Hochschule Konstanz
Department of Computer Science

Submitted by:
Lars Bürger
Marcel Biselli
Simon Rauch
info@wallat.com

Mobile Anwendungen - SS 23

Projektdokumentation

Konstanz, 2023-07-12

Contents

List of Figures

1 Einleitung

Wir, das Entwicklerteam bestehend aus Marcel Biselli, Lars Bürger und Simon Rauch, haben im Rahmen des AIN Kurses "Mobile Anwendungen" im Sommersemester 2023 eine mobile App namens "Wall@" entwickelt. Unser Ziel war es, eine innovative Lösung für das Scannen und Verwalten von Dokumenten zu schaffen.

Mit Wall@ bieten wir eine benutzerfreundliche und effiziente Möglichkeit, Dokumente digital zu erfassen und in einem sicheren digitalen Geldbeutel zu verwalten. Die App richtet sich an alle, die eine bequeme Methode suchen, um Dokumente wie Quittungen, Rechnungen, Ausweise oder andere wichtige Unterlagen zu scannen, zu speichern und jederzeit griffbereit zu haben.

Bei der Entwicklung von Wall@ haben wir bewährte Software-Engineering-Praktiken und -Standards berücksichtigt, um eine stabile, skalierbare und gut strukturierte App zu gewährleisten. Durch den Einsatz der Programmiersprache Dart und des Flutter-Frameworks ist es uns gelungen, eine plattformübergreifende Lösung zu entwickeln, die gleichermaßen auf den Betriebssystemen iOS und Android funktioniert.

Diese Dokumentation dient als verbindliche Ressource für Entwickler, die an der App weiterarbeiten möchten, und bietet Einblicke in die Implementierungsdetails sowie die zugrunde liegenden Technologien.

Bei weiteren Fragen, Feedback oder Anregungen stehen wir Ihnen gerne zur Verfügung. Wir hoffen, dass diese Dokumentation einen wertvollen Beitrag zur effizienten Nutzung und Weiterentwicklung von Wall@ leistet.

Figure 1.1: Wall@ Logo

2 Entwicklungsprozess

Dieses Kapitel bietet einen Überblick über den Entwicklungsprozess von Wall@. Es behandelt die geplanten User Stories und den Einsatz von GitLab für das Projektmanagement und die Zusammenarbeit im Team. Die User Stories dienen als Leitfaden für die Entwicklung und beschreiben die Funktionalität von Wall@ aus Benutzersicht. Der Entwicklungsprozess umfasst verschiedene Phasen und Praktiken, die sicherstellen, dass die App erfolgreich entwickelt, getestet und bereitgestellt werden kann. Zusätzlich wird der Einsatz von GitLab vorgestellt, einer kollaborativen Entwicklungsplattform, die Funktionen zur Versionskontrolle und zum Projektmanagement bietet.

2.1 Issues in GitLab

Für die Koordination und Verwaltung des Entwicklungsprozesses haben wir das Ticket-System in GitLab genutzt. Mithilfe dieses Systems konnten wir Tickets erstellen, Aufgaben zuweisen und die Entwicklung effektiv koordinieren. Wir haben Funktionen wie Ticketzuweisung (an Verantwortliche), Labeling und Kommentarfunktionen genutzt, um die Zusammenarbeit und den Fortschritt innerhalb des Teams zu erleichtern.

2.2 User Stories

Aufgrund der Anforderungen an die App haben wir verschiedene User Stories erstellt, die die Funktionalität von Wall@ aus Benutzersicht beschreiben. Eine Liste der User Stories spowie der aktuelle Entwicklungsstand der einzelnen Stories ist über ein seperates GitLab Board (siehe User Stories Board) einzusehen.

2.3 GitFlow

Für eine effiziente Verwaltung des Entwicklungsprozesses haben wir das GitFlow-Modell implementiert. Es ermöglichte uns, den Code in verschiedenen Branches zu organisieren und stabile sowie entwicklungsorientierte Umgebungen aufrechtzuerhalten. Wir haben den ”master”-Branch für stabile Versionen, den ”develop”-Branch für die Hauptentwicklung und Feature-Branches für neue Funktionen verwendet. Zusätzlich nutzten wir Hotfix-Branches, um kritische Fehler schnell zu beheben und in den ”master” und ”develop”-Branch zu überführen. Zudem wurden alle Commits mit einer entsprechenden Issue-Nummer versehen, um die Nachvollziehbarkeit zu gewährleisten.

