



**Fernfachhochschule
Schweiz**

Mitglied der SUPSI

Keeley

Modularbeit

von

Ivo Eichelberger und Nino Ziswiler

Modul: INF-W-AF006, Mobile App Development (MAD)

Auftrag: Modularbeit

Autoren: Ivo Eichelberger und Nino Ziswiler

Verteiler: Valentin Manthei

Datum: 17. Juni 2025

Version: 1.0

Inhaltsverzeichnis

1	Projektidee	4
2	Grobanforderungen.....	5
2.1	Muss-Anforderungen	5
2.1.1	Authentifikation	5
2.1.2	Belegerfassung per Kamera.....	5
2.1.3	Import von Belegen	5
2.1.4	Übersicht der Buchungen.....	6
2.1.5	Dashboard	6
2.2	Kann-Anforderungen	6
2.2.1	Automatische Texterkennung.....	6
2.2.2	Auswertung nach Kategorien	6
2.2.3	Offline-Modus	6
2.3	Alternative Umsetzung mit eigenständigem Backend	7
3	Anwendungsfalldiagramm	8
4	User Stories.....	9
4.1	Optionale User Stories	9
5	Wireframes	10
5.1	Login	10
5.2	Dashboard	11
5.3	Formular.....	12
5.4	Liste	13
6	Entwicklungsblock 1	14
6.1	Ziel des Entwicklungsblocks.....	14
6.2	Git-Strategie.....	14
6.3	Tätigkeiten	14
6.3.1	Projektsetup und Architektur	14
6.3.2	Statemanagement	14
6.3.3	Design-System und Theme.....	14
6.3.4	Navigation und Routing	15
6.3.5	Screen-Layouts.....	15
7	Entwicklungsblock 2	15
7.1	Ziele	15
7.2	Tätigkeiten	15
7.2.1	Authentifizierung mit Firebase.....	15

7.2.2	Benutzeroberfläche für die Belegerfassung.....	16
7.2.3	Erste Interaktionen mit Mock-Daten.....	16
7.2.4	State Management & Datenfluss	16
7.2.5	Design & Usability	16
8	Entwicklungsblock 3	17
8.1	Ziel des Entwicklungsblocks.....	17
8.2	Git-Strategie.....	17
8.3	Tätigkeiten	17
8.3.1	Theme-Implementation und Design-System	17
8.3.2	Backend-Integration mit Firebase	17
8.3.3	3screen-Updates mit Backend-Anbindung	17
8.3.4	State-Management und Error-Handling	18
8.3.5	CRUD-Operationen auf Booking.....	18
8.3.6	Architektur-Verbesserungen und Refactoring.....	18
9	Technische Lokale Einrichtung des Projekts und Firebase-Integration	19
9.1	Voraussetzungen.....	19
9.2	Installation und Start.....	19
9.3	Firebase Authentication.....	19
9.4	Firebase Firestore	21
9.5	Firebase Analytics	22
9.6	Firebase Storage	22
10	Verwendung von Riverpod für State Management	23
10.1	Integration in das Projekt.....	23
10.2	Verwendung in der Appstruktur.....	24
10.3	AsyncValue für Feedback & Fehlerbehandlung	24
10.4	Vorteile der Lösung	25
11	Projektstruktur, Theme und Wiederverwendbarkeit	25
11.1	Projektstruktur.....	25
11.2	Wiederverwendbare Widgets	25
11.3	Keeley Theme (Farbschema & Designsystem)	26
12	Reflexion und Teamarbeit	27
	Abbildungsverzeichnis	28
	Quellenverzeichnis.....	29
	Hilfsmittelverzeichnis.....	29

1 Projektidee

Selbständige und Privatpersonen stehen täglich vor der Herausforderung, ihre Belege ordnungsgemäss zu verwalten und zu organisieren. Quittungen und Rechnungen sammeln sich an, werden in Schubladen vergessen oder gehen auf Geschäftsreisen verloren. Die manuelle Sortierung ist zeitaufwendig, fehleranfällig und führt oft zu Stress, besonders wenn die Steuererklärung ansteht. Unsere mobile Applikation löst dieses Problem, indem sie eine benutzerfreundliche Lösung für die Belegerfassung unterwegs bietet. Die App fungiert als Ergänzung zu einer bestehenden Buchhaltungssoftware und konzentriert sich auf eine schnelle, unkomplizierte Erfassung von Einnahmen und Ausgaben im Alltag. Nutzer können Belege fotografieren, relevante Buchungsdetails eingeben und diese nahtlos mit der bestehenden Hauptbuchhaltungssoftware synchronisieren. Dies eliminiert nicht nur das Risiko verlorener Dokumente, sondern spart auch wertvolle Zeit im hektischen Geschäftsalltag. Besonders bei der Steuererklärung zeigt sich der grosse Vorteil: Alle benötigten Unterlagen sind zentral in digitaler Form verfügbar, was den Vorbereitungsaufwand drastisch reduziert und die Vollständigkeit der Dokumentation sicherstellt. Die übersichtliche Darstellung aller Transaktionen ermöglicht zudem jederzeit einen klaren Überblick über die finanzielle Situation – ganz gleich, ob im Büro oder unterwegs.



Abbildung 1: Logo

2 Grobanforderungen

In diesem Kapitel werden die Grobanforderungen an die Applikation näher beschrieben.

2.1 Muss-Anforderungen

Nachfolgend sind die Anforderungen aufgeführt, die zwingend umgesetzt werden müssen.

2.1.1 Authentifikation

Die App ermöglicht es Nutzern, sich mit ihrem bestehenden Benutzerkonto der vorhandenen Buchhaltungssoftware anzumelden oder ein neues Benutzerkonto direkt in der App zu erstellen. Dadurch wird sichergestellt, dass sämtliche erfassten Daten automatisch dem korrekten Benutzerkonto zugeordnet werden. Die Authentifikation umfasst folgende Punkte:

- Neuregistrierung eines Benutzerkontos direkt über die mobile App mit Eingabe folgender Daten:
 - E-Mail-Adresse & Passwort
 - Vor- und Nachname
- Anmeldung mittels bestehender E-Mail-Adresse und Passwort der Buchhaltungssoftware.
- *Optional: Nutzung von biometrischer Authentifizierung (z.B. Face-ID, Touch-ID) zur schnellen und sicheren Anmeldung.*

2.1.2 Belegerfassung per Kamera

Die App ermöglicht das Fotografieren eines Belegs direkt mit der Smartphone-Kamera. Nutzer können nach der Aufnahme manuell Informationen eingeben:

- Datum
- Betrag
- Beschreibung
- Kategorie

Die erfassten Daten und das Foto werden in der bestehenden Datenbank abgespeichert.

2.1.3 Import von Belegen

Nutzer können bestehende Bilder oder PDFs vom Gerät importieren. Importierte Dokumente lassen sich in derselben Form bearbeiten wie direkt fotografierte Belege, beschrieben in «2.1.2 Belegerfassung per Kamera».

