Student Research Project
Documentation

# Invoice Scanner App

Term: Autumn 2024/2025


**Project Team:** Roger Marty
Tseten Emjee

**Project Advisor:** Martin Seelhofer



Version: 01.00
Date: 2024-12-19




School of Computer Science
OST Eastern Switzerland University of Applied Sciences

# Abstract

In this day and age, invoices are received in a wide variety of formats. Some as physical paper documents and others digitally. This makes it challenging to keep track of all invoices, their status and other relevant information. To combat this issue a solution should be created that allows scanning of physical invoices or uploading digital ones for processing. Important information should be extracted and returned in a structured format providing a tabular overview.

A mobile app for Android was developed for the frontend. The app is based on Kotlin utilizing Jetpack Compose and Hilt. As a REST-API for the frontend to interact with, a Python-based backend has been created using the FastAPI framework. The backend has been containerized and runs on a virtual server behind a Traefik reverse proxy. Various AWS services have been integrated for storage, including S3 and DynamoDB, as well as computation with Lambda and data extraction using Textract, a machine learning service that goes beyond ordinary OCR. The Firebase Cloud Messaging service was used for push notifications.

The produced Invoice Scanner app solves the described problem. Invoices can be scanned for automatic data extraction or manually created. The app provides an organized overview of all invoices and options to edit, mark as paid and view the associated original invoice file. Acting as a centralized place for invoices the app can be used for management and archiving purposes.

# Contents

# Part I

# Management Summary

# Management Summary

## Initial Situation

Nowadays, invoices are often received in various formats and media. It is difficult to keep track of all of those invoices and ensure they can be retrieved quickly if needed. Knowing which invoices have been paid and which ones are still open is important as well. The described problem will be addressed in this student research project.

The aim of this project is to create the Invoice Scanner mobile app. The app should enable users to scan their invoices and view the extracted information in a structured way. At the same time this project will be used to apply known technologies and methods whilst new expertise can be gathered. This documentation also serves as training on how to write academic papers. Appropriate project tools are utilised where needed.

## Methodology

After the task definition was received, a project plan was made. The project adopted the Scrum+ methodology, which combines elements of the Rational Unified Process (RUP) and Scrum. Following this methodology, a rough long-term plan was created, outlining the four phases of RUP: Inception, Elaboration, Construction and Transition. In addition, the milestones of the project were identified and planned out.

The functional and non-functional requirements were defined in accordance with the task at hand. Some functional requirements were optional, while others were mandatory for the minimim viable product (MVP). Up next was the planning of the actual realisation of the application. This involved important steps like defining the software architecture and then creating a working prototype using the selected technologies to confirm compatibility. Low- and high-fidelity designs for the app's interface were also created. This ensured a smooth and mutually agreed-upon implementation.

Then the construction phase began, during which the requirements were implemented. First a beta version was released containing the core features, followed by the release of the MVP, that covers all the mandatory requirements and use cases. The whole project was done through multiple bi-weekly sprints in accordance to the Scrum+ methodology and overseen in weekly meetings with the advisor.

## Technologies

The end-product was constructed with following technologies and frameworks:

### Version Control: Gitlab

The codebase is maintained in the OST instance of Gitlab under https://gitlab.ost.ch/.

### Frontend: Android, Kotlin, Jetpack Compose, Android Studio

The frontend of the application is a native Android app built in Android Studio and written in Kotlin. The UI code is written in a modern declaritive approach using Jetpack Compose.

### Backend: FastAPI, Python, Terraform

The Python backend serves endpoints with help of the FastAPI framework, bridging the connection of the frontend and the external interfaces. Additionally Terraform is employed for a more streamlined and git-versioned management of the cloud services.

### External Interfaces: AWS, Firebase

The backbone of the application lives in the AWS cloud. The user and invoice data is saved in DynamoDB, while S3 is used for the binary invoice files in formats like PDF and PNG. Lamda is also in use. It is triggered by new uploads to S3 and processes the invoices using Textract, a ML-based OCR tool. After completion, Lambda will notify Firebase to send a push notification back to the frontend client.

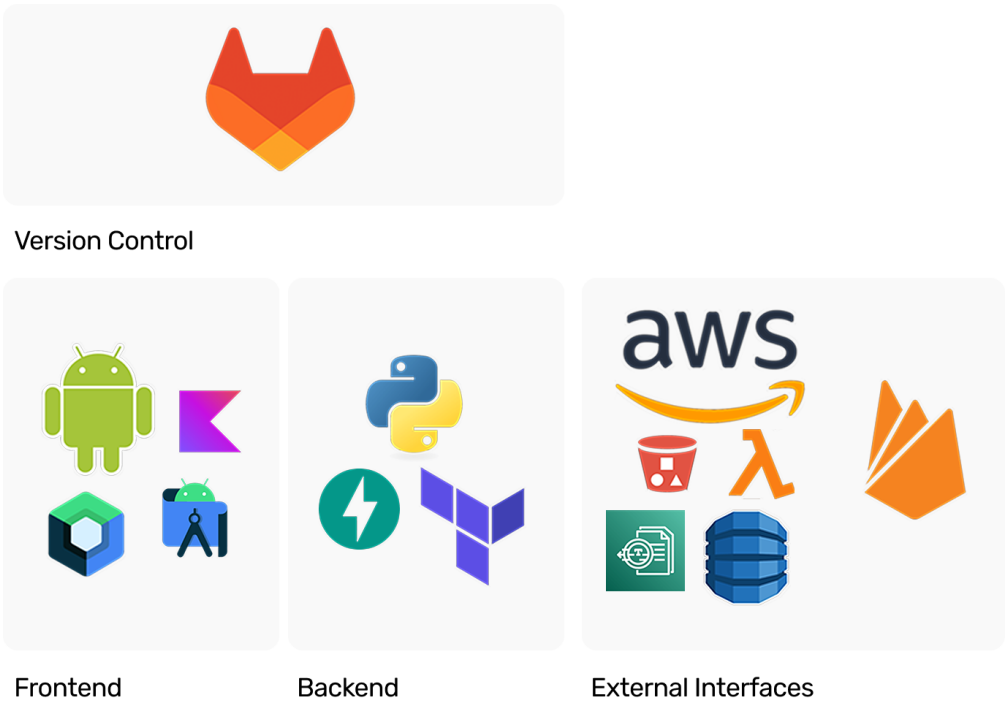


Figure 1: Used technologies

## Result

The final version of the project allows users to take a picture of their physical invoice or upload a digital version. It will then be processed and once the process is finished a push notification is sent to inform the user. A new entry is then created on the overview page where they can quickly access all of their invoices. If information about an invoice is needed or the original file needs to be viewed, the entry can be opened and all information is visible. When an invoice is not available, a manual invoice entry can be created. All entries can be edited and can also be set to `paid`. Entries that are no longer needed can be deleted.

All of the defined requirements for the MVP were implemented and thus this project is considered a success. Only optional requirements remain open. The login feature for example, which was partially implemented but could not be completed in time.

And so the final product still has room for growth. Aside from the predefined optional requirements, there are also ideas gathered from the feedback of the conducted usability tests. Completing the login and authentication feature would be a logical next step and it could be introduced to the currently account-less approach in parallel.

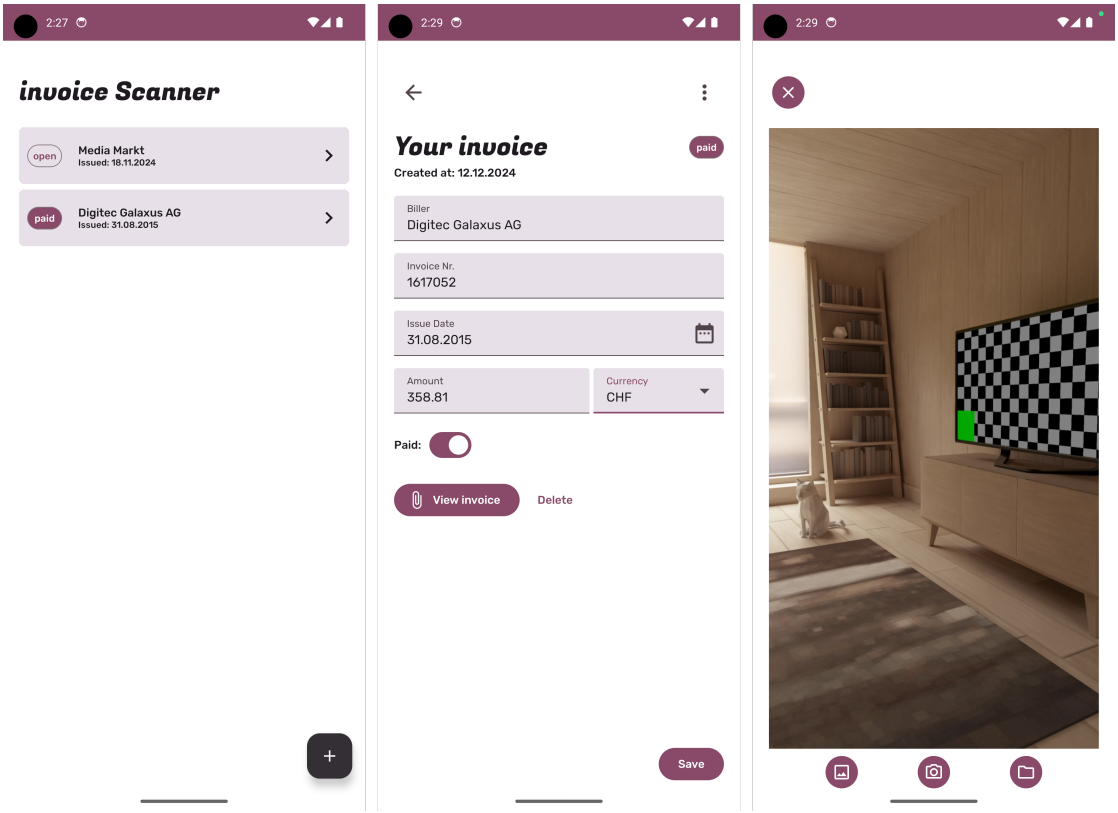


Figure 2: MVP App

# Part II

# Product Documentation

# Chapter 1

# Requirements

Following requirements are based on the task description provided by the advisor. The description can be found in the appendix.

## 1.1 Functional Requirements

Below are all the functional requirements defined for this project. Use cases with a white background are planned for the MVP while the ones with a grey background are categorised as optional use cases.

### 1.1.1 Actors

This project encompasses one actor which is the user of the app. The external systems AWS and Firebase are also displayed in the graphic.

**User**

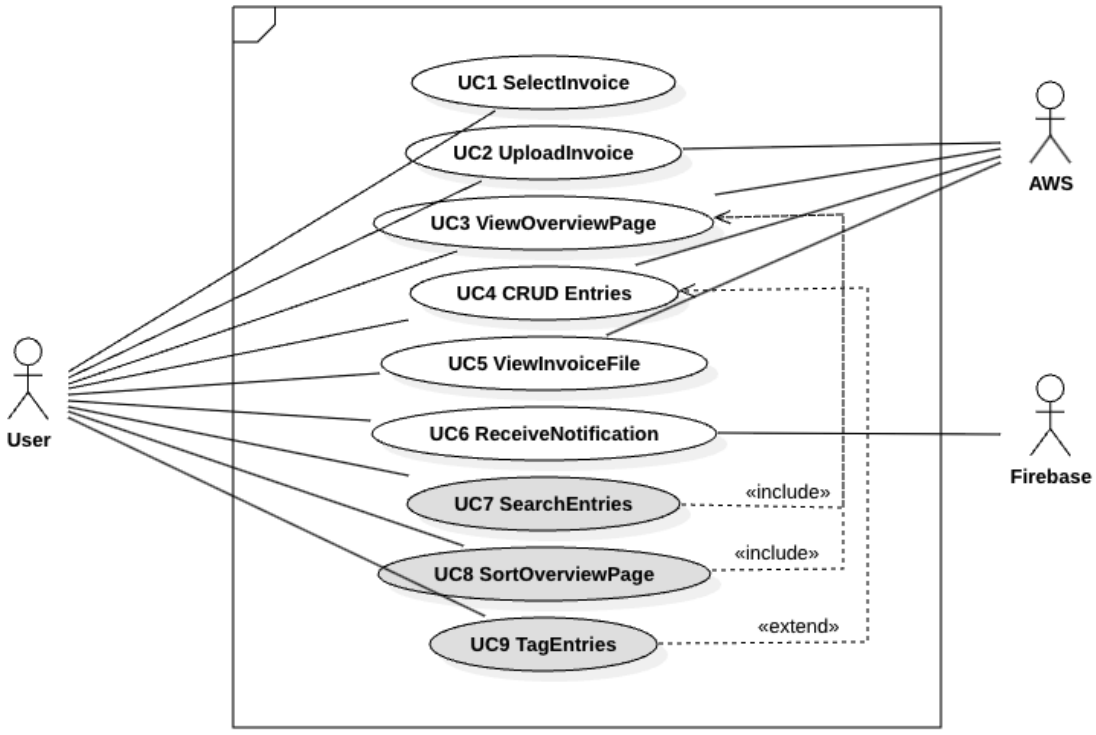


Figure 1.1: Use Case Diagram

### 1.1.2 Use Cases

These are the use cases defined for this project. They are written in the casual format and include alternate scenarios.

| ID | Description | Alternate Scenario |
|---|---|---|
| UC1 | The user can either take a photo of an invoice, choose one from his photo gallery or select a PDF file from the file explorer. | - |
| UC2 | After an invoice has been selected the user can either confirm and the invoice gets uploaded or the user can return and start over. | Connection failure: User gets informed about the failure. |
| UC3 | An overview page with all scanned invoices is available to the user. | No scanned invoices: Hint to add invoice.<br>Connection failure: User gets informed about the failure. |
| UC4 | User can CRUD invoice entries. | Invoice still progressing: User has to wait until it's finished.<br>Connection failure: User gets informed about the failure. |
| UC5 | The files used for scanning can be viewed in the respective invoice entry. | No file provided: Nothing will be shown. |
| UC6 | After an invoice has been processed the user is informed with a push notification. | - |
| UC7 | In the overview page a search function exists to filter the entries. | No search results: A hint is shown to the user. |
| UC8 | The user is able to sort his scanned invoices by various fields. | - |
| UC9 | Each entry can be tagged with a predefined tag which can then be used in the search function. | - |

Table 1.1: Casual Format Use Cases

### Precondition

The above use cases have the precondition of first being authenticated with the system. There are two variants for this precondition, the grey one is optional and not considered for the MVP.

| ID | Description |
|---|---|
| PC1 | The user is authenticated via a device-unique UUID generated on first startup. |
| PC2 | The user registers themselves on first startup and can authenticate themselves with a login and JWT. |

Table 1.2: Precondition Variants

## 1.2 Non-Functional Requirements

The following non-functional requirements are defined for the project, which are based on the ISO/IEC 25010 [1] standard.