Durch die Anwendung des GitFlow-Modells können wir unsere Arbeit besser organisieren, die Codequalität verbessern und Konflikte zwischen den unterschiedlichen Entwicklungsstadien vermeiden. Es ermöglicht uns auch, verschiedene Entwicklungsstränge gleichzeitig zu verfolgen und die Produktionsstabilität aufrechtzuerhalten.

3 Architektur

3.1 MVC+S

3.2 Anwendung in allen View-Komponenten

3.2.1 Dependency-Inversion

3.2.2 Persistence als Service; DAO pattern

Um den Zugriff auf die Datenbank zu vereinfachen wurde ein Persistenz-Service eingerichtet, sodass der Zugriff transparent geschieht. Dieser ist nach dem MVC+S implementiert Siehe Bild 3.1 für eine informelle Graphik der verwendeten Architektur. Die Persistenz-Service ist modular, sodass Komponenten einfach ausgetauscht werden können. Dies erreichen wir mithilfe des DAO-Pattern, um den tatsächlichen Zugriff auf den Speicher zu maskieren. Im Sinne des Dependency-Inversion-Prinzips definiert der `PersistenceService` die DAOs, die dieser benötigt. Für mehr Information zum Aufbau der Persistenz-Schicht, siehe Kapitel **??**.

Zusätzlich wird ein `DbController` definiert, welcher dafür zuständig ist, die Datenbank zu laden und die DAOs zu instanziieren. Dessen Implementation - und damit die der DAOs - wird über Riverpod injected. Controller erhalten eine Instanz des `PersistenceService` über einen Provider, welcher wiederum einen Provider für den Controller aufruft. Letzterer bestimmt die tatsächliche Implementation.

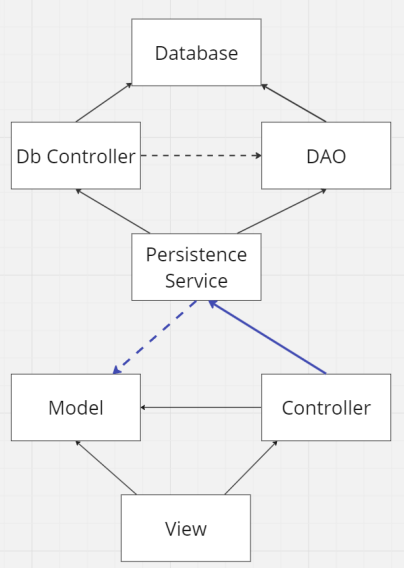

Figure 3.1: MVC+S Architektur

3.3 Compositum Pattern für FolderItems

3.4 State Management (Riverpod)

3.5 Multi-Language Support

3.6 ErrorHandling

3.6.1 Custom FutureBuilders + null safety

4 Persistenz

4.1 Isar

Daten werden mithilfe von Isar persistiert. In Bild 4.1 ist ein ERM-Diagramm der Datenbank abgebildet. Auch wenn innerhalb der Zugriffe auf Isar mit speziellen Datenklassen gearbeitet wird, wandeln die DAOs diese beim Zugriff in die Model-Klassen der MVC-Schicht um. So werden zum Beispiel die Events direkt in die enthaltene Liste der `SingleItem`-Klasse eingefügt und nicht seperat angefragt.