2.1.4 Übersicht der Buchungen

Nutzer erhalten eine übersichtliche Liste aller bereits erfassten Buchungen. Die Liste zeigt mindestens folgende Informationen pro Buchung:

- Datum
- Betrag
- Beschreibung

2.1.5 Dashboard

Die App zeigt die aktuelle Bilanz (Einnahmen abzüglich Ausgaben) für einen definierbaren Zeitraum. Zusätzlich sind im Dashboard die letzten Buchungen ersichtlich.

2.2 Kann-Anforderungen

Im Folgenden sind die zusätzlichen Anforderungen beschrieben, die bei ausreichender Zeit umgesetzt werden können.

2.2.1 Automatische Texterkennung

Automatisches Auslesen von Daten (Datum, Betrag) aus Belegfotos mittels Texterkennung (Vermutlich mithilfe von KI).

2.2.2 Auswertung nach Kategorien

Benutzer erhalten ein Dashboard, das eine Übersicht der Ausgaben und Einnahmen nach Kategorie bietet.

2.2.3 Offline-Modus

Ermöglichen, dass Buchungen auch ohne Internetverbindung erfasst und später synchronisiert werden können.

2.3 Alternative Umsetzung mit eigenständigem Backend

Die beschriebene Projektidee lässt sich grundsätzlich auch unabhängig von der bestehenden Buchhaltungssoftware umsetzen, falls dies besser zu den Anforderungen des Modulstoffs oder den Rahmenbedingungen der Modularbeit passt. In diesem Fall wäre ein Backend-as-a-Service (BaaS) einzusetzen, um die Daten der mobilen Applikation zu speichern und zu verwalten.

Dabei blieben die Kernanforderungen der Applikation unverändert, lediglich die Datenhaltung und -verwaltung würde nicht über die bestehende Applikation, sondern über ein separat entwickeltes oder bereitgestelltes Backend erfolgen.

Die Applikation würde somit nicht mehr als Ergänzung zu einer bestehenden Buchhaltungssoftware fungieren, sondern als eigenständiges Finanztool dienen, mit dem die Einnahmen und Ausgaben im Blick gehalten und kategorisiert werden können.

.

3 Anwendungsfalldiagramm

Auf Grundlage der Projektidee und den Anforderungen wurde das nachfolgende Anwendungsfalldiagramm erstellt, wobei die optionalen zusätzlichen Anwendungsfälle nicht dargestellt sind.

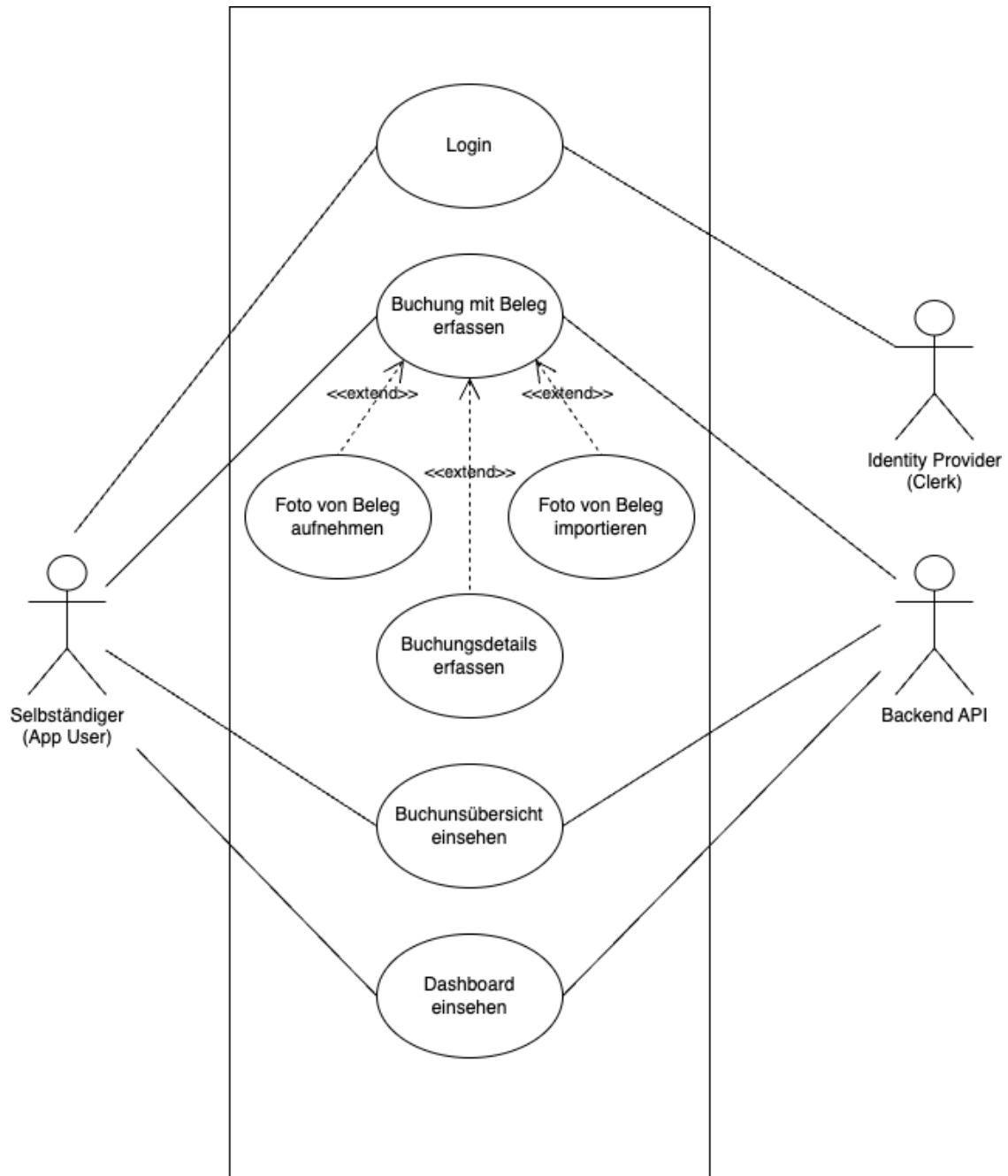


Abbildung 2: Anwendungsfalldiagramm

4 User Stories

Basierend auf den Muss-Anforderungen wurden folgenden User Stories entwickelt:

- Als Selbständiger möchte ich einen Kassenbeleg direkt mit meiner Smartphone-Kamera fotografieren und anschliessend relevante Informationen wie Datum, Betrag, Beschreibung und Kategorie manuell eingeben, damit der Beleg sofort in meiner Buchhaltung erfasst wird.
- Als Selbständiger möchte ich einen Beleg als Bild oder PDF aus meiner Galerie oder einer Cloud importieren, damit ich auch bereits vorhandene oder digitale Belege einfach erfassen kann.
- Als Selbständiger möchte ich eine Übersicht über alle erfassten Buchungen sehen, damit ich jederzeit nachvollziehen kann, welche Ein- und Ausgaben ich bereits erfasst habe.
- Als Selbständiger möchte ich eine Auswertung meiner erfassten Ein- und Ausgaben als Diagramm oder Statistik sehen, damit ich meine Finanzen jederzeit im Blick habe.
- Als Selbständiger möchte ich mich mit meinem bestehenden Benutzerkonto der Buchhaltungssoftware anmelden können, um meine Buchungen automatisch meinem Konto zuordnen zu lassen.
- Als neuer Nutzer möchte ich direkt in der mobilen App ein Benutzerkonto mit meiner E-Mail-Adresse, Passwort sowie Vor- und Nachname erstellen, um sofort mit der Belegerfassung beginnen zu können.