### 1.2.1 Performance Efficiency

| | |
|---|---|
| **ID** | NFR1 |
| **Description** | The scanning process is performant. |
| **Acceptance Criteria** | The time from when the user confirms the upload of the invoice until it's completed must be less than specified with a maximum file size of 10MB, given the user has at least a working 4G internet connection. |
| **Landing Zones** | • Minimal: 15 seconds<br>• Target: 10 seconds<br>• Outstanding: 8 seconds |

Table 1.3: NFR1

| | |
|---|---|
| **ID** | NFR2 |
| **Description** | The backend is able to process many requests. |
| **Acceptance Criteria** | The backend completely processes the request to fetch all invoices of a user. |
| **Landing Zones** | • Minimal: 60 requests per minute<br>• Target: 120 requests per minute<br>• Outstanding: 200 requests per minute |

Table 1.4: NFR2

| | |
|---|---|
| **ID** | NFR3 |
| **Description** | The mobile app pages load quickly. |
| **Acceptance Criteria** | The pages load in less than one second and don't impact the user's experience, given the user has at least an android phone with performance comparable to the Samsung A21 (2020) [2]. |

Table 1.5: NFR3

### 1.2.2 Reliability

| | |
|---|---|
| **ID** | NFR4 |
| **Description** | The backend server supports fault tolerance. |
| **Acceptance Criteria** | If a backend node fails on the OST server the operabilty isn't impacted and the traffic gets loadbalanced to a working node. |

Table 1.6: NFR4

| | |
|---|---|
| **ID** | NFR5 |
| **Description** | There is always a backup of the database data available. |
| **Acceptance Criteria** | Backups for all databases are made daily, which can be restored at any point in time. |

Table 1.7: NFR5

| | |
|---|---|
| **ID** | NFR6 |
| **Description** | User input does not cause the app to crash. |
| **Acceptance Criteria** | No crashes in usability and regression testing occur due to user input. |

Table 1.8: NFR6

### 1.2.3 Maintainability

| | |
|---|---|
| **ID** | NFR7 |
| **Description** | The project is effectively and efficiently modifiable without introducing defects. |
| **Acceptance Criteria** | Industry standard code guidelines are enforced with linters, formatters and pipelines to ensure quality. |

Table 1.9: NFR7

### 1.2.4 Security

| | |
|---|---|
| **ID** | NFR8 |
| **Description** | Data is only transferred over secure connections and is encrypted during transit. |
| **Acceptance Criteria** | Only HTTPS connections are used for data transfer between frontend, backend and AWS. |

Table 1.10: NFR8

| | |
|---|---|
| **ID** | NFR9 |
| **Description** | Data at rest is encrypted at all times. |
| **Acceptance Criteria** | The data in DynamoDB and S3 is never saved as plain-text and is encrypted instead. |

Table 1.11: NFR9

| | |
|---|---|
| **ID** | NFR10 |
| **Description** | A user can only see his own data. |
| **Acceptance Criteria** | A user must not be able to access another users data upon opening the app. This is tested by developers on an adhoc basis. |

Table 1.12: NFR10

### 1.2.5 Interaction Capability

| | |
|---|---|
| **ID** | NFR11 |
| **Description** | A user is aware when errors happen. |
| **Acceptance Criteria** | Messages are shown to inform the user about system errors. |

Table 1.13: NFR11

| | |
|---|---|
| **ID** | NFR12 |
| **Description** | The app is easy to use. |
| **Acceptance Criteria** | The feedback from the usability tests is positive. Focus on the accessibilty and general feedback questions. |

Table 1.14: NFR12

# Chapter 2

# Domain Analysis

This chapter describes the results of the domain analysis.

## 2.1 Domain Model

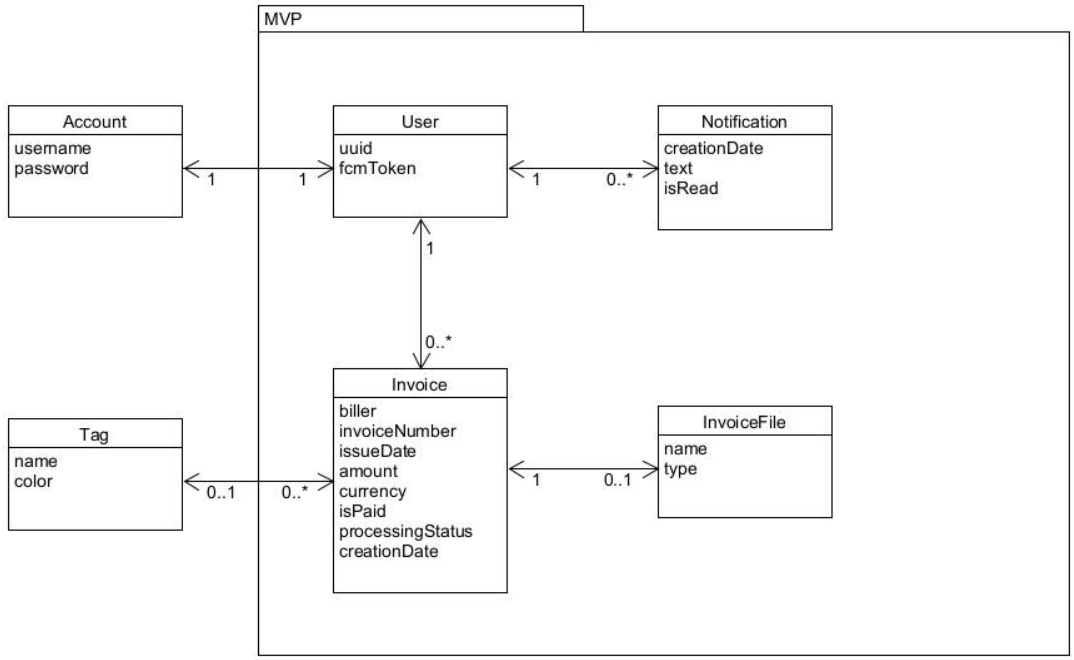


Figure 2.1: Domain Model Diagram

The above graphic shows the domain model diagram, separated into an MVP group and two external entities defined for the optional goals. While most of the diagram is self explanatory, following are descriptions for some special cases.

### Invoice

The `status` field is used to reflect the process status of an invoice and is an enum. This enum has three items namely:

- `Processing`
- `Complete`
- `Failed`
  Failure is defined by any error during the extraction process, a missing field or an insufficient confidence score for the completed extraction.

### Tag

The tag entity reflects the ability to tag invoices. To note here is that tags are exclusive to one another, meaning an invoice can only have none or one tag assigned to it.

# Chapter 3

# Architecture

This chapter presents the various architectural decisions made for the product and their reasoning behind each choice.

## 3.1 Technology Decisions

This section shows the technologies of the project and explains their choices.

### 3.1.1 Frontend

**Android / Kotlin** The frontend of the system is a native android application written in Kotlin. Using the modern Kotlin language instead of Java is officialy recommended when developing a new android app.

**Jetpack Compose** For the UI code, the project uses Jetpack Compose [3]. This declaritive way of writing UI code lessens the overall amount of code and encourages modularity.

The team had a positive experience working with this frontend stack from the previous SE-Project module. This was the main influence for the choices.

### 3.1.2 Backend

**Python / FastAPI** The main part of the backend is written in Python using the FastAPI framework [4]. It is a modern web framework perfectly suited for the needed REST API endpoints, with out-of-the-box `async` support and overall less LOC than the alternative Flask, for example.

**Terraform** Everything AWS related is done via IaC using Terraform [5]. All services and their exact settings can be configured as code, instead of the AWS console. This allows for fast changes and version control.

### 3.1.3 Infrastructure

**Traefik** It is used as a reverse proxy and for its loadbalancing capabilities. [6]

**Watchtower** This is used for monitoring the container registry of the backend repository. [7]

### 3.1.4 External Interfaces

**AWS** Due to some AWS services perfectly fitting the use cases of the app, as well as a general interest in the AWS infrastructure, following services are used and operated via the Python backend:

- **S3:** For uploading and storing the invoice documents AWS S3 is used. [8]
- **Lambda:** This is a serverless compute service, triggered when new invoice document is uploaded to S3. It forwards the document to the extraction service and afterwards saves it in the database. [9]
- **Textract:** Integration of AI for the extraction and processing of relevant data is an integral part of this project. For this purpose Textract is used. This service goes beyond standard OCRs since it is powered ML. It has a specialized Expense API which can be used for invoices and receipts. [10]
- **DynamoDB:** Finally DynamoDB is used for the data storage, holding the invoice and user data. DynamoDB is a serverless NoSQL database. [11]

**Firebase** The whole flow of uploading, extracting and saving invoice data can take time depending of the size of the invoice. In order to inform the frontend when the process has completed, FCM is used to send push notifications to the app. [12]

### 3.1.5 Use of AI/LLMs

In general, the use of LLMs is acknowledged in this project. Considering their efficiency compared to a "classic" search engine for quick problem solving, as well as their widespread use in the industry itself, their omission would be senselessly limiting.

Since these LLMs are of course not always correct, the responses given by them are always screened thorougly by the team, before incorporating them into the codebase.

## 3.2 C4 Model

Following is the C4 model of the product, depicting an overview of the system architecture. Only the Levels 1 through 3 have been modeled, on the basis that the Level 4 code diagram would be too specific and subject to many changes. The diagrams from Level 1-3 are sufficient enough to gain a good understanding of the architecture.

### 3.2.1 Context

The context diagram gives an overview over all the systems involved in the product and how they roughly interact with each other.

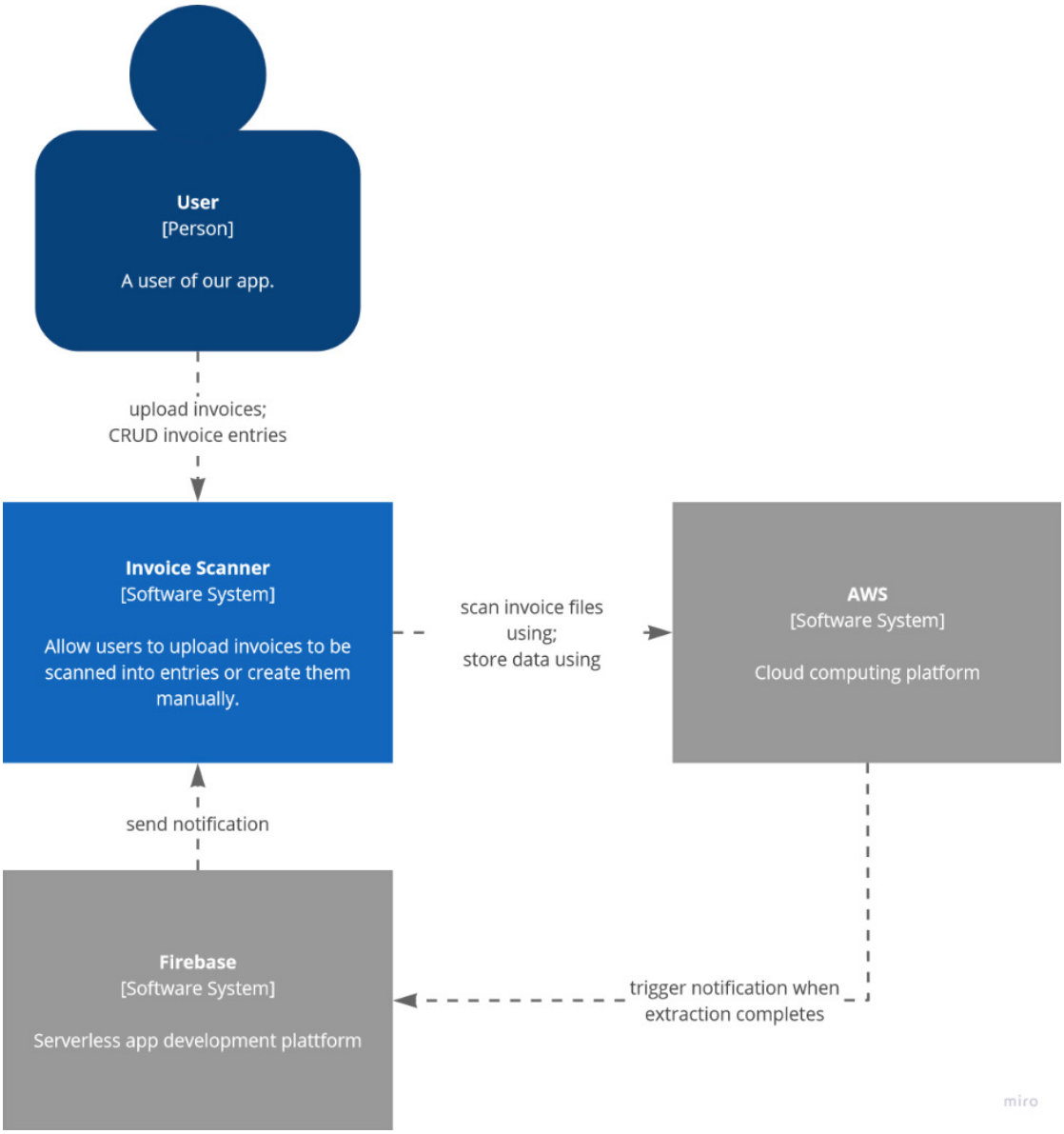


Figure 3.1: C4 Context Diagram

### 3.2.2 Container

The container diagram takes a closer look into the Invoice Scanner system itself, revealing its parts "Mobile App" and "Backend". The external AWS system containers are also

modeled, since the configuration and orchestration of their instances are a big part of the project.

