie wurde zum Beispiel schon auf Linux und Windows erfolgreich verwendet, ohne dafür nennenswerte Änderungen vorzunehmen.

✔ Erfüllt.

6 Liste aller behobenen Bugs

Während der Qualitätssicherung haben sich Fehler in der Anwendung gezeigt, die behoben werden mussten.

6.1 Frontend

Der Preis der Gerichte aktualisiert sich nicht

Fehlerhafte Komponenten:

- View `Speiseplanansicht`
- Komponente `View`

Kurzbeschreibung: Wenn der Nutzer in den Einstellungen seine Preisklasse anpasste, konnte er keinen Preiswechsel im Speiseplan feststellen. Dieses Fehlverhalten betraf nur die Listenansicht und wurde durch Synchronisation behoben.
Behoben und getestet von *Alexander Kutschera* ✔

Das Wechseln einer Mensa ist nicht möglich

Fehlerhafte Komponenten:

- View `Speiseplanansicht`
- Komponente `Database`

Kurzbeschreibung: Nachdem die Mensa im Drop-Down gewechselt wurde, konnte sie nicht mehr ausgewählt werden oder bei Betätigung passierte kein Wechsel. Verantwortlich war ein Fehler in der Datenbank. Dieser Fehler wurde behoben.
Behoben und getestet von *Elena Häußler* ✔

Gerichte mit einer Null-Sterne-Bewertung werden beim Filtern nicht angezeigt

Fehlerhafte Komponenten:

- View `Speiseplanansicht`
- Komponente `MealData`

Kurzbeschreibung: Gerichte mit null Sternen wurden von dem Filter ausgeschlossen, wenn nach einer minimalen Bewertung gefiltert wurde. Da null Sterne bedeutet, dass ein Gericht noch nicht bewertet wurde, ist es wünschenswert, dass diese angezeigt werden. Das Fehlverhalten wurde korrigiert.
Behoben und getestet von *Elena Häußler* ✔

Sortierung wird nicht übernommen

Fehlerhafte Komponenten:

- View `Speiseplanansicht`
- View `Filterdialog`
- Komponente `MealData`
- Komponente `View`

Kurzbeschreibung: Die Sortierung wurde nicht übernommen. Der Fehler wurde behoben.
Behoben und getestet von *Elena Häußler* ✔

Bilder werden nach dem Melden weiterhin angezeigt

Fehlerhafte Komponenten:

- View `Speiseplanansicht`
- View `Detailansicht`
- View `Galeriedialog`
- View `Meldedialog`
- Komponente `Database`

Kurzbeschreibung: Ein Bild wurde dem Nutzer direkt nach dem Melden noch angezeigt. Durch das Löschen des Bildes in der Datenbank und im Meal-Objekt, wird das Bild entfernt und das UI wird aktualisiert. Der genannte Fehler tritt nicht mehr auf.
Behoben und getestet von *Elena Häußler* ✔

Favoriten können nicht entfernt werden

Fehlerhafte Komponenten:

- View `Favoritenansicht`
- Komponente `Database`

Kurzbeschreibung: Ausgewählte favorisierte Gerichte konnten nicht mehr aus den Favoriten entfernt werden. Grund dafür war ein Fehler in der Datenbank, der beseitigt wurde. Dieses Problem tritt nicht mehr auf.
Behoben und getestet von *Alexander Kutschera* ✔

Temporäre Fehlermeldungen im Darkmode

Fehlerhafte Komponenten:

- View `alle Banner`
- Komponente `View`

Kurzbeschreibung: Fehlermeldungen konnten im Dark-Mode nicht erkannt werden, da der Kontrast zu niedrig war. Durch einen besseren Kontrast sind die Fehlermeldungen im Dark- und Lightmode inzwischen deutlich erkennbar.
Behoben und getestet von *Alexander Kutschera* ✔

Bewertungen werden nicht richtig aktualisiert

Fehlerhafte Komponenten:

- Komponente `MealData`

Kurzbeschreibung: Bewertungen konnten nur aktualisiert werden, indem man die App neu gestartet hat. Durch eine Aktualisierung der UI, konnte der Fehler behoben werden.
Behoben und getestet von *Alexander Kutschera* ✔

Favoriten können nicht hinzugefügt werden

Fehlerhafte Komponenten:

- Komponente `Database`

Kurzbeschreibung: Favoriten konnten nicht hinzugefügt werden. Der Grund dafür war ein Fehler in der Datenbank. Favoriten können nun hinzugefügt werden.
Behoben und getestet von *Alexander Kutschera* ✔