4.1 Optionale User Stories

Im Folgenden werden die optionalen User Stories formuliert:

- Als Selbständiger möchte ich biometrische Authentifizierung (z.B. Face-ID, Touch-ID) verwenden, um mich schnell und sicher in der App anzumelden.
- Als Selbständiger möchte ich, dass Datum und Betrag automatisch aus fotografierten Belegen mittels Texterkennung (KI) ausgelesen werden, um die manuelle Eingabe zu reduzieren und Zeit zu sparen.
- Als Selbständiger möchte ich ein Dashboard mit einer übersichtlichen Auswertung meiner Einnahmen und Ausgaben nach Kategorien sehen, um detaillierte Einblicke in meine Finanzdaten zu erhalten.
- Als Selbständiger möchte ich Buchungen auch ohne Internetverbindung erfassen können, sodass diese später automatisch synchronisiert werden, sobald ich wieder online bin.

5 Wireframes

Die folgenden Wireframes bieten einen visuellen Überblick über das geplante Design und die Funktionalität, sind jedoch noch nicht finalisiert und könnten in der endgültigen Version abweichen.

5.1 Login

The wireframe depicts a mobile application login screen. At the top, a status bar shows the time '9:30' on the left, a black circle in the center, and signal and battery icons on the right. The main content area features the 'KEE LEY' logo, which consists of the words 'KEE' and 'LEY' stacked vertically in white, bold, sans-serif capital letters, set against a square background with a vertical gradient from pink to orange. Below the logo, there are two input fields. The first is labeled 'E-Mail' and contains the text 'neil@tothemoon.ch'. The second is labeled 'Passwort' and is filled with ten dots. A solid black button with the white text 'Anmelden' is positioned below the password field. At the bottom of the form, there are two links: 'Registrieren' on the left and 'Passwort vergessen?' on the right. The entire interface is centered on a white background.