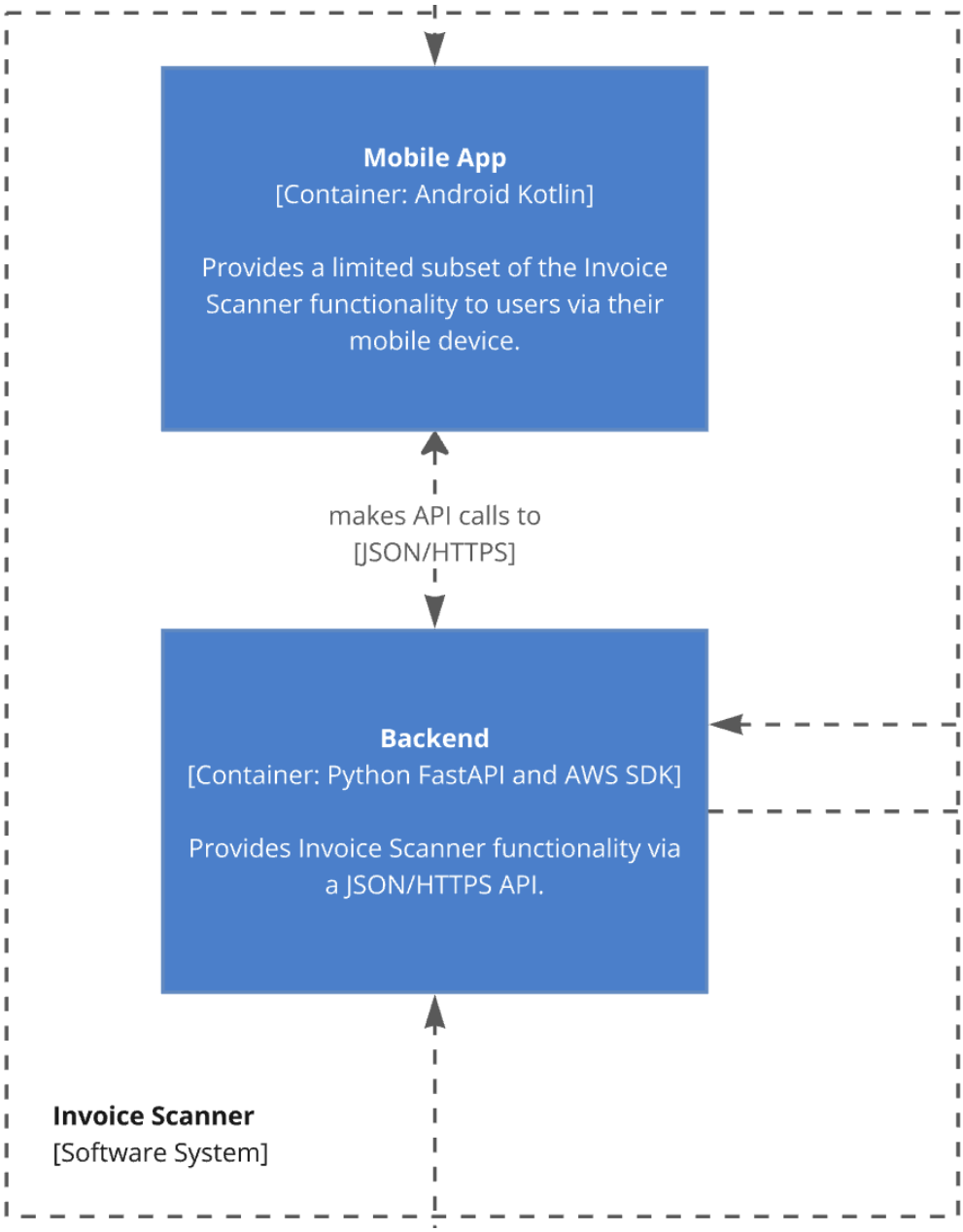


Figure 3.2: C4 Container Diagram - Invoice Scanner

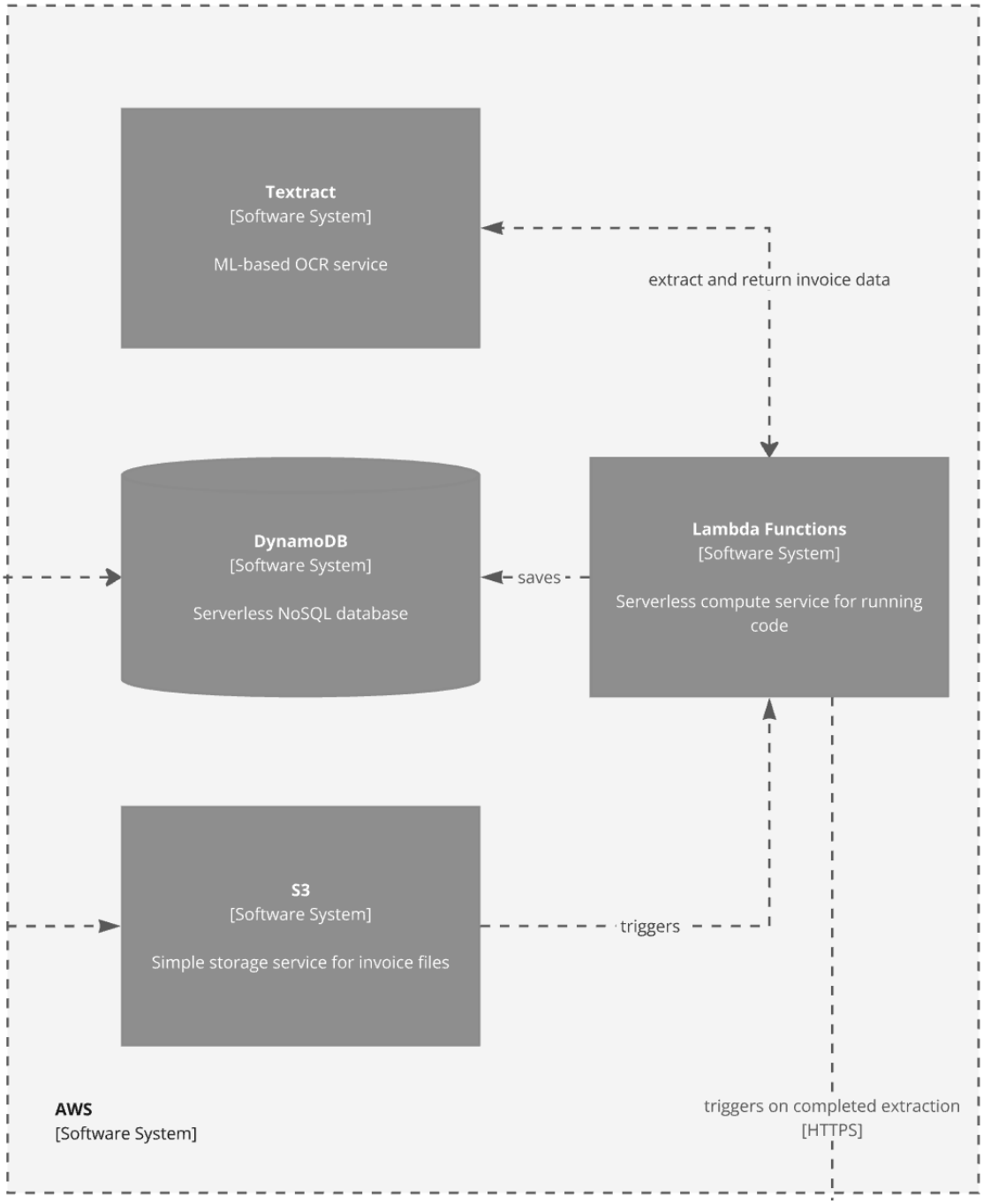


Figure 3.3: C4 Container Diagram - AWS

### 3.2.3 Component

Lastly, zooming into the Invoice Scanner containers "Mobile App" and "Backend" presents the layers they're composed from.

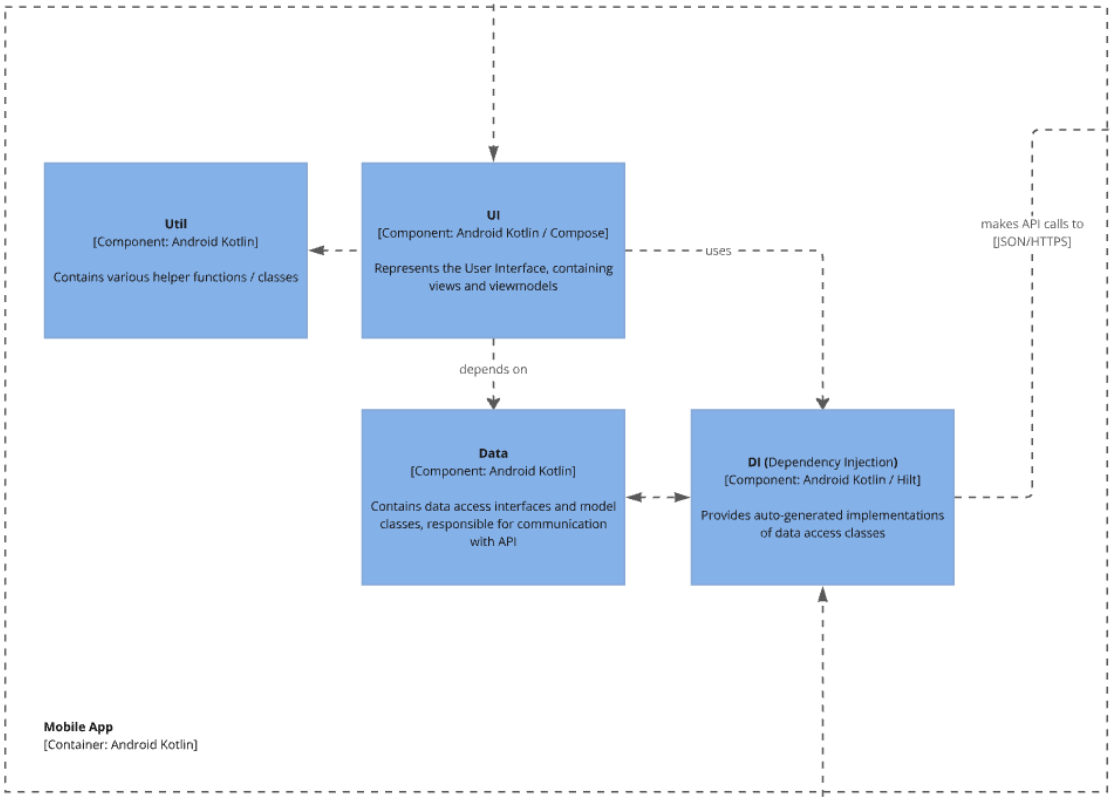


Figure 3.4: C4 Component Diagram - Mobile App

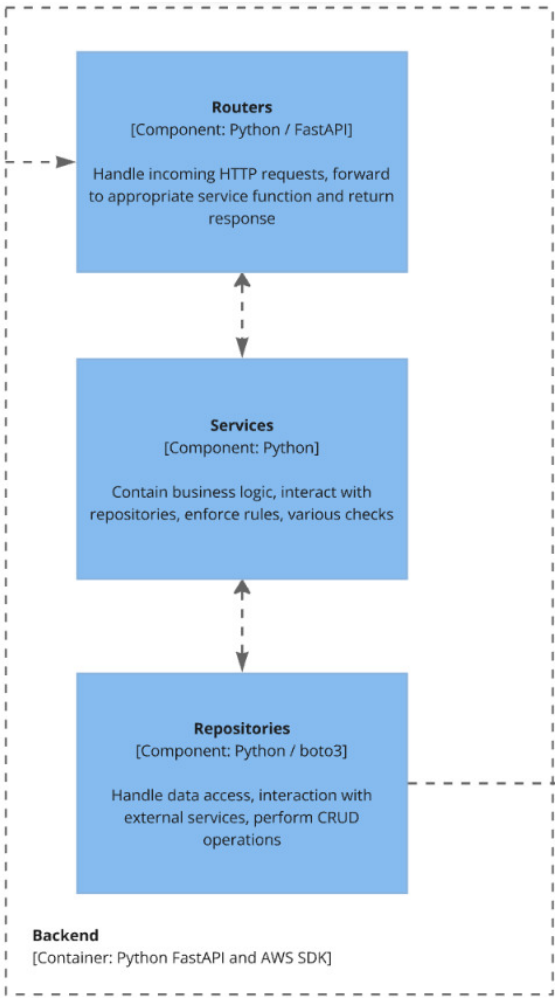


Figure 3.5: C4 Component Diagram - Backend

## 3.3 Component Interaction Diagram

Following process describes how the main workflow of the app, the invoice scanning, works and how the components interact with each other on a high-level basis. If a user confirms to upload a certain invoice, it gets forwarded from the app to the backend. The backend calls the necessary AWS services and returns a response to the app. The invoice gets processed by Lambda and Textract and eventually the result is saved in DynamoDB. A notification is triggered through Firebase and send to the app to give the user feedback.

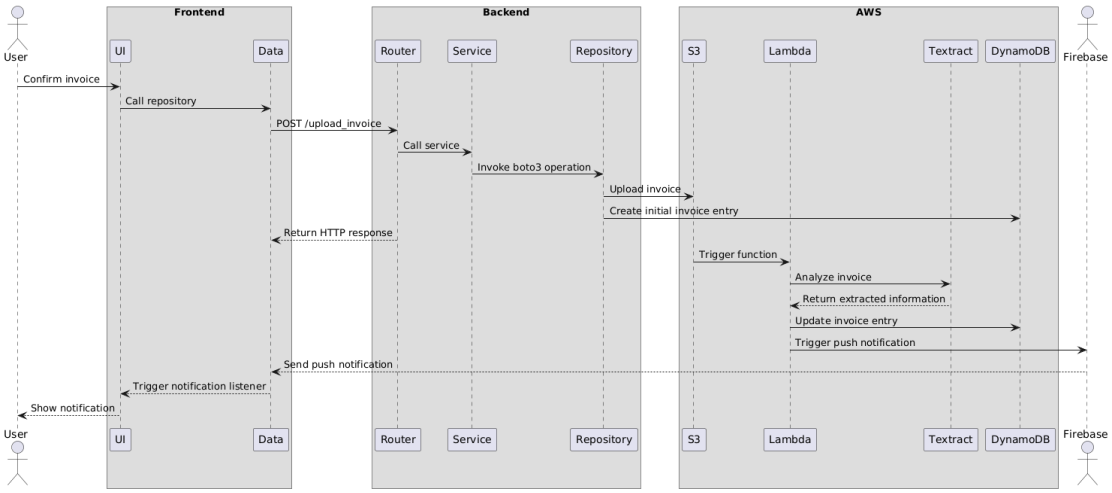


Figure 3.6: Main Workflow CID

## 3.4 Frontend

The frontend app architecture is based on the recommended best practices from Google.

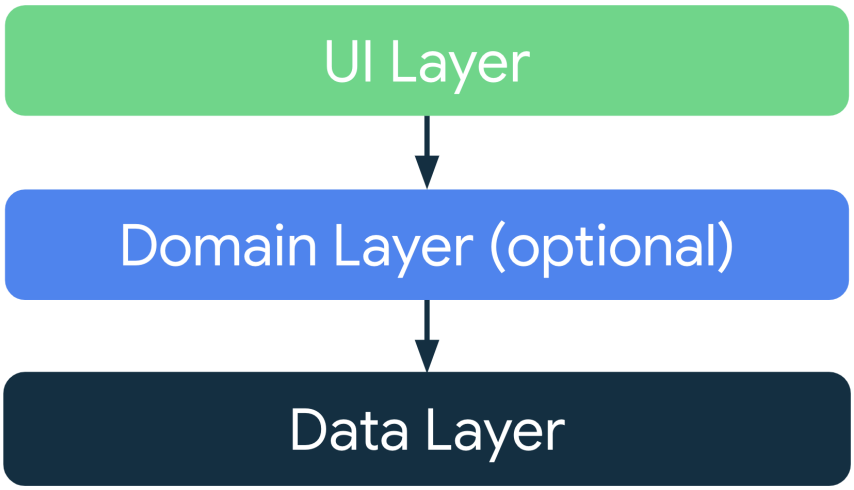


Figure 3.7: Typical App Architecture from Google [13]

The app is split up into UI layer, optional domain layer and lastly the data layer, which handles interfaces and connections to the backend. Furthermore, the data Layer itself is split up into repository and data sources. The repository is the connection point for the higher level layers, tasked with aggregation and handling of the data coming from the data sources.

The dependencies are as pictured. Uni-directional and from the top to bottom. This ensures that changes to the volatile UI layer do not effect the more stable data layer functionalities. With proper use of interfaces and dependency injection using the Dagger/Hilt library [14], the testability of the codebase is very high.

As seen in the C4 Component Diagram, there is also a Util component to extract needless complexity from the UI layer/component.

## 3.5 Backend

The API follows a three-layer architecture approach, where each layer handles different responsibilities for clear separation of concerns. The backend consists of a router layer, services layer and a repository layer. The router layer handles all HTTP requests, while the services are responsible for business logic. The repositories handle the data operations and acts as the link between the backend and AWS.

For enhanced testability and loose coupling, dependency injection is used between the layers.

Furthermore, to create a more maintainable and readable codebase, each layer exists once for each domain. There is a user domain and invoice domain. While they follow the same exact patterns, their focus is on their own respective functionality and models.

## 3.6 CI/CD

Of all the repositories, the frontend and backend ones rely most on the pipelines in place. Below is an overview of how they are structured and the tasks they accomplish. The pipelines run on every commit and depending on the branch, different tasks are executed.