Favoritenstatus wird nicht global aktualisiert

Fehlerhafte Komponenten:

- Komponente `View`

Kurzbeschreibung: Nach dem Entfernen eines Favoriten in der Favoritenansicht wird das Gericht in der Speiseplanansicht immer noch als Favorit angezeigt. Das Problem wurde behoben.
Behoben und getestet von *Alexander Kutschera* ✔

Allergene überlappen in der Filteransicht

Fehlerhafte Komponenten:

- View `F`ilterdialog
- Komponente `View`

Kurzbeschreibung: Die Filteransicht zeigte bei Endgeräten mit verschiedenen Displaygrößen die Allergene in der Filteransicht nicht richtig an. Dabei waren alle Allergene übereinander und eine selektive Auswahl war nicht möglich. Der Grund war ein Fehler in einer Berechnung. Der Fehler wurde behoben.
Behoben und getestet von *Alexander Kutschera* ✔

Der Galeriedialog wirft Fehler, sobald kein Bild mehr geladen werden kann

Fehlerhafte Komponenten:

- View `G`aleriedialog
- Komponente `View`

Kurzbeschreibung: Es wurde eine Exception angezeigt, wenn keine Verbindung zum Server vorhanden war. Dies wurde durch hinzugefügtes Error-Handling gelöst. Dieser Fehler wird nun behandelt.
Behoben und getestet von *Alexander Kutschera* ✔

6.2 Backend

Fehlerhafte Flickr-URL-Entschlüsselung

Fehlerhafte Komponenten:

- Komponente `FlickrApi`

Kurzbeschreibung: Die API akzeptiert bei den meisten Anfragen mehrere Versionen einer `photo_id`. Diese `photo_id` muss mit `Base58` aus der übergebenen „short-url“ decodiert werden, da sonst, wie zuvor erwähnt, manche Anfragen an die API scheiterten.
Behoben und getestet von *Alexander Albers* ✔

Flickr Lizenz wird initial nicht erkannt

Fehlerhafte Komponenten:

- Komponente `FlickrApi`

Kurzbeschreibung: Möchte man ein neu hochgeladenes Bild bei Flickr über die App verlinken, wurde es nicht akzeptiert, da Flickr die initiale Lizenz eines Bildes sonderbar behandelt. Es wurde ein Fall für neu hochgeladene Bilder in der Flickr-Komponente erstellt. Das Problem wurde dadurch behoben.
Behoben und getestet von *Alexander Albers* ✔

Ähnliche Gerichte im gleichen Speiseplan

Fehlerhafte Komponenten:

- Komponente `MealplanManagement`
- Komponente `Database`

Kurzbeschreibung: Gerichte im selben Speiseplan eines Tages wurden als gleich angesehen. Dadurch wurden manche Gerichte nicht in die Datenbank eingefügt und somit nicht im Speiseplan der App angezeigt.
Behoben und getestet von *Alexander Albers* ✔

Stack-Overflow beim Verlinken eines Bildes

Fehlerhafte Komponenten:

- Komponente `FlickrApi`

Kurzbeschreibung: Beim Aufrufen jeglicher FlickrAPI-Funktionen kam es aufgrund eines ungewollten fehlerhaften rekursiven Aufrufs zu einem Stack-Overflow.
Behoben und getestet von *Jonatan Ziegler* ✔

Bewertung eines Gerichtes kann nicht aktualisiert werden

Fehlerhafte Komponenten:

- Komponente `Database`

Kurzbeschreibung: Beim Bewerten eines Gerichtes kam es zu einem Schlüssel-Fehler in der Datenbank, wenn schon eine Bewertung von dem Nutzer vorhanden war. In solchen Fällen wird nun die alte Bewertung überschrieben.
Behoben und getestet von *Jonatan Ziegler* ✔

„GESCHLOSSEN"-Gerichte ignorieren

Fehlerhafte Komponenten:

- Komponente `SwKaParser`

Kurzbeschreibung: Wenn eine Linie geschlossen hat, hat diese manchmal ein einziges „Gericht" mit dem Namen „GESCHLOSSEN". Diese „Gerichte" werden ignoriert, sodass die Liste an Gerichten der Linie leer ist, was per Konvention bedeutet, dass die Linie geschlossen ist.
Behoben und getestet von *Peer Booken* ✔

Alter eines Bildes in Administrator-E-Mail

Fehlerhafte Komponenten:

- Komponente `Mail`

Kurzbeschreibung: Das Alter des Bildes wurde vorher nicht in der Mail als Information mitgesendet. Dies wurde dann behoben.
Behoben und getestet von *Peer Booken* ✔