Abbildung 3: Wireframes - Login

5.2 Dashboard

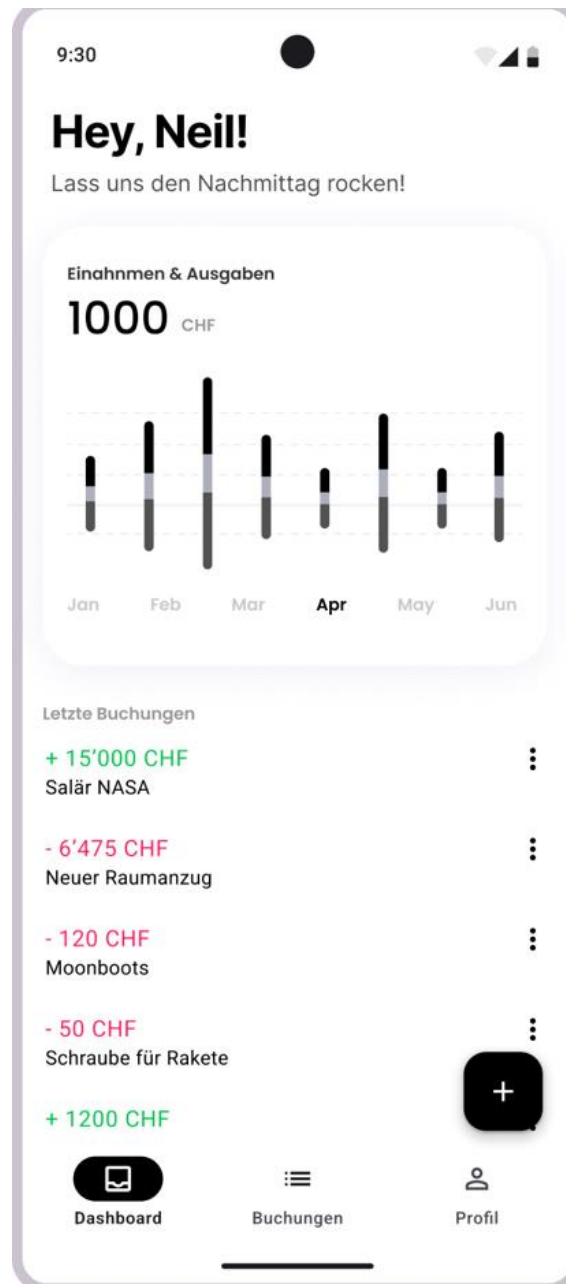


Abbildung 4: Wireframes - Dashboard

5.3 Formular

9:30

Neue Buchung

☐ Einnahme ☐ Ausgabe

Datum

Betrag

Kategorie

Beleg hinzufügen

Abbildung 5: Wireframes - Formular

5.4 Liste

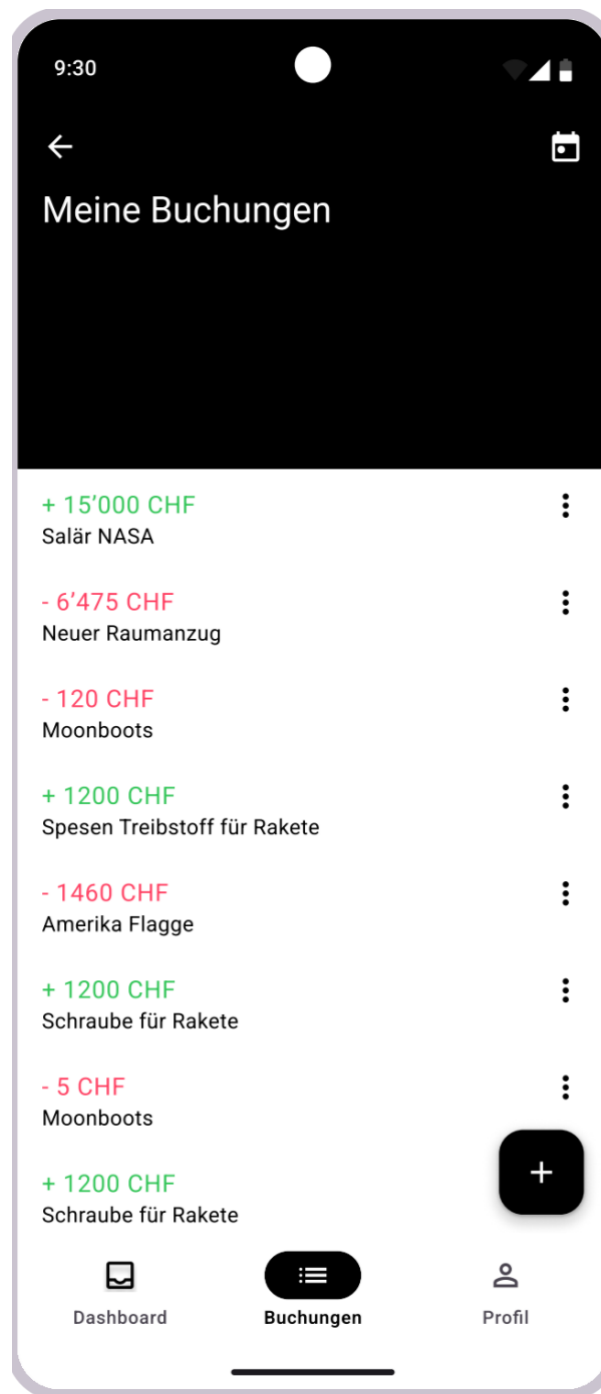


Abbildung 6: Wireframes - Liste

6 Entwicklungsblock 1

Diese Planung definiert die Tätigkeiten für den ersten Entwicklungsblock.

6.1 Ziel des Entwicklungsblocks

Im ersten Entwicklungsblock soll die Grundstruktur der App aufgesetzt werden, einschliesslich des Basis-Layouts und der Menüführung, ohne bereits tiefgreifende Funktionalität zu implementieren. Dies bildet das Fundament für die weitere Entwicklung und ist entscheidend für die nachfolgenden Entwicklungsblöcke, in denen die Authentifizierung und die Anbindung an eine Datenbank realisiert werden.

6.2 Git-Strategie

1. Feature-Branch "entwicklungsblock-1" vom main-Branch abspalten
2. Regelmässige Commits mit aussagekräftigen Nachrichten
3. Pull-Request am Ende des Entwicklungsblocks für Code-Review

6.3 Tätigkeiten

6.3.1 Projektsetup und Architektur

- Flutter-Projekt initialisieren und Grundabhängigkeiten einrichten
- Projektstruktur nach Clean Architecture oder MVVM-Muster aufsetzen
- Definieren der Ordnerstruktur für Features, Widgets, Models und Services
- Einrichtung von Git mit Feature-Branch für diesen Entwicklungsblock

6.3.2 Statemanagement

- Evaluierung und Auswahl des geeigneten Statemanagement-Ansatzes (Provider oder Riverpod)
- Implementierung der Basis-State-Architektur
- Erstellung von Mock-Datenmodellen für die UI-Entwicklung
- Definition von klaren Zustandübergängen zwischen Screens

6.3.3 Design-System und Theme

- Implementierung des App-Themes basierend auf den Wireframes
- Definition von Farbpaletten, Typografie und Abstände
- Erstellung grundlegender UI-Komponenten (Buttons, Cards, Input-Felder)
- Integration des Keeley-Logos und der App-Icons

- Entwicklung von konsistenten Style-Guidelines für die App

6.3.4 Navigation und Routing

- Implementierung der Bottom-Navigation gemäss Wireframes
- Aufsetzen eines Routing-Systems für die App-Screens
- Erstellung leerer Platzhalter-Screens für die Hauptbereiche:
 - Dashboard-Screen
 - Belegerfassungs-Screen
 - Buchungsübersicht-Screen
 - Profil-Screen (optional)
- Navigation zwischen den Screens implementieren
- Einrichtung von Übergängen zwischen Screens

6.3.5 Screen-Layouts

- Entwicklung des Dashboard-Screen-Layouts mit Platzhaltern für Diagramme und Listen
- Erstellung des Layouts für die Belegerfassung mit Kamera- und Galerie-Buttons
- Implementierung des Layouts für die Buchungsübersicht mit einer einfachen Liste
- Gestaltung des Formularlayouts für die Belegdetails

7 Entwicklungsblock 2

7.1 Ziele

Im zweiten Entwicklungsblock wird die Funktionalität der App erweitert, indem die Benutzer-Authentifizierung vollständig implementiert und erste interaktive Funktionalitäten wie Formularverarbeitung, Validierung und Feedback integriert werden. Ziel ist es, die Benutzerführung durch echte Dateninteraktion zu erweitern – aufbauend auf dem bereits erstellten Basislayout.

7.2 Tätigkeiten

7.2.1 Authentifizierung mit Firebase

- Integration von Firebase Authentication (mit E-Mail/Passwort und E-Mail-Link)
- Nutzung von FirebaseUI-Flutter zur Vereinfachung der Authentifizierungs-UI
- Implementierung einer geschützten Navigation: Weiterleitung zu Login, falls der Benutzer nicht authentifiziert ist

7.2.2 Benutzeroberfläche für die Belegerfassung

- Formular für neue Belege mit Feldern:
 - Datum (DatePicker)
 - Betrag (TextField mit Validierung)
 - Beschreibung (TextField)
 - Kategorie (Dropdown)
- Verwendung von Standardkomponenten aus dem UI-Kit (z. B. shadcn_ui)
- Integration von Kamerabutton und Upload-Button (noch ohne Backend-Funktion)
- Implementierung von Formularvalidierung mit sofortigem Feedback

7.2.3 Erste Interaktionen mit Mock-Daten

- Anzeigen erfasster Belege aus lokalen Mock-Daten (Liste)
- Einbindung einer Platzhalter-Karte für jeden Beleg mit Datum, Betrag, Beschreibung
- Visuelle Rückmeldung nach erfolgreichem Erfassen (SnackBar oder Dialog)

7.2.4 State Management & Datenfluss

- Entscheidung für Provider oder Riverpod finalisieren (falls nicht geschehen)
- Einführung eines zentralen App-State zur Verwaltung:
 - Authentifizierter Benutzer
 - Temporäre FormulardatenTrennung von UI und Logik über Controller/ViewModel-Struktur (MVVM)

7.2.5 Design & Usability

- Feinjustierung des App-Themes (z. B. Farben, Schriftgrößen, Icons)
- Einhalten der Styleguides aus Block 1
- Mobile Responsiveness für verschiedene Bildschirmgrößen

8 Entwicklungsblock 3

8.1 Ziel des Entwicklungsblocks

Im dritten Entwicklungsblock soll die App vollständig mit dem Firebase-Backend verbunden und ein einheitliches Design-System implementiert werden. Dabei werden alle Screens mit dem shadcn-Theme versehen, umfassende Error- und Loading-States eingeführt und die Architektur grundlegend überarbeitet. Dies stellt sicher, dass die App eine professionelle Benutzeroberfläche erhält und robust mit dem Backend interagiert.