### 3.6.1 Frontend Pipeline

The pipeline consists of five stages that each contains their related tasks. It starts off by linting the whole codebase to scan for any programmatic or stylistic errors. This is also done by pre-commit hooks but an additional check is done on the remote repository. Following that, all unit tests get executed and the test coverage is measured. A report of all unit tests and the test coverage percentage are made available to the developers in merge requests. An additional task is then needed in the report stage to convert it into the correct format. Depending on the branch the commit is made on, either a debug or release APK is built and saved as an artifact. Finally, security checks are carried out.

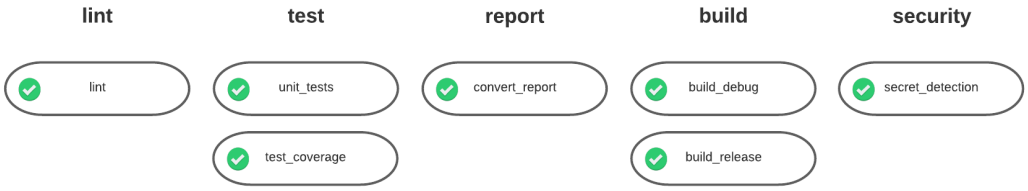


Figure 3.8: Frontend Pipeline

### 3.6.2 Backend Pipeline

In the backend repository, the process it is quite similar. It begins with the linting, followed by the unit tests and the test coverage measurement. Afterwards, the Dockerfile is used to build either a develop or a production docker image, which is then pushed to the repository with the respective tag. In the end, various security checks are made.

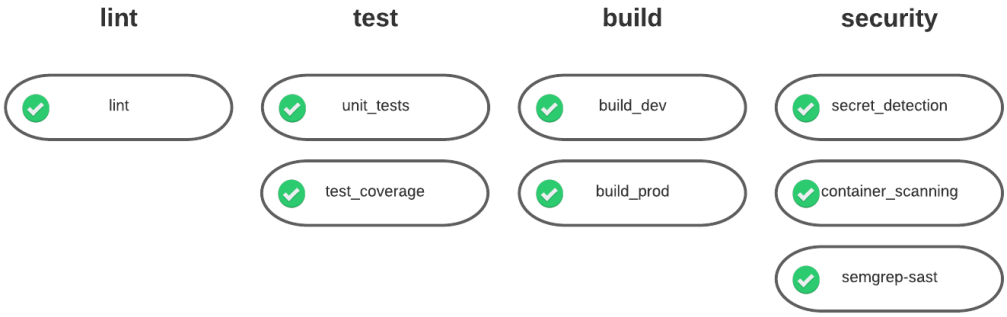


Figure 3.9: Backend Pipeline

## 3.7 Server Infrastructure

To host the backend, a virtual server from OST is in use. The backend is containerized and multiple replicas run on the server. Healthchecks are made by Traefik and all healthy nodes receive traffic.

For continuous deployment a pull method is used. If a newer version is pushed to the registry, the changes are detected and containers with the new version are created while old ones are destroyed.

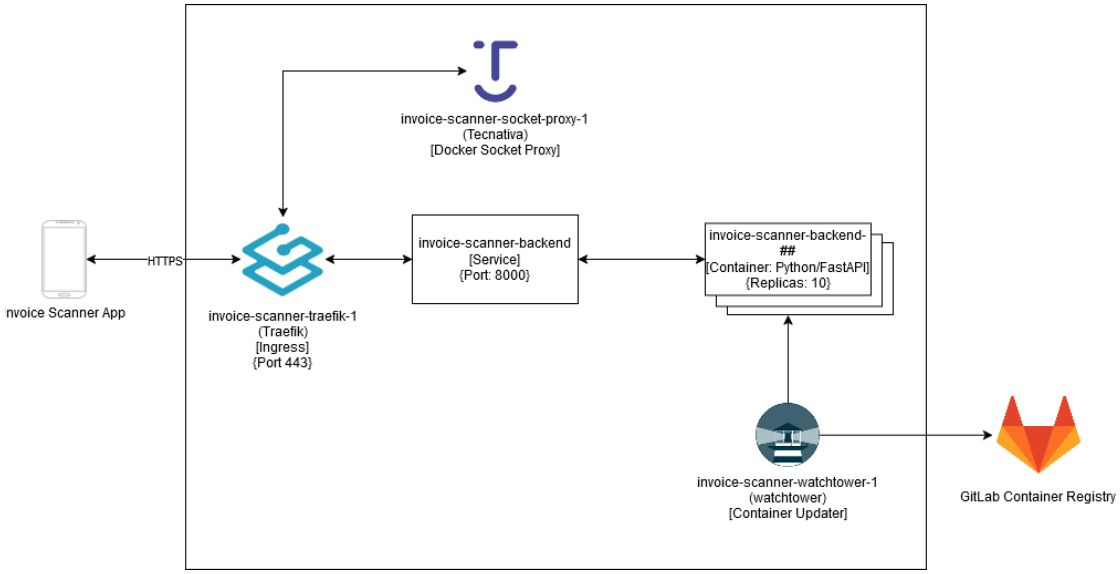


Figure 3.10: Server Environment

## 3.8 Extension

The modular architecture used across all codebases enhances the maintainability and flexibility. New and optional features can be added without affecting existing functionality. The pre-commit hooks and pipelines ensure that the code quality remains high and complies with the defined guidelines.

## 3.9 Scaling

This section discusses how the project can be scaled regarding performance and support for additional platforms.

### 3.9.1 Performance

The backend is the primary component that could be scaled or optimized regarding performance. Other services used in the project are serverless, and scaling is automatically handled by the providers themselves. The server hosting the backend is limited in its resources. There are multiple possibilities to improve this.

**Kubernetes:** Instead of relying on Docker Compose to run the application, Kubernetes could be used. It allows for container orchestration and automatic scaling based on the current load.

**AWS:** Since some AWS services are used in this project, hosting the backend on AWS is a viable option. This could be achieved using ECS [15] or EKS [16].

**Vertical Scaling:** Increasing the allocated resources of the virtual server is another improvement although only temporary.

### 3.9.2 Platforms

This project focuses on creating an app for the Android platform. For future extension, additional platforms could be supported.

**iOS:** Developing an app for the iOS operating system would cover the entire smartphone sector.

**Web:** To support non-smartphone devices, a web app would be necessary. This would enable almost all devices to access the Invoice Scanner's functionalities.

# Chapter 4

# Design

Here in this chapter, the visual and user interface decisions are documented. This includes icon and app designs.

## 4.1 Colors

The color choices are based on the official OST Logo, specifically the purple shades.

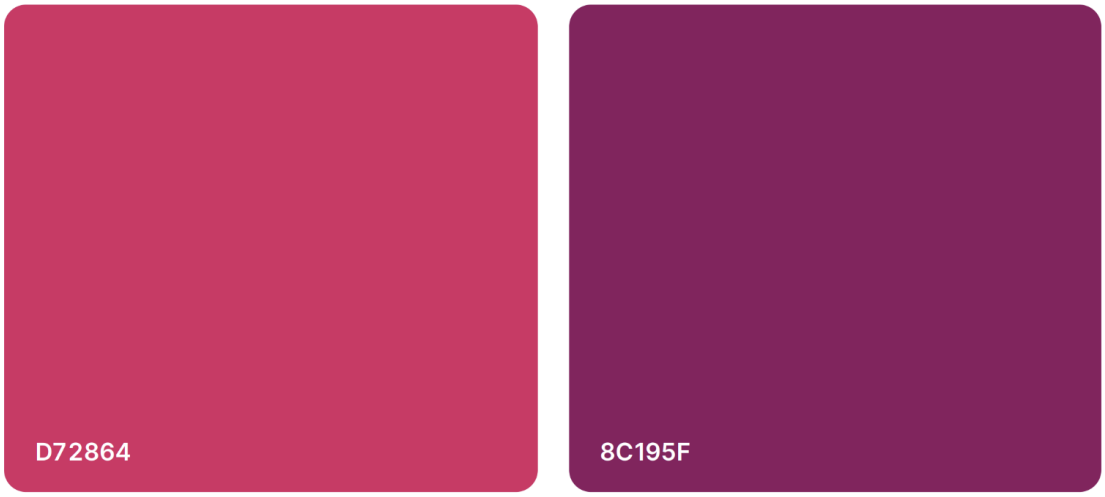


Figure 4.1: Main Colors

## 4.2 Logo

Following logo is used for the app icon and other necessary placements. CHF is used as a relation to currency which aligns with the purpose of the Invoice Scanner app. The colors used derive from the above color scheme.


Figure 4.2: Invoice Scanner App Logo

## 4.3 Prototyping

To save time and avoid rash decisions during the implementation phase, a lot of effort goes into the design prototypes. The prototypes are all made in Figma [17], with assets of their official Material 3 Design Kit [18]. All designs can be found in the appendix.

### 4.3.1 Low-Fidelity

The low-fidelity design of the app showcases all the different interactions and rough screens that occur in the app. Actual UI decisions like colors or fonts are not the focus here.

### 4.3.2 High-Fidelity

The high-fidelity prototype shows how the app should actually look like after implementation. The color scheme is generated using the Material Theme Builder plugin [19] in Figma and then adjusted. While some deviation in the actual implementation is acceptable, it should resemble it closely.

# Chapter 5

# Implementation

This chapter describes the actual implementation of the application, as planned in the architecture chapter.

## 5.1 Frontend

This section documents the implementation of the frontend architecture, as well as other off-note implementation aspects in more detail.

### 5.1.1 App Architecture

As seen below in the project structure of the app, the architecture is implemented according as planned.

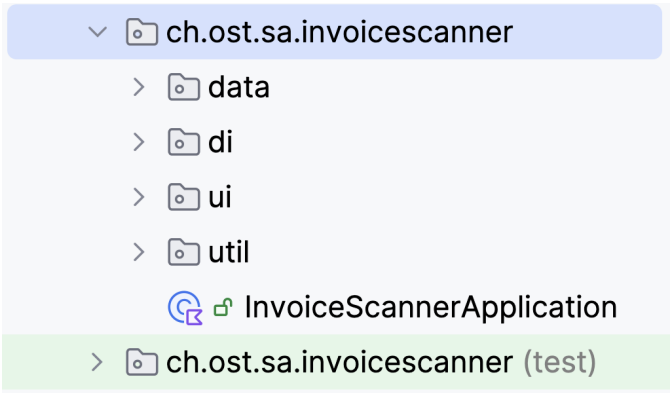


Figure 5.1: App Project Structure

Following is a sequence diagram of the deletion of an entry in the `OverviewScreen`. This showcases the interworkings of the separate packages/layers.

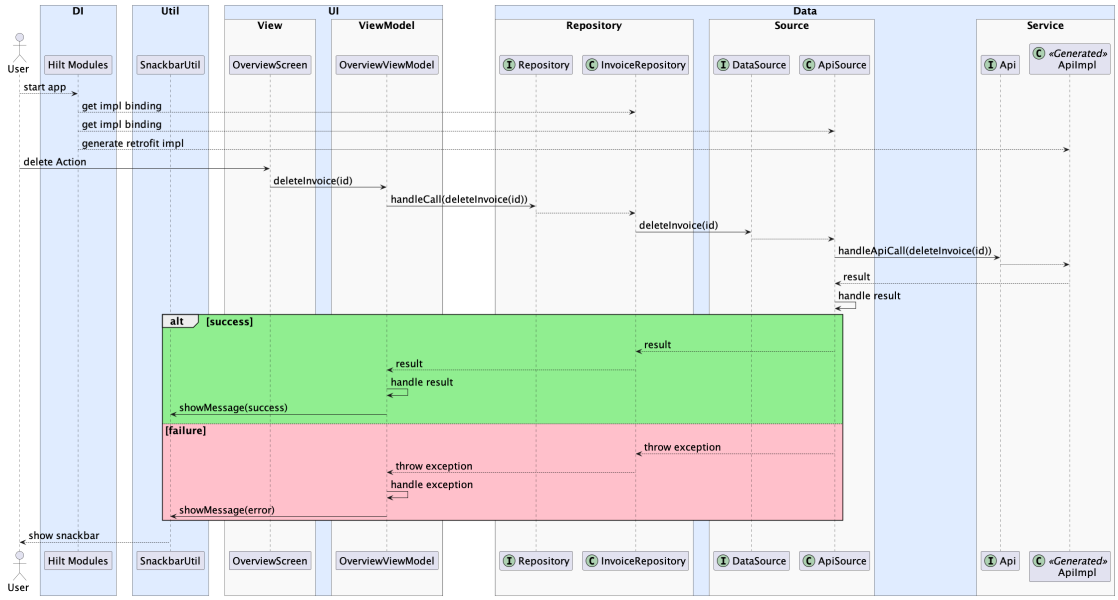


Figure 5.2: Delete Sequence Diagram

The diagram shows that the classes do not depend on the implementation themselves but rather on their respective interfaces. The implementations get binded and/or generated during compilation by the Hilt library. These binding are defined in their own modules in the DI package. Through such use of interfaces, testing becomes much easier because it enables the use of Fakes.

```
1    package ch.ost.sa.invoicescanner
2
3    import ...
22
23   class FakeFactory {
24       companion object {...}
100
101      object FakeSuccessApi : Api {...}
167
168      object FakeErrorApi : Api {...}
232
233      object FakeDataSource : DataSource {...}
298
299      object FakeKeyValueStore : KeyValueStore {...}
317
318      object FakeRepository : Repository {...}
393  }
394
```

Figure 5.3: FakeFactory Class

### 5.1.2 Error and Response Handling

The sequence diagram in figure 5.2 shows two places where a call is "handled". Namely, once in the `OverviewModel` class and again in the `ApiSource` class. This encompasses the error handling of external API calls. In general it looks like this:

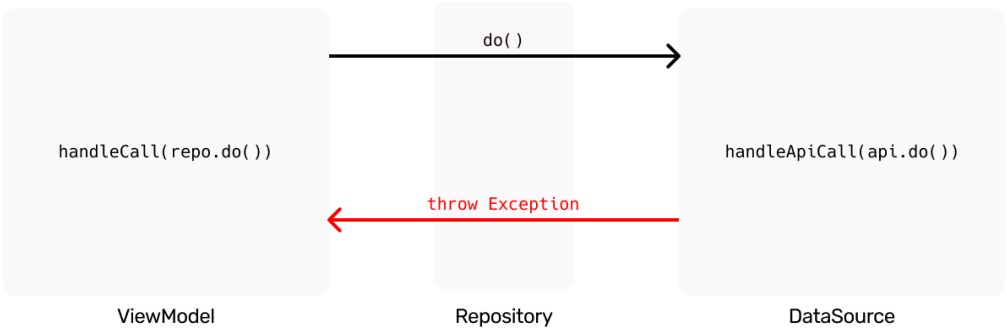


Figure 5.4: API Error Handling

All viewmodels in the app inherit the `BaseViewModel` class, which provides the `handleCall(...)` function. Utilizing the fact that in Kotlin functions are treated as first-class citizens, `handleCall(...)` is written as a higher-order function. It takes a function call to the `Repository` as a parameter and handles any thrown exceptions in a central place.

The function `handleApiCall(...)` is a higher-order function as well. It takes a call to the `Api` as parameter and handles the HTTP response. If the response code is 500 or otherwise not in the 200...300 range, it throws a custom `InternalServerException` exception all the way back to the `BaseViewModel`.