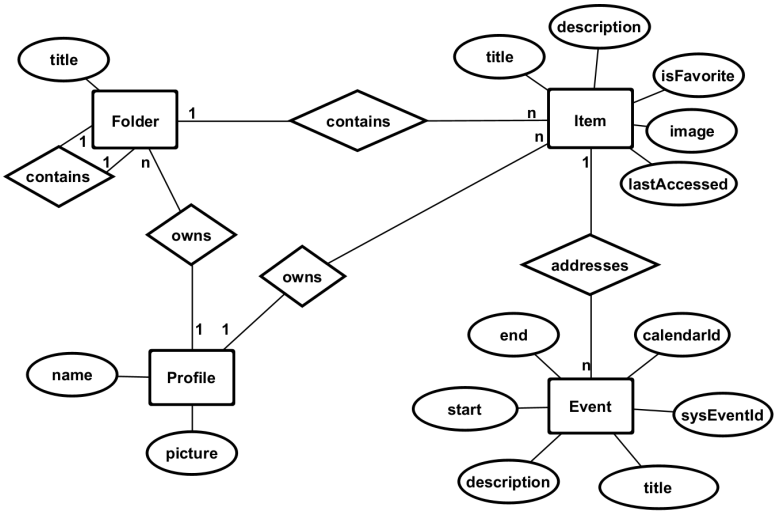

Figure 4.1: Persistenz ERM Diagram

Eine technische Besonderheit sind die Profile. Profile erlauben dem Nutzer, Daten noch einmal grober zu kategorisieren als Ordner. Um dies umzusetzen, verwaltet Isar ebenfalls eine Tabelle für Profile. Jeder andere Eintrag der Datenbank wird dann einem Profil zugeordnet. Die Besonderheit besteht darin, dass beim erstmaligen Starten der App ein Standard-Profil angelegt wird, welches zudem als eine globale Sicht fungiert. Das bedeutet, wenn ein Nutzer weitere Profile anlegt und mit Daten füllt, sind diese Daten auch über das Standard-Profil sichtbar. Zudem funktioniert die App auch ohne weitere Profile und es ändert sich nichts an der Nutzung, wenn ein Nutzer keine Profile erstellen möchte.

Funktionalitäten zum Verschieben von Inhalten zwischen Profilen aber auch zwischen Ordnern sind im Backend bereits implementiert, allerdings sind diese noch nicht über die View zugreifbar.

4.2 Shared Preferences für Settings

4.3 Bilder lose im Application Documents directory

5 Schnittstellen

5.1 Kamera via Package

Bei der Entwicklung von Wall@ war es erforderlich, ein Kamera-Paket in die App zu integrieren, um die Funktionalität des Dokumentenscanners zu ermöglichen. Nach einer gründlichen Evaluierung wurden die folgenden drei verschiedene Kamera-Pakete untersucht, um die Anforderungen unserer App zu erfüllen.

Package	Feature	Problem
flutter_document_scanner	Auto Crop, Manuel Crop, ScannLook	compiling errors
document_scanner_flutter	Auto Crop, Manuel Crop, ScannLook, pdf conv.	depends on photo_view package 0.12.0
cunning_document_scanner	Auto Crop, Manuel Crop	-

Ursprünglich hatten wir uns für das zweite Kamera-Paket entschieden, das unsere Anforderungen optimal erfüllte. Jedoch ist dieses Paket nicht mit Dart 3 kompatibel, was zu Kompatibilitätsproblemen und potenziellen Einschränkungen bei der Weiterentwicklung führen könnte.

Um dieses Problem zu lösen, wurde ein Issue auf dem Entwickler-Git des bevorzugten Kamera-Pakets erstellt (siehe Issue on GitHub) und das Problem kommuniziert. In der Zwischenzeit haben wir uns für ein alternatives Kamera-Paket entschieden, das mit Dart 3 kompatibel ist und unsere grundlegenden Anforderungen erfüllt. Durch die Nutzung des letzten Packet wurde die Funktionalität des Dokumentenscanners in Wall@ implementiert. Es ist somit möglich Bilder mittels der Kamera von Dokumenten aufzunehmen wobei die Ränder automatisch erkannt werden und das Bild entsprechend zugeschnitten wird.

Langfristig planen wir jedoch, eine eigene Lösung für den Dokumentenscanner in Wall@ zu entwickeln. Dies ermöglicht uns eine maßgeschneiderte Implementierung, die besser auf die spezifischen Anforderungen und zukünftigen Erweiterungen unserer App abgestimmt ist. Die Entwicklung einer eigenen Lösung wird es uns auch ermöglichen, volle Kontrolle über die Funktionalität und die Integration in den Rest der App zu haben.

In diesem Zuge soll auch die Funktionalität des Dokumentenscanners erweitert werden, so dass Bilder nicht nur als Bild, sondern auch als PDF gespeichert werden können und diese mittels Filter bearbeitet werden können so dass z.B.: Probleme mit der Belichtung oder dem Kontrast behoben werden können.

6 Generelles (Flutter)

6.1 Cupertino-Widgets

Unser Flutter-Projekt verwendet Cupertino-Widgets und setzt adaptive Design-Techniken ein, um eine Benutzeroberfläche zu erstellen, die den Design-Mustern von Apple-Anwendungen ähnelt.