8.2 Git-Strategie

- Feature-Branch "entwicklungsblock-3" vom main-Branch abspalten
- Regelmässige Commits mit aussagekräftigen Nachrichten
- Pull-Request am Ende des Entwicklungsblocks für Code-Review

8.3 Tätigkeiten

8.3.1 Theme-Implementation und Design-System

- Vollständige Implementierung des shadcn-Themes in der App
- Einheitliche Verwendung des Themes auf allen bestehenden Screens
- Anpassung aller UI-Komponenten an das neue Design-System
- Umsetzung konsistenter Styling-Guidelines
- Überarbeitung der Farbpaletten und Typografie nach Shadcn-Standards

8.3.2 Backend-Integration mit Firebase

- Vollständige Verbindung der App mit dem Firebase-Backend
- Anbindung der Firestore-Datenbank für Datenoperationen
- Implementation von Firebase Analytics für CRUD-Tracking auf Bookings

8.3.3 3screen-Updates mit Backend-Anbindung

- Dashboard-Screen: Backend-Datenintegration und shadcn-Theme
- Profil-Screen: Vollständige Implementierung mit shadcn-Design
- Listen-Screen: shadcn-Komponenten und Backend-Datenanbindung
- Formular: shadcn-Theme und Backendanbindung
- Authentifizierungs-Screens: Design-Update mit shadcn

8.3.4 State-Management und Error-Handling

- Implementierung Loading-States für alle API-Calls
- Entwicklung Error-States und Fehlerbehandlung
- Integration von Toast-Nachrichten und User-Feedback-Systemen
- Optimierung der State-Übergänge zwischen verschiedenen Screens

8.3.5 CRUD-Operationen auf Booking

- Implementierung der Bearbeitungslogik mit Backend-Verbindung
- Integration der Löschfunktionalität mit Bestätigungsdialogen
- Entwicklung der Erstellungslogik für neue Datensätze
- Implementierung von Validierungslogik auf Client- und Server-Seite

8.3.6 Architektur-Verbesserungen und Refactoring

- Umfassende Code-Refactoring für bessere Maintainability
- Implementierung verbesserter Separation of Concerns
- Optimierung der Projektstruktur und Modularisierung

9 Technische Lokale Einrichtung des Projekts und Firebase-Integration

9.1 Voraussetzungen

- Flutter SDK ≥ 3.16
- Dart ≥ 3.2
- Android Studio oder VS Code mit Flutter/Dart Plugins
- Ein Firebase-Projekt (für Authentifizierung, Datenbank und Analytics)

9.2 Installation und Start

Ist das Projekt geklont. Kann es so lokal in betrieb genommen werden:

- `git clone https://github.com/nziswiler/keeley.mobile.git`
- `cd keeley.mobile`
- `flutter pub get`
- `flutterfire configure`
- `flutter run`

Hinweis: Die Datei `firebase_options.dart` wird automatisch mit `flutterfire configure` erstellt.

9.3 Firebase Authentication

Firebase Authentication wurde für die Benutzerverwaltung verwendet. Dabei kommen E-Mail/Passwort sowie die FirebaseUI-Komponenten zum Einsatz.

Eingesetzte Technologien:

- `firebase_auth`
- `firebase_ui_auth`
- `firebase_core`
- `flutter_riverpod`

Registrierung und Login

- Nutzer können sich via E-Mail und Passwort registrieren.
- Die Authentifizierung erfolgt über einen `CustomSignInScreen` und `CustomSignUpScreen`, die auf den FirebaseUI-Komponenten basieren.
- Biometrische Anmeldung ist optional vorgesehen.

Schutz der Routen

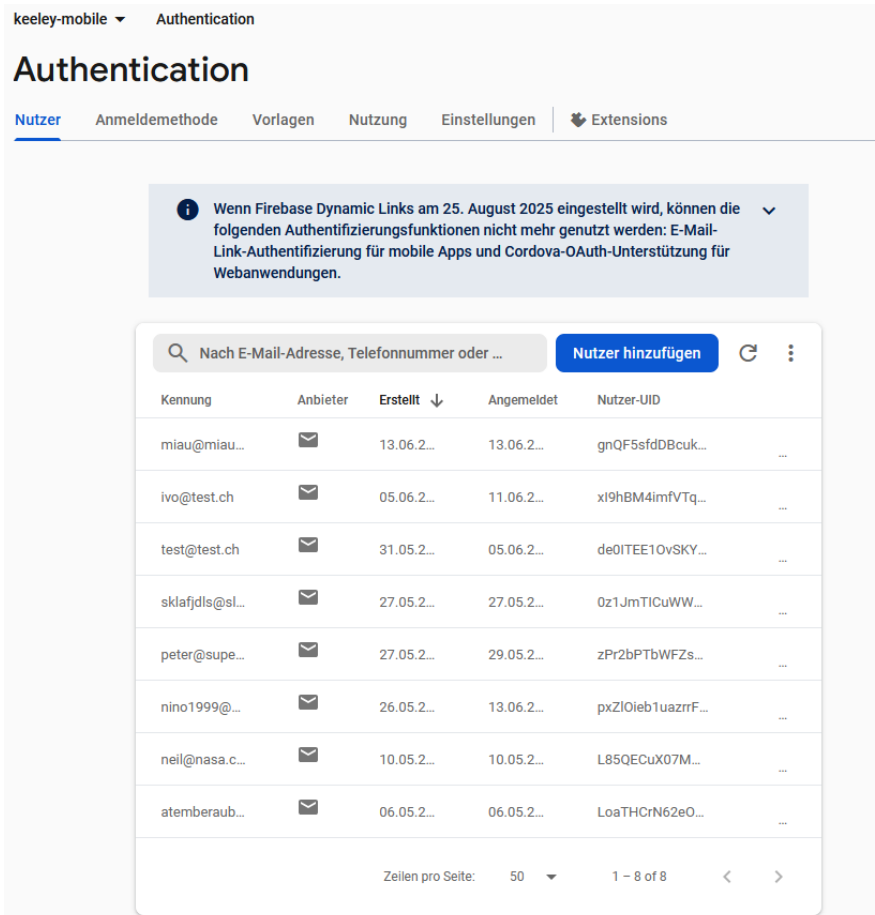
- Nutzer können sich via E-Mail und Passwort registrieren.
- Die Authentifizierung erfolgt über einen CustomSignInScreen und CustomSignUpScreen, die auf den FirebaseUI-Komponenten basieren.
- Biometrische Anmeldung ist optional vorgesehen.