This setup allows the UI layer to handle any exceptions and directly show corresponding snackbar messages. This is in accordance to the defined dependency directions in the planned architecture.

```kotlin
open class BaseViewModel : ViewModel() {
    protected suspend fun <T> handleCall(
        action: suspend () -> T,
        onError: (Exception) -> Unit = { e -> baseOnError(e) },
    ): T? {
        return try {
            action()
        } catch (e: Exception) {
            onError(e)
            null
        }
    }

    private fun baseOnError(e: Exception) {
        when (e) {
            is ConnectException -> showMessage(...)
            is InternalServerException -> showMessage(...)
            else -> showMessage(...)
        }
    }
}
```

Listing 5.1: BaseViewModel Class

### 5.1.3 Notification Listener

Reacting to FCM notifications when an entry has finished processing, is an integral part of the app.

The Firebase library offers its base class `FirebaseMessagingService` to handle FCM events, including a notification listener. But that alone is not enough to cleanly and correctly react to completed invoice processes.

To achieve this, Kotlin's `SharedFlow` is used. Simply put, a `Flow` in Kotlin shares emitted values to all its collectors/subscribers. By exposing a public `SharedFlow` through the `InvoiceRepository`, all viewmodels are able to subscribe to it. Subsequently, all `MyFirebaseMessagingService` has to do, is emit a value to the flow via the `InvoiceRepository`.

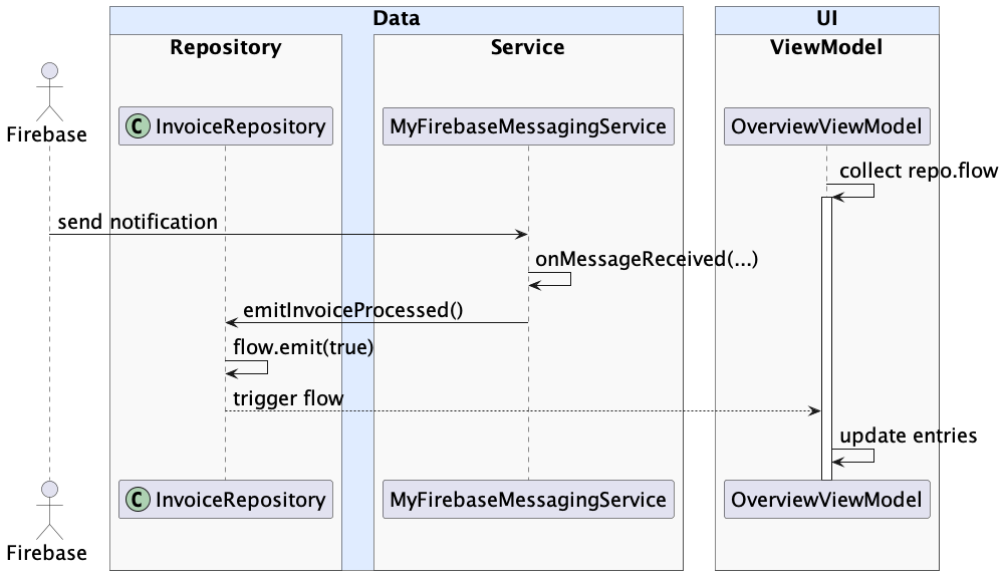


Figure 5.5: SharedFlow Notification

## 5.2 Backend

The implementation of the backend architecture is shown in this section, as well as other relevant details.

### 5.2.1 API Architecture

As planned, the architecture is split up into multiple layers, mainly the routers, services and repositories. There are also modules for data structures, foundational logic (exceptions, dependencies etc.), helper functions and the authentication.

The AWS Python SDK boto3 [20] is used in the repositories to interact with the AWS services and perform the required operations.

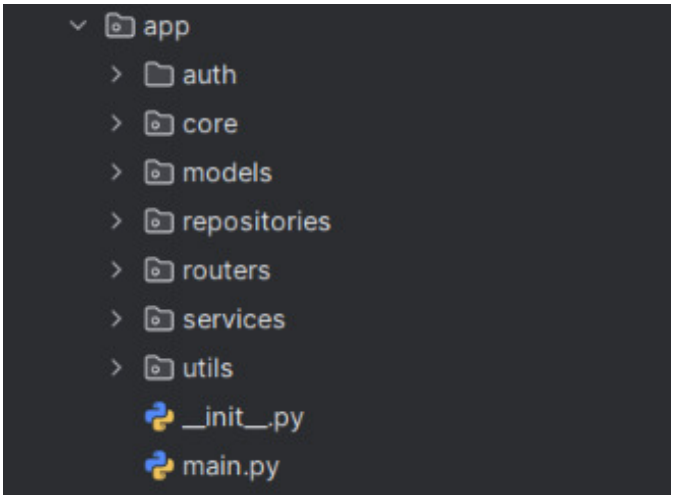


Figure 5.6: API project structure

### 5.2.2 Endpoints

The following tables shows the endpoints the backend provides for the app. These endpoints enable user and invoice related operations.

All invoice related endpoints also contain the `user_id` as a path parameter. This can be read from the JWT token once the PC2 is implemented.

| Method | Endpoint | Description |
|---|---|---|
| GET | /invoices | Retrieves a list of all invoices that belong to a user. |
| GET | /invoices/{invoice_id}/file | Fetches the file associated with a specific invoice. |
| POST | /upload_invoice | Creates a database entry and uploads the invoice file to object storage for further processing. |
| POST | /create_manual_invoice | Creates a new manual invoice entry. |
| POST | /upload_invoice_file/{invoice_id} | Uploads an invoice file which will be associated to a specific invoice entry. |
| PUT | /invoices/{invoice_id} | Updates the details of a specific invoice. |
| DELETE | /invoices/{invoice_id} | Deletes a specific invoice by its ID. |
| DELETE | /invoices/{invoice_id}/file | Deletes the invoice file associated with the defined invoice. |

Table 5.1: Invoice API Endpoints

| Method | Endpoint | Description |
|---|---|---|
| POST | /register_user | Registers a new user in the database. |
| PUT | /update_fcm_token | Updates the FCM token for an existing user. |

Table 5.2: User API Endpoints

### 5.2.3 Data structures

To ensure consistency across the application, data structures are defined. They describe the primary entities and their attributes according to the domain model. These models can be reused throughout the application, and incoming requests can automatically be validated to ensure the data conforms to the required format.

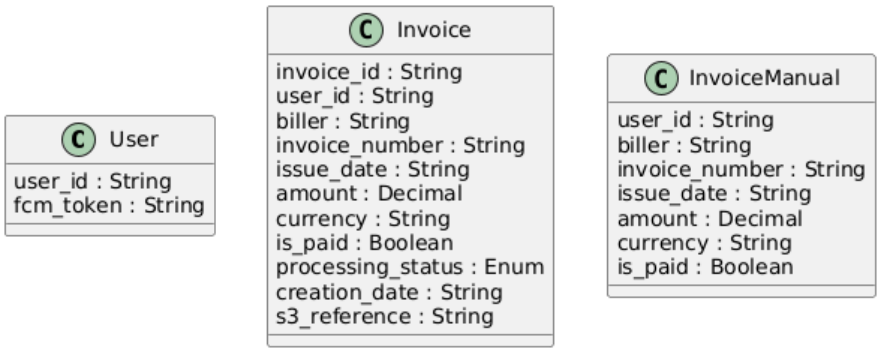


Figure 5.7: Backend models

## 5.3 AWS

The project leverages several AWS services, primarily for storage and compute. Following is an overview of the services and their configurations. By utilizing managed and serverless services, high availability and scalability is achieved.

### 5.3.1 Architecture

As shown in the figure below, Terraform is used to create and manage the required services. Each type of service is placed in its own Terraform file. To securely store the data and protect against corruption in this collaborative environment, the S3 + DynamoDB approach is used. This approach allows saving the state remotely and locking if in use.

The lambda function is placed in its own subdirectory, as well as the lambda layer. This layer contains special dependencies required by the lamda function that are not part of the default modules.

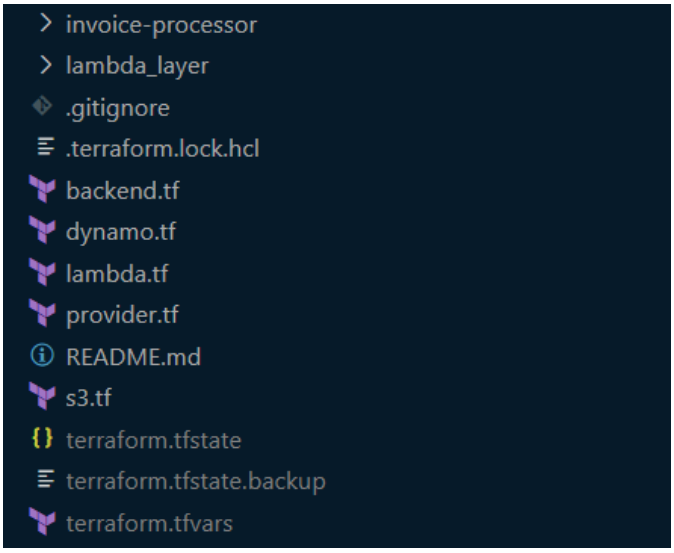


Figure 5.8: AWS Repository

### 5.3.2 S3 / Lambda

S3 is used for storing invoice files. The S3 bucket is attached as a trigger to a lambda function. The function retrieves the uploaded object once an event notification is received. This object is then forwarded to Textract for processing.

The Textract response is a JSON file containing thousands of lines. This first has to be filtered so only the relevant fields remain. Then depending on the confidence score and completeness of the required fields, the database entry is updated.

Information required by the function is supplied via the object metadata in the form of key-value pairs.

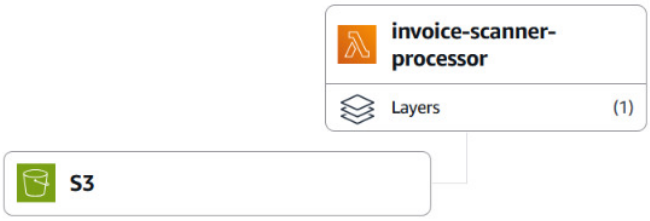


Figure 5.9: Lambda function

### 5.3.3 DynamoDB

DynamoDB is a fully managed, key-value NoSQL database. Two separate tables for the users and invoices are used. The table content always corresponds to the data structures mention in section 5.2.3.

To support all required query types, a GSI is used on the invoices table. This allows for efficient queries using non-key attributes.

# Chapter 6

# Results

This chapter shows the various results and achievements of the project.

## 6.1 NFR Validation

The NFRs are validated twice over the course of this project. Once with the release of the beta and the second time with the release of the MVP. For each NFR, there is a description about the measurment methods, including the results and its acceptance status.

### 6.1.1 Beta Validation

With the beta release, the NFRs are validated for the first time and lead to the following results.

| ID | Measured | Accepted |
|---|---|---|
| NFR1 | Various tests are conducted where the time between confirming the upload and receiving a notification is evaluated. The results depend on the file size of the invoice. Using the maximum filesize of 10MB takes around 12 seconds. Processing time of lower file sizes goes as low as five seconds. | ✓ |
| NFR2 | Multiple stress tests show that the backend is able to handle more than 200 requests per minute. | ✓ |
| NFR3 | All the pages load quickly and well under one second. | ✓ |
| NFR4 | The backend server runs multiple replicas which are loadbalanced. When a node stops working, the health checks detect it. With the current configuration it takes a maximum of four seconds to detect an unhealthy node. | ✓ |
| NFR5 | Invoices stored in S3 by default have a high enough durability. For the data in the DynamoDB tables PITR is used. Using point-in-time-recovery, all tables have the ability to be restored to any given second in the last 35 days. | ✓ |
| NFR6 | The features where the user can input data are not yet complete. | ✗ |
| NFR7 | Pipelines are in place for all repositories where it is relevant. Together with pre-commit hooks they lint and format the projects to ensure quality. | ✓ |
| NFR8 | A Let's Encrypt certificate is installed on the server. With this all connections are made over HTTPS. Calls to AWS only use HTTPS as well. | ✓ |
| NFR9 | All data in the DynamoDB tables and in the S3 bucket are encrypted at rest. | ✓ |
| NFR10 | Under normal circumstances a user only sees his own data which fulfills this requirement. Protection against malicious intent will be present once the second precondition is complete. | ✓ |
| NFR11 | Snackbar notifications pop up if an errors occurs and inform the user with the appropriate information. | ✓ |
| NFR12 | The general feedback from the usability tests is positive. No major issues were reported but small improvement suggestions have been noted. | ✓ |

Table 6.1: Beta NFR Validation

### 6.1.2 MVP Validation

Below is the protocol from the MVP validation of all NFRs.

| ID | Measured | Accepted |
|---|---|---|
| NFR1 | Revalidated. | ✓ |
| NFR2 | Revalidated. | ✓ |
| NFR3 | Revalidated. All pages of the MVP load in under one second. | ✓ |
| NFR4 | Revalidated. | ✓ |
| NFR5 | Revalidated. | ✓ |
| NFR6 | User input which is mainly in the detailed invoice view can't cause the app to crash and is properly validated. Valid file formats for uploading invoices are also defined. | ✓ |
| NFR7 | Revalidated. | ✓ |
| NFR8 | Revalidated. | ✓ |
| NFR9 | Revalidated. | ✓ |
| NFR10 | Revalidated. Protection against malicious intent will be present once the second precondition is complete. | ✓ |
| NFR11 | Revalidated. A failed invoice extraction is marked accordingly and a pop-up with additional information appears. | ✓ |
| NFR12 | General feedback from all usability tests is positive. | ✓ |

Table 6.2: MVP NFR Validation

## 6.2 Final Product

The final product covers all use cases defined for the MVP, specifically UC1 to UC6. Of course the authentication precondition PC1 is also included. The NFRs, as seen in the above section, are all validated and accepted.