Cupertino-Widgets sind UI-Komponenten, die vom Flutter-Framework bereitgestellt werden und das Aussehen und Verhalten von iOS-Anwendungen in der Gegenwart widerspiegeln. Diese Widgets haben das ikonische Design von iOS, das sauber und minimalistisch ist, mit abgerundeten Ecken und subtilen Animationen. Indem wir Cupertino-Widgets verwenden, stellen wir sicher, dass unsere App eine konsistente und vertraute Benutzererfahrung für iOS-Benutzer bietet und das Design an das Apple-Ökosystem anpasst.

6.2 Adaptives Design mit MediaQuery und Slivers

Um die App an verschiedene Bildschirmgrößen und -ausrichtungen anzupassen, verwenden wir die MediaQuery-Klasse von Flutter. MediaQuery ermöglicht es uns, Informationen über die aktuellen Bildschirmdimensionen, Pixeldichte und andere relevante Metriken des Geräts abzurufen. Durch die Nutzung dieser Informationen passen wir das Layout und das Verhalten der UI-Komponenten unserer App dynamisch an und stellen so eine optimale Benutzererfahrung auf verschiedenen Geräten sicher.

Mit MediaQuery können wir beispielsweise die verfügbare Bildschirmgröße ermitteln und das Layout entsprechend anpassen. Wir können die Schriftgröße, den Abstand oder sogar die gesamte Struktur der Benutzeroberfläche basierend auf der Bildschirmgröße und -ausrichtung des Geräts anpassen. Durch die Anpassung der Benutzeroberfläche an verschiedene Geräte stellen wir sicher, dass unsere App auf einer Vielzahl von Bildschirmen visuell ansprechend und funktional bleibt.

Zusätzlich zur MediaQuery nutzen wir Slivers, eine weitere leistungsstarke Funktion in Flutter, um adaptives Design zu erreichen. Slivers ermöglichen es uns, flexible und scrollbare Layouts zu erstellen, die sich automatisch an den verfügbaren Bildschirmplatz anpassen. Durch die Verwendung verschiedener Sliver-Widgets wie `SliverAppBar`, `SliverList` und `SliverGrid` können wir den Inhalt und die Struktur unserer App dynamisch anpassen, wenn der Benutzer scrollt oder sich die Ausrichtung des Geräts ändert.

6.3 Animationen

In unserem Projekt nutzen wir Animationen, um eine lebendige und ansprechende Benutzererfahrung zu schaffen. Hier sind zwei wichtige Anwendungsfälle, in denen Animationen zum Einsatz kommen:

6.3.1 Hero-Animation von Bildcontainern

Wir nutzen die "Hero-Animation", um flüssige Übergänge zwischen Bildcontainern zu ermöglichen. Wenn ein Benutzer von der Ordneransicht in einen Einzeleintrag navigiert

oder in der Einzelansicht das Bild im Vollbildmodus betrachtet, wird das Bild nahtlos von einem Container zum anderen animiert. Diese Animation sorgt für einen harmonischen und reibungslosen Übergang und verbessert die visuelle Kontinuität in unserer App.

6.3.2 Animation des Kamera-Buttons

Um den Kamera-Button in unserer Home-Ansicht ansprechend zu positionieren, haben wir eine Animation implementiert, bei der der Button über der Navigationsleiste schwebt. Dadurch wird der Button als zentrales Bedienelement hervorgehoben und ermöglicht dem Benutzer einen schnellen Zugriff auf die Kamerafunktion.

Bei einem Wechsel zu einer anderen Seite unserer App haben wir eine weitere Animation integriert. Der Kamera-Button bewegt sich sanft in die Navigationsleiste hinein und passt sich dem neuen Kontext an. Durch diese Animationstechnik schaffen wir einen nahtlosen Übergang und eine intuitive Benutzererfahrung.

6.4 Widgets der Woche

Im Rahmen unseres Projekts nutzen wir eine Reihe von Flutter-Widgets, die als "Widgets der Woche" bekannt sind. Diese leistungsstarken Widgets helfen uns dabei, bestimmte Funktionen und Interaktionen in unserer App umzusetzen. Hier sind einige der Schlüsselwidgets, die wir eingesetzen:

1. **CupertinoNavigationBar:** Wir haben das `CupertinoNavigationBar`-Widget verwendet, um eine elegante und intuitive Navigation innerhalb unserer App zu ermöglichen. Dieses Widget stellt eine Navigationsleiste im iOS-Stil dar, die es Benutzern ermöglicht, zwischen verschiedenen Ansichten und Bildschirmen unserer App zu wechseln.