Beispiel für die Redirect-Logik:

```
redirect: (context, state) {  
  final isLoggedIn = authRepository.currentUser != null;  
  if (isLoggedIn && path.startsWith('/signIn')) return '/dashboard';  
  if (!isLoggedIn && !path.startsWith('/signIn')) return '/signIn';  
  return null;  
},
```

Abbildung 7 Beispiel Firebase Auth

So sehen die Userprofile in Firebase aus. Es kann auch optional eine mehrfache Authentifizierung eingerichtet werden.



The screenshot shows the Firebase Authentication console for the project 'keeley-mobile'. The 'Nutzer' (Users) tab is selected. A notification banner at the top states: 'Wenn Firebase Dynamic Links am 25. August 2025 eingestellt wird, können die folgenden Authentifizierungsfunktionen nicht mehr genutzt werden: E-Mail-Link-Authentifizierung für mobile Apps und Cordova-OAuth-Unterstützung für Webanwendungen.' Below the banner is a search bar with the text 'Nach E-Mail-Adresse, Telefonnummer oder ...' and a 'Nutzer hinzufügen' button. A table lists the following users:

Kennung	Anbieter	Erstellt ↓	Angemeldet	Nutzer-UID
miau@miau...	📧	13.06.2...	13.06.2...	gnQF5sfDBcuk...
ivo@test.ch	📧	05.06.2...	11.06.2...	xl9hBM4imfVTq...
test@test.ch	📧	31.05.2...	05.06.2...	de0ITEE10vSKY...
sklafjds@sl...	📧	27.05.2...	27.05.2...	0z1JmTICuWW...
peter@supe...	📧	27.05.2...	29.05.2...	zPr2bPTbWFZs...
nino1999@...	📧	26.05.2...	13.06.2...	pxZlOieb1uazrrF...
neil@nasa.c...	📧	10.05.2...	10.05.2...	L85QECuX07M...
atemberaub...	📧	06.05.2...	06.05.2...	LoaTHCrN62eO...

At the bottom of the table, it shows 'Zeilen pro Seite: 50' and '1 - 8 of 8'.

Abbildung 8 Firebase Authentication

9.4 Firebase Firestore

Die App speichert Buchungsdaten der Nutzer in der Firebase Firestore Datenbank.

Struktur

- Jede Nutzer-ID hat ein Dokument unterhalb von users/{uid}
- Innerhalb dieses Dokuments existiert eine Subcollection bookings mit den einzelnen Bookings

Beispiel:

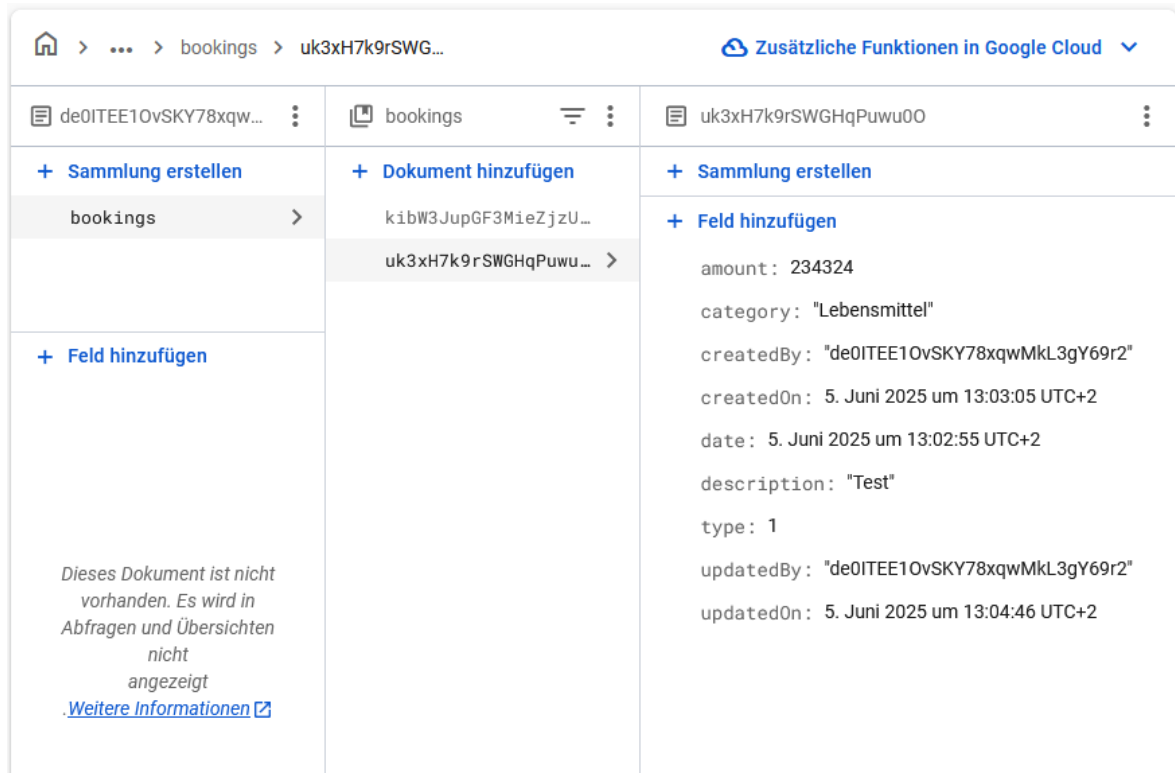


Abbildung 9 Firebase Firestore

Zugriff über BookingService:

- Über bookingServiceProvider wird eine Methode bereitgestellt, die CRUD-Operationen ausführt.
- Der BookingsController ist für das Löschen von Buchungen zuständig:

```
final bookingService = ref.read(bookingServiceProvider);  
await bookingService.deleteBooking(booking.id);
```

Performance

Firestore zeigt derzeit <1000 Lesevorgänge pro Woche und sehr geringe Schreibvorgänge an. Dies ist performant und innerhalb des Spark-Plans (kostenlos)

9.5 Firebase Analytics

Firebase Analytics wurde aktiviert, um das Nutzerverhalten zu analysieren und das Produkt iterativ zu verbessern.