In addition to those requirements, some additional, not predefined features are implemented. These features came up during meetings and/or usability tests and were refined, estimated and prioritised in an iterative fashion.

| ID | Description |
|---|---|
| EC1 | On app start, the user's internet connection is checked. If he is not connected, he will be informed and the app will try again every second. |
| EC2 | The user is informed when the processing step fails. This either occurs when the confidence score of Textract is below 50% or if any of the fields are missing after the extraction process. He is informed in the form of an icon in the `OverviewScreen` and a popup in the `DetailScreen`. |
| EC3 | In addition to receiving a notification when processing of an entry is complete, the `OverviewScreen` automatically refreshes when the user is in the app during receival. |
| EC4 | In addition to viewing the invoice file in the `DetailScreen`, the user can also delete it from the entry or replace it. |
| EC5 | Dark mode compatibility. Dark mode is set based on the system settings. |

Table 6.3: Extra Use Cases

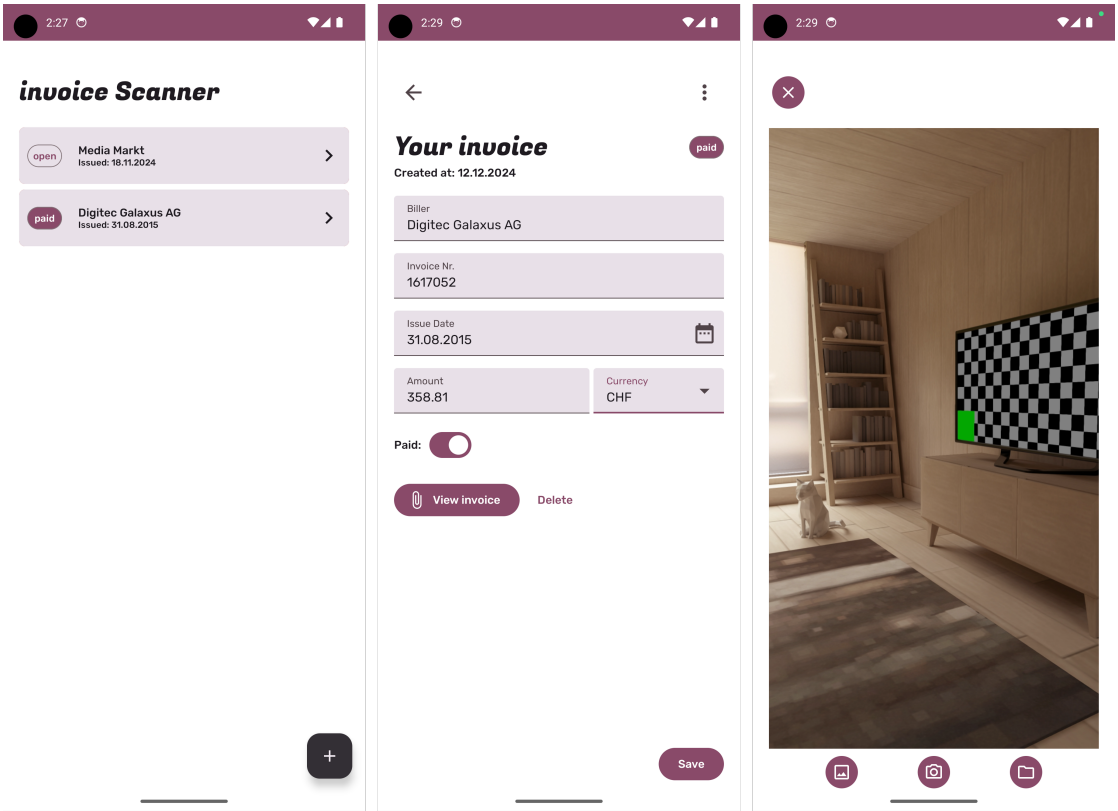


Figure 6.1: MVP App - Light

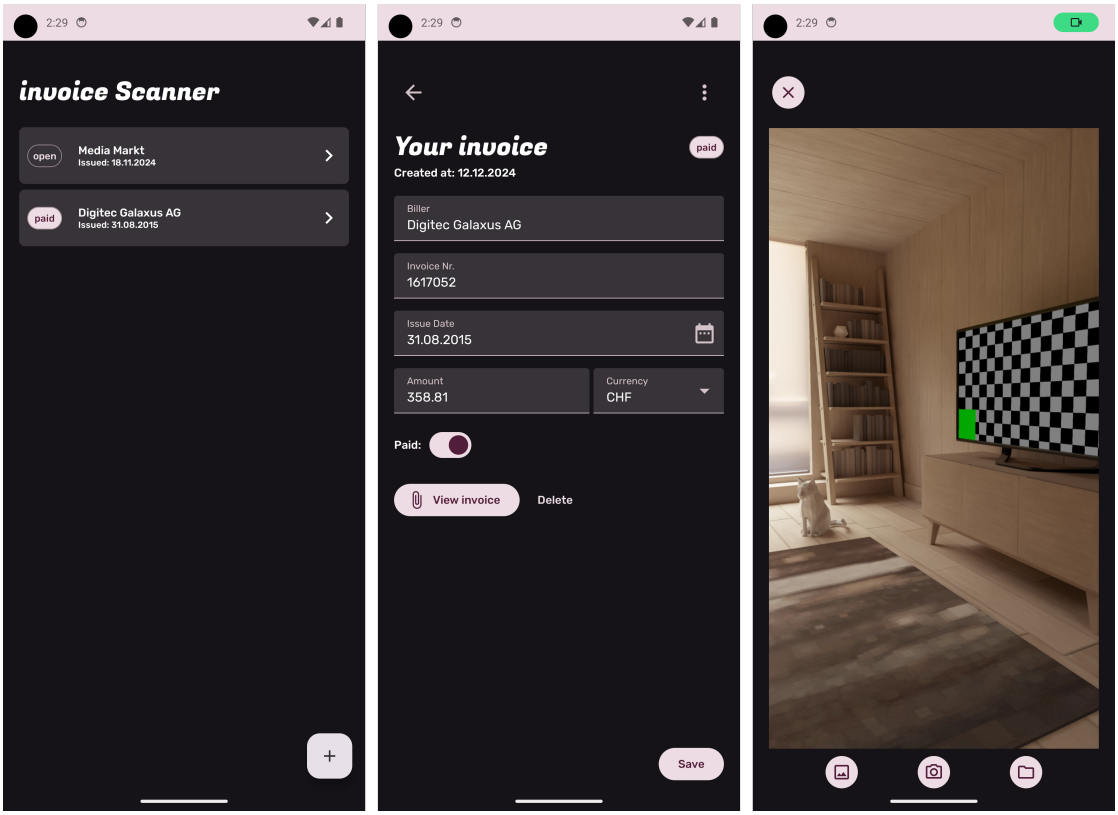


Figure 6.2: MVP App - Dark

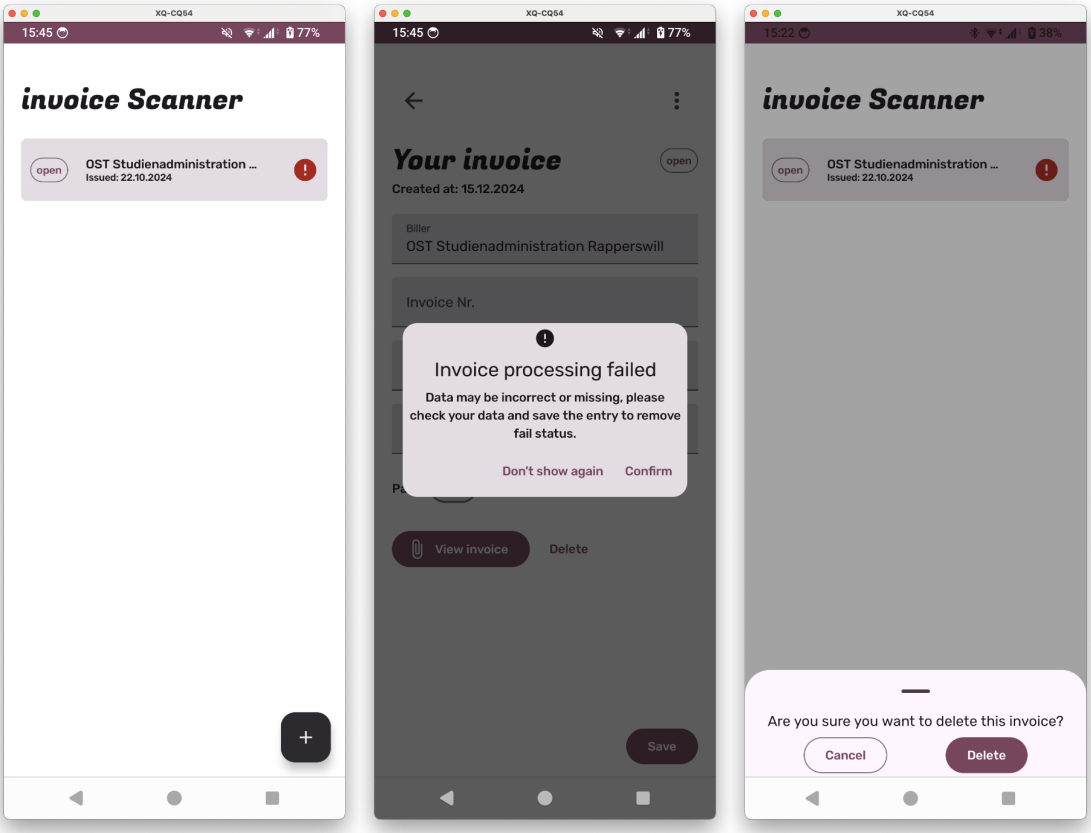


Figure 6.3: MVP App - Processing Failed and Delete


Figure 6.4: MVP App - Tablet Version

# Part III

# Project Documentation

# Chapter 7

# Project Plan

This chapter addresses the planning aspect of the project and the tools that are used.

## 7.1 Planning

This section focuses on short- and long-term planning, as well as methodologies, responsibilities and the identified risks.

### 7.1.1 Methodology

In this project, the OST-original Scrum+ methodology is used, which combines RUP for long-term planning and Scrum for short-term planning. This provides the ability of planning and defining long-term goals, while also keeping the agility of Scrum. Using a sprint length of two weeks results in seven sprints, which are allocated to the four RUP phases.

### 7.1.2 Roles and Responsibility

The following assigned roles and responsibilities do not imply, that the work of the assignee is exclusive them. As a team of two, assistance is provided to one another in various tasks.

**Scrum Master + Project Manager:** Tseten Emjee
Responsibility: Leading Scrum Meetings, Writing Meeting Minutes

**Product Owner:** Roger Marty
Responsibility: Leading Refinement Meetings and sanitation of the backlog

**DevOps:** Roger Marty
Responsibility: Overview and general responsibility over pipelines and infrastructure

**Frontend:** Tseten Emjee
Responsibility: Overview and general responsibility over the mobile app

**Backend:** Roger Marty
Responsibility: Overview and general responsibility over the backend including external systems

**Testing:** Roger Marty
Responsibility: Overview and general responsibility over testing

**Architecture:** Tseten Emjee
Responsibility: Overview and general responsibility over architecture

### 7.1.3 Meetings

Following are the timeslots for the regularly held meetings:

- **Sprint Planning / Review**: Every two weeks on Monday 10h00 - 11h00
- **Refinement Meeting**: Every second Monday in Sprint 10h00 - 11h00
- **Weekly Sync with Advisor**: Every Monday 11h00 - 12h00

### 7.1.4 Long-Term Plan

This is the defined long-term plan. These estimations are rough and subject to change during the course of the project.


Figure 7.1: Long-Term Plan

### 7.1.5 Milestones

Following five milestones mark the key deliverables and completion of phases.

**Milestone 01: Requirements** 14.10.2024
The goal of the first milestone is to establish the foundation for the project and defining the functional and non-functional requirements for the project.

**Planned Deliverables:**

- Define functional requirements
- Define non-functional requirements
- Create a domain model consistent with the defined requirements

**Milestone 02: Prototype** 28.10.2024
This milestone focuses on assuring the selected technologies work together by creating a prototype.

**Planned Deliverables:**

- Prototype App
  - Specifications: Android App that can submit an invoice to an AWS S3 bucket via the backend. Once the invoice is uploaded, Firebase is triggered and a notification is shown on the mobile phone.

**Milestone 03: Beta** 25.11.2024
The Beta milestone signifies the peak of the construction phase and that it soon comes to an end.

**Planned Deliverables:**

- Beta App
  - Specifications: Near feature-complete with bugs and few features remaining.

**Milestone 04: MVP** 09.12.2024
With this milestone the construction phase is over and the MVP is complete.

**Planned Deliverables:**

- MVP App
  - Specifications: Feature-complete with almost no bugs remaining.

**Milestone 05: Final Submission** 20.12.2024

This is the final milestone. It signifies the completion and submission of the project.

**Planned Deliverables:**

- Abstract for brochure
- Final Product
  - Specifications: MVP with no bugs and maybe optional features included.
- Final Documentation
  - Specifications: Complete and submission-ready documentation, proof-read by both team members.

### 7.1.6 Short-Term Plans

The short-term plans rely on Scrum. Based on the progress of the previous sprint, the next sprint is planned in the bi-weekly planning meeting. Planning poker is used to ensure bias-less and consenus-based time estimations for each story. Following is a history of all sprints and their finished tasks/stories.

**Sprint 1**

| **Product** | **Documentation** |
|---|---|
| - Setup various tools | - Complete project planning (Long-term plan, milestones, risks,...) |
| | - Prepare quality measures |

Table 7.1: Short-Term Plan Sprint 1

**Sprint 2**

| **Product** | **Documentation** |
|---|---|
| - Setup DEV environments | - Complete requirements engineering (FR, NFR, OR) |
| - Configure virtual server | - Create C4 model |
| - Configure CI/CD pipelines | - Create domain model |

Table 7.2: Short-Term Plan Sprint 2

**Sprint 3**

| **Product** | **Documentation** |
|---|---|
| - Set up prototype | - Review and adjust risks |
| - Create frontend app foundations | - Create container interaction diagrams |
| - Create backend/AWS foundations | - Document finalized technology decisions |
| - Make low/high fidelity designs | |

Table 7.3: Short-Term Plan Sprint 3

**Sprint 4**

| **Product** | **Documentation** |
|---|---|
| - Setup UUID identification | - General revisions and updates |
| - Implement upload functionality | |
| - Prepare various backend endpoints | |

Table 7.4: Short-Term Plan Sprint 4

**Sprint 5**

| **Product** | **Documentation** |
|---|---|
| - Implement Textract extraction workflow | - Prepare and perform usability tests |
| - Construct backend endpoints | - Re-evaluate risks |
| - Polish upload functionality for different file types | - Validate NFRs |
| - Implement error handling | |
| - Create overview page | |
| - Release beta version | |

Table 7.5: Short-Term Plan Sprint 5

**Sprint 6**

| Product | Documentation |
|---|---|
| - Add manual creation and edit endpoints | - Perform second usability test round |
| - Create detail page | - Re-evaluate risks |
| - Fix bugs and add various improvements | - Re-validate all NFRs |
| - Setup authentication in backend | |
| - Begin with login feature in frontend | |

Table 7.6: Short-Term Plan Sprint 6

**Sprint 7**

| Product | Documentation |
|---|---|
| - Minor changes and improvements | - Write abstract and management summary |
| | - Document usability tests |
| | - Finalise architecture and implementation chapters |
| | - Complete and revise documentation |
| | - Prepare for final submission |

Table 7.7: Short-Term Plan Sprint 7

### 7.1.7 Risk Management

This chapter identifies and visualizes the project risks and how they are handled. A changelog for all risk changes throughout the project is created.

### Identified Risks

| ID | Description |
|---|---|
| Technical Risks | |
| R1 | Loss of project data / code |
| R2 | Unauthorized use of interfaces |
| R3 | Performance issues |
| R4 | Code quality inadequate |
| R5 | Technologies are incompatible |
| Project Risks | |
| R6 | Misjudment of the time schedule |
| R7 | Absence of team member |
| R8 | Lack of team communication |
| R9 | Insufficient knowledge of technologies |
| R10 | Outage of project relevant tools |

Table 7.8: Identified Risks

### Risk Matrix

| **Probabilities** | **Severity** | | | |
|---|---|---|---|---|
| | Negligible | Marginal | Critical | Catastrophic |
| High | | | | |
| Likely | | | | |
| Possible | | | | |
| Unlikely | | R1 | R2, R7 | |
| Rare | R4 | R3, R6, R8 | R9 | R5, R10 |

Table 7.9: Risk Matrix

**Risk Handling**

| ID | Prevention | Mitigation |
|---|---|---|
| Technical Risks | | |
| R1 | Commit often, local backups | Restore from local backup |
| R2 | Do not expose any secrets, keys and ports | Refresh / deactivate keys and secrets |
| R3 | Clean coding, well planned architecture | Review code and architecture |
| R4 | Coding guidelines, linters, piplines and pre-commit hooks | Refactoring |
| R5 | Initial research of technologies | Rapid change of technologies |
| Project Risks | | |
| R6 | Time buffer in planning, review progress of sprints | Adjust scope, prioritize core-functions |
| R7 | Eat healthy, stay fit | Smooth transfer of knowledge and tasks |
| R8 | Sync often, work together | Extraordinary meetings |
| R9 | Sufficient time for research | Check documentation, ask in forums or discuss with advisor |
| R10 | Check for alternatives | Quickly switch to alternatives |

Table 7.10: Risk Handling

**Risk Changelog**

| Date | Change | Cause |
|---|---|---|
| 17.10.2024 | R5: Possible/Catastrophic to Rare/Catastrophic | Working prototype as defined in milestone |
| 25.11.2024 | R6: Possible/Critical to Unlikely/-Critical | Beta released on time, according to milestone |
| 25.11.2024 | R9: Unlikely/Catastrophic to Rare/Critical | Multiple features already implemented and no issues occured |
| 07.12.2024 | R6: Unlikely/Critical to Rare/-Marginal | All planned MVP features have been implemented |

Table 7.11: Risk Changelog

## 7.2 Tooling

Various management tools used in the project are noted in this section.