2. **FutureBuilder:** Um Daten aus einer Datenbank abzurufen und anzuzeigen, nutzen wir das `FutureBuilder`-Widget. Dieses Widget ermöglichte es uns, asynchrone Operationen auszuführen und das Ergebnis in Echtzeit in unserer Benutzeroberfläche zu aktualisieren. Dadurch können wir eine reibungslose und reaktive Benutzererfahrung gewährleisten.

3. **GridView:** Um eine übersichtliche Darstellung von Ordnern in unserer App zu ermöglichen, habnutzenen wir das `GridView`-Widget. Mit diesem Widget konnten wir die Ordneransicht in ein Rasterlayout umwandeln, in dem Benutzer auf einfache Weise durch die verschiedenen Ordner navigieren und sie auswählen zu können.

SafeArea	LinearGradient	StatefulBuilder
GestureDetector	Hero	HeroMode
SliverAppBar	CupertinoActivityIndicator	Divider
CupertinoActionSheet	DraggableScrollableSheet	CupertinoAlertDialog
Stack	AnimatedOpacity	MediaQuery
Flexible	Dismissible	SizedBox
LayoutBuilder	SliverList	SliverGrid

6.5 Themes

Unser Projekt implementiert ein flexibles Theme-System, das es uns ermöglicht, dass Design unserer App einfach anzupassen oder neue Themes hinzuzufügen. Die Hauptfarben für sowohl das Dark- als auch das Light-Theme sind bewusst so gestaltet, dass sie den Apple-Farben ähneln jedoch unserem Color-CI entsprechen. Dadurch schaffen wir eine konsistente visuelle Ästhetik und sorgen dafür, dass unsere App in das Apple-Ökosystem passt.

6.6 Spezifische Packages

Wir nutzen verschiedene Packages in unserem Projekt, um spezifische Funktionalitäten zu implementieren. Dabei haben folgende Packages eine wichtige Rolle gespielt:

1. **social_share Package:** Wir nutzen das social_share Package, um den Benutzern die Möglichkeit zu geben, Einträge aus unserer App einfach und schnell über verschiedene soziale Medienplattformen zu teilen. Durch die Integration dieses Packages können wir eine nahtlose Teilen-Funktionalität implementieren und es unseren Benutzern ermöglichen, ihre Erfahrungen und Inhalte mit anderen zu teilen.

2. **device_calendar Package:** Das device_calendar Package spielte eine wichtige Rolle bei der Integration von Kalenderfunktionen in unsere App. Mit diesem Package können wir Ereignisse zu den Systemkalendern der Benutzer hinzufügen und entfernen. Dadurch konnten wir eine nahtlose Integration mit den Kalendern auf den Geräten unserer Benutzer gewährleisten und es ihnen ermöglichen, wichtige Termine und Ereignisse direkt aus unserer App heraus zu verwalten.

3. **isar Package:** Das isar Package wurde verwendet, um eine effiziente Datenbank- und Persistenzverwaltung in unserer App zu ermöglichen. Mit diesem Package können wir Datenbankoperationen wie das Speichern, Abfragen und Aktualisieren von Daten nahtlos und performant durchführen. Die Verwendung von isar ermöglicht es uns, die Daten in unserer App effektiv zu verwalten und die Benutzererfahrung zu verbessern.

4. **beamer Package:** Das beamer Package hat eine wichtige Rolle bei der Implementierung des Routings innerhalb unserer App gespielt. Mit diesem Package können wir eine effiziente Navigation zwischen verschiedenen Bildschirmen und Ansichten realisieren. Beamer ermöglicht es uns, komplexe Routenstrukturen zu verwalten und eine benutzerfreundliche und intuitive Navigation durch unsere App zu gewährleisten.

Durch die gezielte Nutzung dieser Packages können wir spezifische Funktionen implementieren und die Benutzererfahrung unserer App erweitern. Sie ermöglichen es uns, Teilen, persistente Bildspeicherung und Kalenderintegration reibungslos und effizient in unsere App zu integrieren.

6.7 Splash Screen

Neben dem Logo (siehe 1.1) wurde zudem ein ansprechender Splash Screen wurde für unsere App entwickelt, um einen visuell ansprechenden Start zu gewährleisten. Der Splash Screen wird beim Start der App angezeigt und bietet den Benutzern eine sofortige visuelle Identifikation unserer Marke und des Designs.