Funktionen:

- Tracking von Logins, App Starts und Bildschirmwechseln
- Daten werden in der Firebase Console aufgeschlüsselt
- Nutzung erfolgt DSGVO-konform ohne Rückschlüsse auf personenbezogene Daten

Konfiguration

- Analytics ist mit dem Firebase-Projekt verknüpft und wurde bei der flutterfire configure-Einrichtung automatisch registriert
- Es wurden einige benutzerdefinierte Events integriert (z. B. booking_added, user_signed_in)

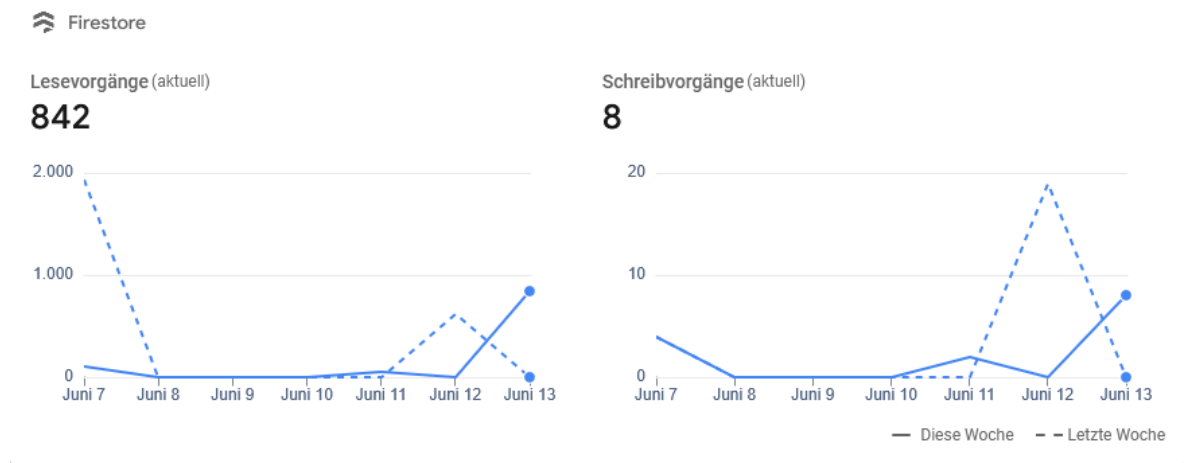


Abbildung 10 Firebase Analytics

9.6 Firebase Storage

Obwohl Firebase Storage im Projekt sichtbar ist, wurde dieser im aktuellen Entwicklungsstand noch nicht produktiv genutzt, da dafür Kosten anfallen könnte. Eine Integration zur Speicherung von Belegfotos ist vorgesehen für eine mögliche Erweiterung. Das UI dafür besteht gemäss Anforderungen bereits.

10 Verwendung von Riverpod für State Management

Riverpod ist ein Framework zur Zustandsverwaltung in Flutter. Es wurde als Weiterentwicklung von Provider konzipiert und ermöglicht die Trennung von UI und Anwendungslogik. Dabei definiert man sogenannte Provider, die Daten und Zustände bereitstellen, etwa für Authentifizierung, API-Aufrufe oder Formularvalidierung. Riverpod unterstützt verschiedene Typen von Zustand, darunter synchrone (StateProvider) und asynchrone (FutureProvider, AsyncNotifier) Varianten. Durch die zentrale Verwaltung und Typsicherheit erleichtert Riverpod das Testen, Strukturieren und Erweitern von Flutter-Apps.

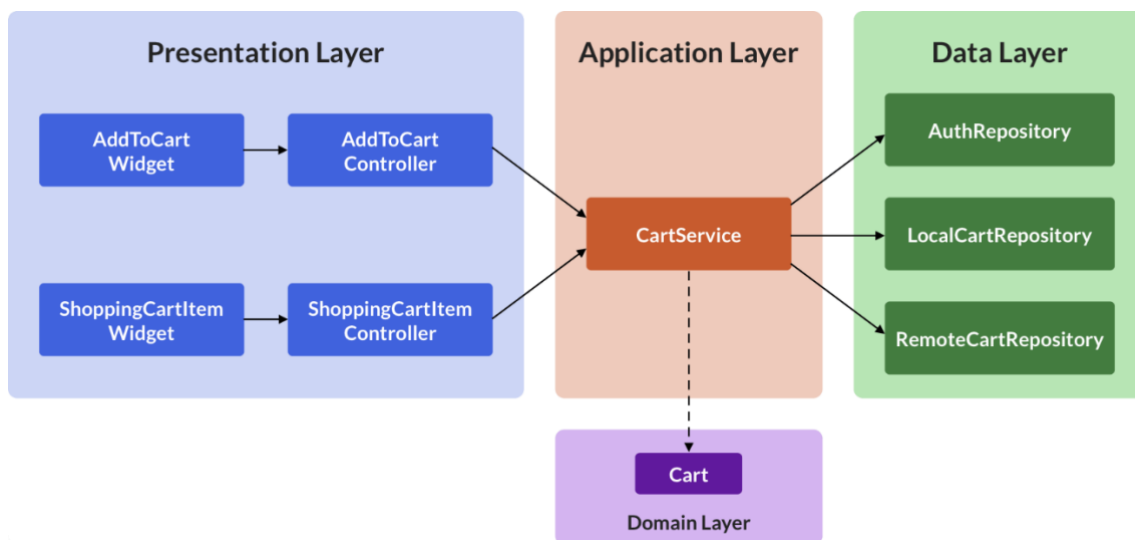


Abbildung 11 Flutter Architektur

10.1 Integration in das Projekt

Die Anwendung verwendet `flutter_riverpod` gemeinsam mit `riverpod_annotation`, um eine saubere Trennung von UI und Geschäftslogik zu gewährleisten. Alle Provider werden durch die Verwendung von `@riverpod` automatisch generiert (.g.dart Dateien).

10.2 Verwendung in der Appstruktur

Die Provider sind in Feature-Module aufgeteilt (z. B. auth, bookings, dashboard).

- **authProviders.dart**: Stellt eine Liste verfügbarer Authentifizierungsoptionen bereit.
- **AuthController**: Steuert Authentifizierungsprozesse (Login, Registrierung, Logout, Profilaktualisierung, Nutzerlöschung) und nutzt AsyncValue zur Statusverwaltung.

Beispiel:

```
state = const AsyncLoading();
try {
  await authRepository.signOut();
  state = const AsyncData(null);
} on Exception catch (e) {
  state = AsyncValue.error(e, StackTrace.current);
}
```

Abbildung 12 Beispiel Riverpod

- **BookingsController**: Verwaltet den Lade-, Fehler- und Erfolgszustand bei CRUD-Operationen auf Buchungen.
- **dashboardServiceProvider** & **monthlyStatsProvider** & **categoryExpensesProvider**:
 - Aggregieren statistische Werte zu Einnahmen/Ausgaben und Kategorien basierend auf den Einträgen aus Firestore
 - Nutzen dazu getBookingsInDateRange aus dem BookingRepository.

10.3 AsyncValue für Feedback & Fehlerbehandlung

Die State-Klassen nutzen AsyncValue, um Lade- und Fehlerzustände korrekt abzubilden.