### 7.2.1 Documentation

The documentation is written in `LaTeX` and stored in a GitLab repository where the configured pipeline builds the output PDF on every commit. This provides version control and the possibility to rollback to a previous version if needed. To ensure that the documentation corresponds with the OST guidelines and requirements, the LaTeX template from the module SE Project is used.

### 7.2.2 Code

The codebase is divided into frontend, backend, AWS-related code/configuration. For each codebase, a separate repository in the GitLab subgroup is created. This provides clear separation and each repository has its own pipeline with automatic builds and tests. The infrastructure configurations are saved in a repository as well.

### 7.2.3 Tracking

Everything that needs to be tracked is reflected in Jira. The sprints, epics, issues, bugs and also the working hours of each member are recorded there.
(Jira Board)

### 7.2.4 Workflow

The steps the tasks go through is illustrated in the workflow below. All tasks start in the "To Do" column. If a team member starts working on a task, it is forwarded to "In Progress" and stays there until everything is implemented. On completion the tasks goes to "In Review" where another team member checks the work. If it is acceptable it goes into the "Testing" column. If all tests pass, the task can be marked as done.

In certain states the tasks have the possibility to be moved back to a previous state if, for example if the tests fail.

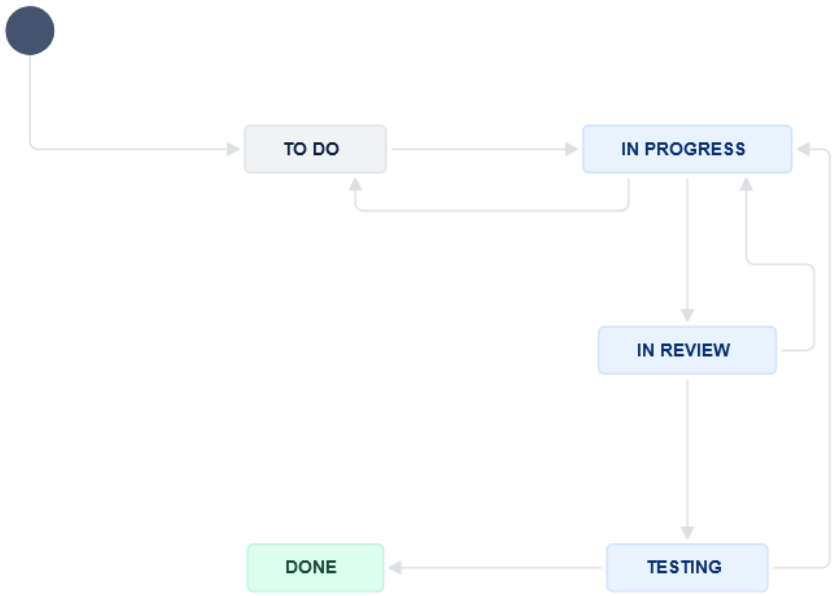


Figure 7.2: Jira Workflow Scheme

# Chapter 8

# Quality Measures

This chapter shows the measures and tools used to ensure high product quality. This starts with the code itself, the testing and goes all the way to the pipelines and workflows used in the repositories.

## 8.1 Code

To ensure that a high standard of code quality gets maintained, linters and formatters are in use. The frontend and backend each have their respective tools. With linters the code is statically analyzed and the programmer is informed about any formatting violations, programming errors or bugs. The use of formatters ensures that the code adheres to defined guidelines and conventions.

| | **Frontend (Kotlin)** | **Backend (Python)** |
|---|---|---|
| Linter | ktlint [21] | Flake8 [22] |
| Formatter | ktlint [21] | Black [23] |

Table 8.1: Code Quality Tools

The linters and formatters are executed in the pipeline on every commit to the repository, every merge and also locally as pre-commit hooks.

## 8.2 Gitflow

All code repositories use the same Git workflow described below. Each merge from one branch into another branch requires a review and an approval by another team member and all pipelines need to be successful. Then the team member that requested the merge can confirm it.

| Branch | Description |
|---|---|
| Main Branch | Always contains the latest stable release of the software which is used in production environments. |
| Develop Branch | Branched from the main branch in the beginning. If new release approaches and develop branch is stable and pipelines are successful, it gets merged into main branch. |
| Feature Branch | Every issue that gets handled gets a separate feature branch, branched of from the latest develop branch.<br>Naming convention: feature/[issueID]-[issueTitle] |
| Bugfix Branch | Every bug that needs to be fixed requires a separate bugfix branch either from the latest develop branch or directly from the main branch depending on the severity.<br>Naming convention: bugfix/[issueID]-[issueTitle] |

Table 8.2: Gitflow Branch Description

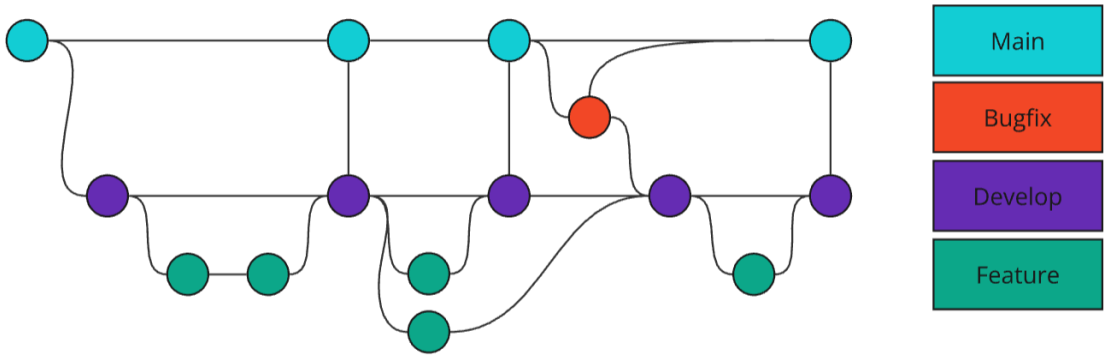


Figure 8.1: Gitflow Workflow

## 8.3 DoR / DoD

- DoR: Issue is ready for estimation in terms of effort and can be put in the backlog.
  - Content of the issue is understood
  - Acceptance criteria / requirements are defined
  - Estimated effort is documented
  - Can be completed in one sprint

- DoD: Issue status can be changed to "Done".
  - Acceptance criteria are implemented
  - Tests and pre-commit hooks pass
  - Pipelines are successful
  - Documentation is updated

## 8.4 Metrics

The test coverage metric is used to test the source code for quality and reliability. This is the main and only code metric used in this project. The percentage is calculated on each commit and shown to the developers on each merge request. This ensures that tests are not forgotten and reminds the developers to cover enough code with the tests. At the same time, not all the available time should go into testing just so this number is high enough which is why the target is set at 70%. This is the same reason why no other metrics are used that could be too distracting.

## 8.5 Testing

In addition to the testing methods mentioned below, regression testing is also implemented. This ensures that changes do not introduce new errors into already existing functionality. This is implemented by continuously checking all unit tests and by testing the new functionality manually.

### 8.5.1 Frontend

Several types of testing are used for the frontend.

#### Usability Tests

To evaluate the user's experience when using the app and its functionalities, usability tests are performed. A few testers are selected and work through the test protocol described below in a one-on-one session with a team member who observes and notes the results. Through this, insight is gained about what could be implemented in a different way and if the interface is intuitive or needs changes. These findings are then discussed in a meeting and if needed, implemented in the next sprint. The usability tests are done once enough functionality exists, specifically after the beta and after the MVP release. Following things are needed for conduct a usability test:

- Team member
- Test participant
- Test smartphone containing the latest stable release of the app

- Private area
- Internet

The usability test protocols for the beta and MVP releases can be found in the appendix.

#### Unit Testing

Next to usability tests the code itself also needs to be tested. For this unit tests are used. This enables specific code sections to be validated regarding their functionality. The tests are implemented using the JUnit4 [24] framework. Written tests are executed with the pre-commit hooks and in the GitLab pipelines. A merge can not be approved if there are tests that fail.

### 8.5.2 Backend

The backend entirely relies on unit tests. With them, code components can be checked to see if they match expected behaviour. The unit tests are conducted using the pytest [25] framework. Written tests will be executed with the pre-commit hooks and in the GitLab pipelines. A merge can not be approved if there are tests that fail.

## 8.6 Pipelines

The details regarding the actual implementation and steps of the CI/CD process can be seen in chapter 3.
To ensure that above described quality measures are enforced throughout the project, pipelines are configured for each repository. These pipelines run on every commit and every merge. Linters, formatters and tests are executed and metrics are calculated and shown in the merge requests. The Gitflow workflow is enforced through the appropriate GitLab settings.

# Chapter 9

# Project Monitoring

This chapter focuses on the tracking and evaluation of the project progress.

## 9.1 Time Tracking Reports

Jira is used to keep track of when and for how long a developer works on a task. For each task an issue is created and assigned to a project member. This enables the possibility to get detailed insights about the time spent on the project which can be seen through the following reports:

- **Total Time** (Link)
- **Sprint 1** (Link)
- **Sprint 2** (Link)
- **Sprint 3** (Link)
- **Sprint 4** (Link)
- **Sprint 5** (Link)
- **Sprint 6** (Link)
- **Sprint 7** (Link)

## 9.2 Time Evaluation

This section illustrates the logged working hours. The total time availabe for this project is 480 hours (2x240). Ultimately, 491 hours were spent.

### 9.2.1 Work Distribution

Following graphic shows the accumulated hours logged for each team member. This corresponds almost exactly to the mandatory effort.

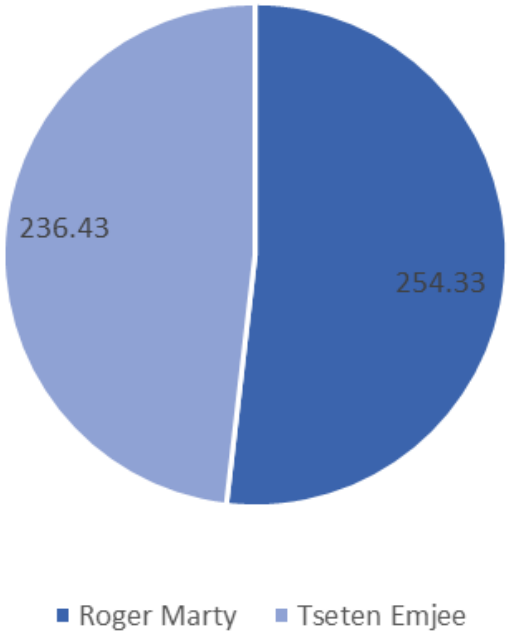


Figure 9.1: Work distribution

### 9.2.2 Work History

This figure shows the logged hours per author, per sprint. Throughout the whole project the hours spent are similar between both team members.

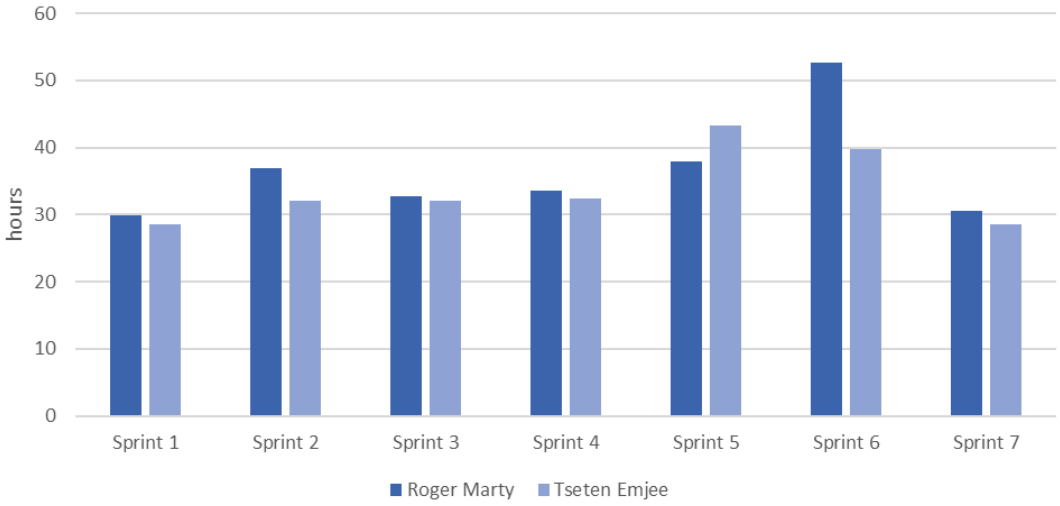


Figure 9.2: Work history

### 9.2.3 Overview Epics

The following pie chart shows the amount of assigned issues each epic has. The category "None" includes bugfixes and epics that do not have a parent.

Important to note here, is that the amount of issues does not equal the workload of an epic. Some issues are more complex than others.

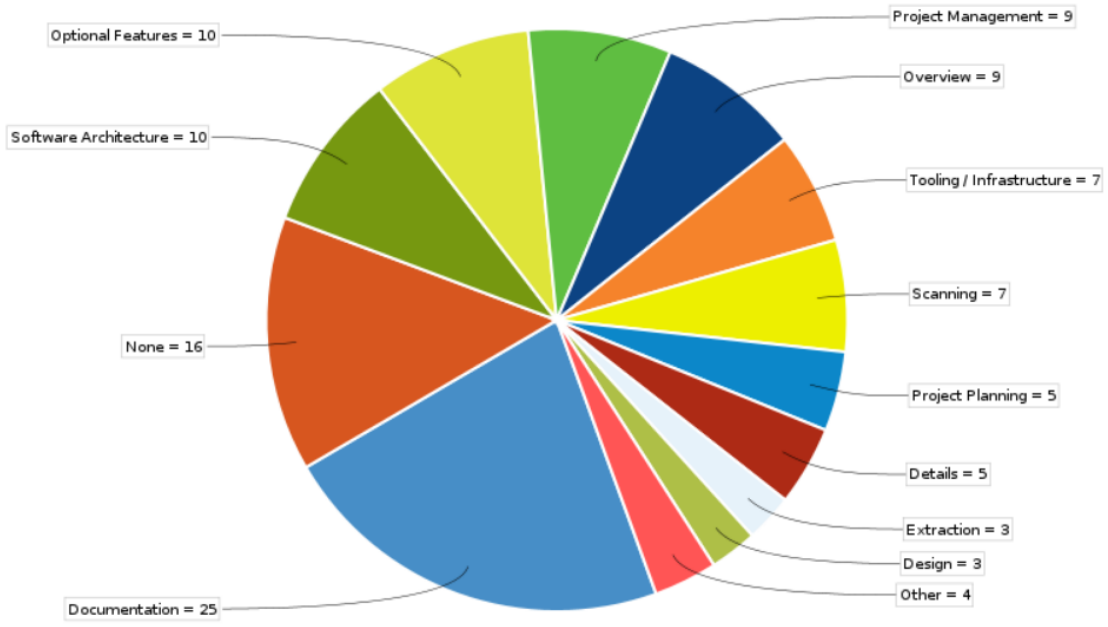


Figure 9.3: Overview of epics

### 9.2.4 Project Timeline

The following timeline shows when which epic was worked on. This graph fairly accurately mirrors the long term plan created at the start of the project.