Damit wird sichergestellt, dass die UI auf jeden Status reagieren, kann:

```
switch (controller) {
  case AsyncLoading(): return CircularProgressIndicator();
  case AsyncError(:final error): return Text('Fehler: \$error');
  case AsyncData(:final data): return SuccessView();
}
```

Abbildung 13 Beispiel Errorhandling

10.4 Vorteile der Lösung

- Typsicherheit durch generierte Provider
- Trennung von UI und Logik gemäss MVVM-Prinzipien
- Effizientes Re-Rendering nur bei Änderungen im State
- Hohe Testbarkeit durch entkoppelte Provider-Logik

Riverpod bildet das technische Rückgrat der Keeley-App und sorgt für konsistentes State-Handling in sämtlichen Bereichen der Anwendung.

11 Projektstruktur, Theme und Wiederverwendbarkeit

11.1 Projektstruktur

Die Projektstruktur folgt dem Prinzip der Feature-basierten Modularisierung und orientiert sich an Clean Architecture / MVVM:

src/

- |— common/ → Globale Exceptions, Logger, Widgets
- |— constants/ → Zentrale Keys und Strings
- |— features/
 - | |— auth/ → Authentifizierung: Controller, Screens, Widgets
 - | |— bookings/ → Buchungserfassung: Logik, Formulare, Screens
 - | |— dashboard/ → Statistik & Auswertungen
- |— routing/ → GoRouter-Konfiguration und Navigation
- |— theme/ → Keeley Theme und Design-Konstanten
- |— utils/ → Hilfsfunktionen (z. B. Formatierung)
- |— main.dart → Einstiegspunkt mit FirebaseInit und ProviderScope

11.2 Wiederverwendbare Widgets

Die App enthält einige generische Widgets, die mehrfach verwendet werden, hier ein paar Beispiele:

- AuthSubmitButton – kontextabhängiger Button für Authentifizierung
- AuthNavigationLink – Umschalter zwischen SignIn und SignUp
- BookingCard – Darstellung eines einzelnen Belegs mit Datum/Betrag

Diese Komponenten wurden bewusst gekapselt und sind in features/<bereich>/presentation/widgets/ abgelegt.

11.3 Keeley Theme (Farbschema & Designsystem)

Die Anwendung nutzt ein eigenes Theme auf Basis von `shadcn_ui`, dass durch die Datei `keeley_theme.dart` definiert ist. Dort wird ein individuell angepasstes Farbschema für **Light- und Darkmode** definiert:

Besonderheiten des `KeeleyColorScheme`:

- Eigene Farben für „income“ & „incomeForeground“
- Einheitliches Verhalten für primary, destructive, muted etc.
- Erweiterung von `ShadColorScheme`

Beispiel:

`const KeeleyColorScheme.light()` → rosa Primärfarbe, weisse Cards

`const KeeleyColorScheme.dark()` → dunkle Oberfläche, grüne Income-Akzente

`KeeleyChartColors`:

- Definiert eine zentrale Farbpalette für Diagramme
- Wird z. B. für Balken im Dashboard verwendet (chart1–chart5)

Sizes und Gaps:

- Einheitliches Spacing über Sizes (z. B. `Sizes.p16`, `p48`, ...)
- Gaps wie `gapW8`, `gapH16` vereinheitlichen UI-Abstände und machen das Layout pflegbarer

Diese Struktur ermöglicht konsistente Designs, bessere Wiederverwendbarkeit und erleichtert zukünftige Erweiterungen der App. Sie wurde vollständig in die `shadcn_ui`-Bibliothek integriert, was manuelle UI-Dopplungen vermeidet und einheitliches Verhalten über alle Screens hinweg sicherstellt.

12 Reflexion und Teamarbeit

Die Zusammenarbeit im Team hat sehr gut funktioniert. Wir konnten Aufgaben effizient aufteilen und bei Bedarf flexibel unterstützen. Für uns beide war es das erste Mal, dass wir eine mobile App mit Flutter entwickelt haben. Durch unsere Vorkenntnisse aus der Webentwicklung konnten wir viele Konzepte (z.B. UI-Aufbau, Komponentenstruktur, Routing) gut übertragen.

Auch die Arbeit mit Firebase war für uns neu. Wir waren positiv überrascht, wie einfach sich Firebase Authentication, Firestore und Analytics integrieren liessen. Besonders hilfreich war die gute Dokumentation und die reibungslose Verbindung mit Flutter. Trotzdem sind wir uns nicht sicher, ob wir Firebase in einer echten Produktionsumgebung einsetzen würden, da wir dort möglicherweise mehr Kontrolle über die Datenhaltung und Server benötigen würden.

Riverpod haben wir ebenfalls zum ersten Mal verwendet. Anfangs war das Konzept mit den verschiedenen Providern, AsyncValue und der automatischen Code-Generierung nicht ganz einfach zu verstehen. Mit zunehmender Erfahrung hat uns Riverpod aber viele Dinge erleichtert. Ein Beispiel war die globale Steuerung von Authentifizierungs- und Ladezuständen über AuthController und BookingsController, bei der wir klar zwischen UI und Logik trennen und z. B. Fehler direkt über AsyncError behandeln konnten, ohne eigene State-Klassen schreiben zu müssen.

Insgesamt war es ein lehrreiches Projekt, das uns viele neue Technologien nähergebracht hat und uns gezeigt hat, wie man moderne Mobile Apps strukturiert und entwickelt.

Abbildungsverzeichnis

Abbildung 1: Logo.....	4
Abbildung 2: Anwendungsfalldiagramm.....	8
Abbildung 3: Wireframes - Login.....	10
Abbildung 4: Wireframes - Dashboard.....	11
Abbildung 5: Wireframes - Formular	12
Abbildung 6: Wireframes - Liste	13
Abbildung 7 Beispiel Firebase Auth	20
Abbildung 8 Firebase Authentication	20
Abbildung 9 Firebase Firestore	21
Abbildung 10 Firebase Analytics.....	22
Abbildung 11 Flutter Architektur	23
Abbildung 12 Beispiel Riverpod	24
Abbildung 13 Beispiel Errorhandling	24

Quellenverzeichnis

- Flutter App Architecture – Riverpod, MVC and clean code. <https://codewithandrea.com/articles/flutter-app-architecture-riverpod-introduction/>
Abgerufen am 15. Juni 2025.
- Firebase Documentation. <https://firebase.google.com/docs>
Abgerufen am 15. Juni 2025.
- FlutterFire. Using Firebase with Flutter. <https://firebase.flutter.dev>
Abgerufen am 15. Juni 2025.
- State management in Flutter. <https://docs.flutter.dev/data-and-backend/state-mgmt>
Abgerufen am 15. Juni 2025.
- Riverpod Documentation. <https://riverpod.dev>
Abgerufen am 15. Juni 2025.
- shadcn_ui. <https://github.com/nank1ro/flutter-shadcn-ui>
Abgerufen am 15. Juni 2025.

Hilfsmittelverzeichnis

Hilfsmittel	Einsatzgebiet	Betroffene Stellen
ChatGPT	Formulierungshilfe	Gesamtes Dokument
ClaudeAI	Formulierungshilfe	Gesamtes Dokument