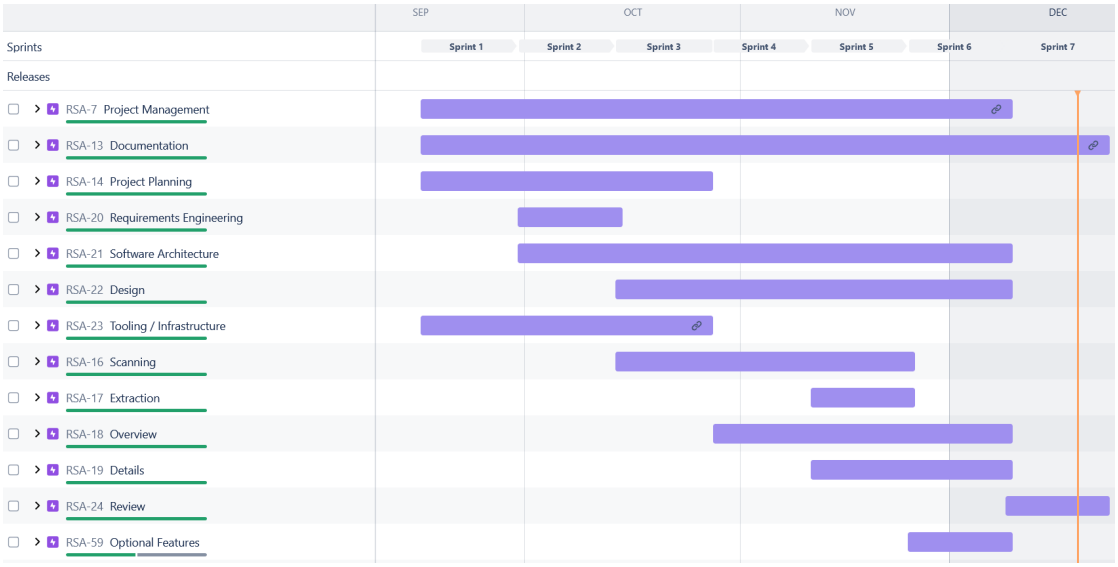


Figure 9.4: Project timeline

### 9.2.5 Milestone Fulfilment

All five defined milestones were achieved on time. In general, it can be said that the project planning is very consistent with the actual accomplished work. Part of the buffer was needed but the project was still completed on time.

## 9.3 Repository Analytics

This section covers various repository statistics including the defined test coverage metric.

### 9.3.1 Test Coverage

The goal of achieving a minimum of 70% test coverage was defined at the start of the project. Even if testing sometimes had to be postponed, the goal was ultimately achieved in the frontend and backend. Following figures show the development of the metric on the develop branch throughout the project.

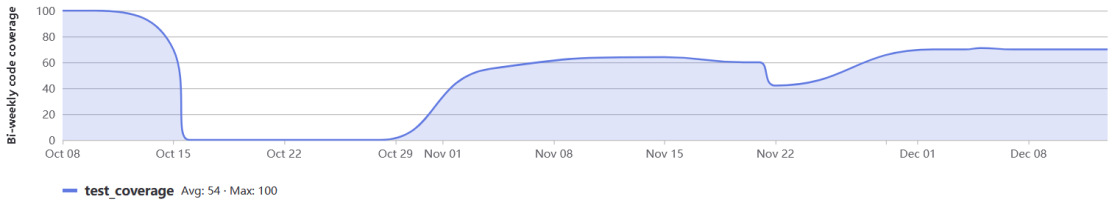


Figure 9.5: Frontend test coverage

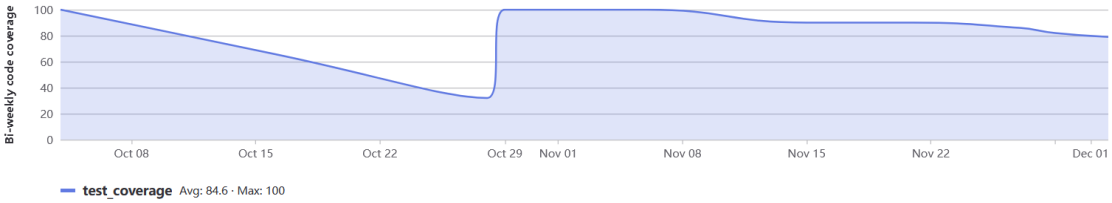


Figure 9.6: Backend test coverage

### 9.3.2 Commits

The amount of commit is mostly spread out. All repositories show a spike during the initialisation phase, followed by a break where the project planning was the main focus. Afterwards the commits are evenly spread out.

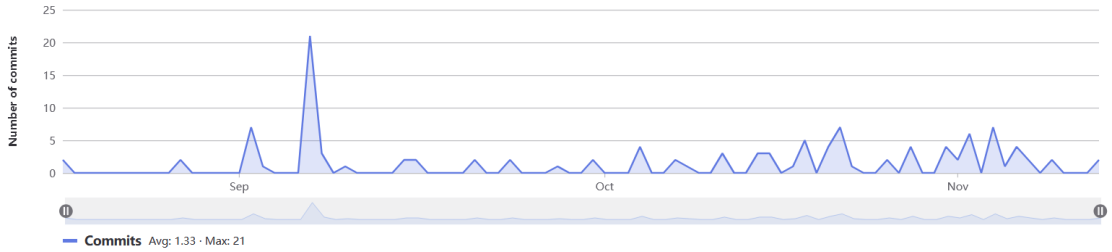


Figure 9.7: Frontend commits

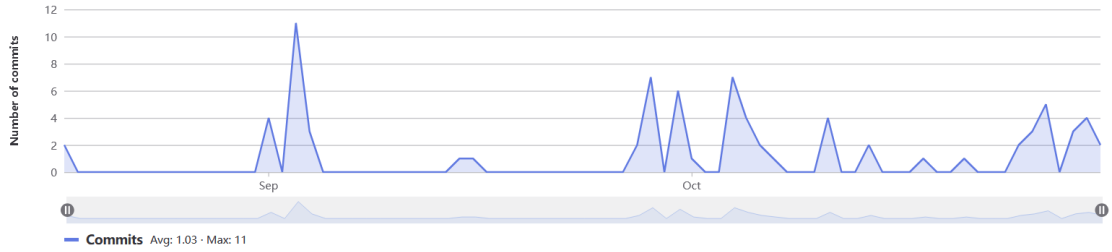


Figure 9.8: Backend commits

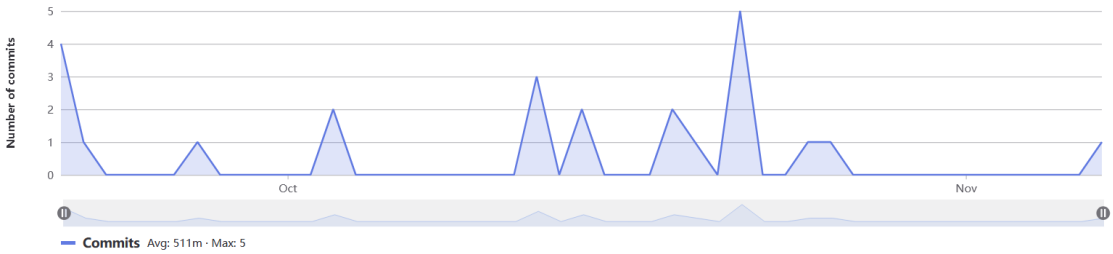


Figure 9.9: AWS commits

# Part IV

# Closing Thoughts

# Chapter 10

# Conclusion

This chapter evaluates the success of the product and the project as a whole, while also outlining future plans.

## 10.1 AI-Integration

The usage of Textract was a success. Due to Textract being a part of the AWS ecosystem the integration went seamlessly. Having a dedicated extension specifically for managing invoices proved to also be highly beneficial.

As expected, there were cases where it came to its limits and was not able to extract all required information. Its ability to extract data proved to be sufficient for the needs of the product.

## 10.2 Evaluating Success

To reiterate, the aim of this project was to create a solution, that provides a central place where one can manage their invoices. By simply uploading a picture or the original PDF file of the invoice, the application should scan and extract relevant information. This information is then available for viewing, editing and deletion.

Based on this goal, 9 use cases and 2 preconditions were defined, alongside 12 NFRs to ensure quality. For the MVP, use cases UC1 to UC6 were mandatory, as well as the precondition PC1. The NFRs all had to be validated and accepted. The MVP was defined, so that upon meeting all requirements, the goal and aim of this project would be achieved.

Reviewing the documented results, the project was successful. All MVP requirements have been met and some additional features have also been implemented to create a better product.

## 10.3 Future

With the mandatory use cases completed, the optional use cases UC7, UC8 and UC9 remain open. In addition, the precondition PC2, providing login authentication with JWT, was started in Sprint 6 but could not be completed in time.

So the future path of the product is quite clear, with focus on completing the new authentication method and the open use cases that expand the functioniality of the currently plain `OverviewScreen`. There were also requests for a `Due Date` field from the conducted usability tests, which would require reviewing the AWS Textract capabilities.

While the team considers the codebase clean, a review and consolidation of the current state would be recommended before starting work on the new features.

# Chapter 11

# Personal Reports

This chapter contains personal reports by the team members, reflecting on their own work in this project.

## 11.1 Tseten Emjee

This project was my second time working on a fully native Android app written in Kotlin. This being the case, I was able to quickly start into the project. But still there were some lessons to be learned.

In general the work was smooth, many things of the Android app development process I was already familiar with. The cooperation with Roger Marty also went well. We split up the workload according to our strengths.

What I struggled with, was working with the file system and files in general in Android. There was an initial hurdle regarding permissions and figuring out a central snackbar messaging system also proved to be difficult.

The Jetpack UI code while easy and quick for me to write, I can say that I wasn't very consistent with parameterizing the composables and the nestings of them. Also regarding project managment in general, the estimations were often too optimistic. So one key takeaway for me is the importance of being more generous with my time estimations. Another one would be not to forsake clean code for speed.

My personal goal in this project was to create a clean and best-practice oriented app architecture. Particularly the data layer, split up into repository and data sources was a focus for me. So being able to achieve this, was a big highlight for me. In addition using the more in-depth features of Kotlin like higher-order functions and `SharedFlow` for event listing was enjoyable as well.

Overall I am very statisfied with our work in this project. It was an important learning experience for me.

## 11.2 Roger Marty

For me personally, the project was a success. It was not the first collaboration with Tseten, and, as with previous projects, the teamwork in this one went really well. We had constant exchanges and were always up-to-date.

We went into the project knowing our respective strengths and split up the work accordingly. Tseten was thus ably to fully focus on the app, while I covered the backend, AWS and infrastructure side.

The main area I need to improve is clearly time estimation. It happened quite often that I was too optimistic when planning the next iteration of issues and the time required for them. While the discrepancies were not too large, it happened and should be considered more carefully in future projects.

One challenge I encountered was regarding the Textract API. The service uses different interfaces when working with single-page versus multi-page documents. This was not clearly highlighted in the documentation and led to an unexpected hurdle, but I was still able to implement it, although not in the cleanest approach, which would require an additional service to be integrated.

Since I did not create that many backends entirely on my own, additional research was required. It also led to the fact, that I optimized the backend structure a bit in the middle of the construction phase. This brought me valuable insight and has not kept me from completing other issues.

One of my highlights had to do with the partially implement login/authentication feature. Security always plays a big part in an application and although the optional feature could not be finished on time, I learned a lot. Simply the decision as to which provider to choose was an important step. I first started with AWS Cognito, because AWS services were already used in this project. But after research I stumbled across Auth0, which specialises in the authentication domain. Their documentation and overall features were superior.

The decision to go with FastAPI was nice as well. It was a fitting framework and I was able to create a clean backend while following the best practices.

# Chapter 12

# Note of Thanks

We would like to take this opportunity to thank our advisor Martin Seelhofer for his time and effort. He provided us with an interesting project idea after we reached out to him.

The weekly meetings were pleasant and informative. In the meetings he guided us through challenges and uncertainties that arised. This ensured that we stayed on track and were able to complete this thesis. Even outside of meetings we could contact him and always received a quick and helpful response.

# Part V

# Lists

# Glossary

Table 12.1: Glossary

| Term | Description |
|---|---|
| API | Application Programming Interface offers endpoints which allows other applications to access data or other features. |
| AWS | Amazon Web Services is a widely used cloud that offers many different services. |
| CID | A Component Interaction Diagram acts as a high-level sequence diagram and shows how each component of the C4 model interacts in their respective use cases. |
| CI/CD | Through Continuous Integration and Continuous Deployment developers can make changes quicker and more reliable. |
| DoD | Definition of Done says when an issue can be defined as done. |
| DoR | Definition of Ready describes when an issue is ready to be put in the backlog. |
| FCM | Firebase Cloud Messaging allows the sending of cross-platform messages for Android, iOS and web application. |
| FR | Functional Requirements define what features and functionalities the software should have. |
| GSI | A global secondary index can be used when queries using attributes other than the specified key attributes are required. |
| HTTPS | To guarantee security in transit the Hypertext Transfer Protocol Secure can be used. |

Continued on next page

Table 12.1 – continued from previous page

| Term | Description |
|---|---|
| IaC | Instead of creating e.g. AWS services via the web interface everything can be defined in code using Terraform for example. Infrastructure as Code helps with automation and version control. |
| JWT | JSON Web Token, defines a compact and self-contained way for securely transmitting information between parties as a JSON object. |
| LOC | Lines of Code is often used as a metric in software projects. It stands for the number of lines in the source code. |
| ML | Machine Learning is the study of algorithms which learn from data and then apply it to unseen data. |
| MVP | The Minimum Viable Product describes the minimal usable product. |
| NFR | Non-Functional Requirements describe how the software should behave. |
| OCR | Optical Character Recognition refers to the process of extracting contents from an image and convert it into machine-readable text. |
| RUP | Rational Unified Process describes a software development process. |
| S3 | Refers to the AWS Simple Storage Service which is an object storage service that offers high availability, scalability and security. |
| Scrum | Scrum is a project management framework built on being agile. |
| UUID | Is a Universally Unique Identifier to identify entities. |

# Bibliography


[1] ISO25000, "ISO/IEC 25010," https://iso25000.com/index.php/en/iso-25000-standards/iso-25010?linkId=100000045879485&utm_pview=8, 2022, [Online; accessed 26-September-2024].

[2] GSMArena, "Samsung Galaxy A21," https://www.gsmarena.com/samsung_galaxy_a21-10172.php, 2024, [Online; accessed 7-October-2024].

[3] Google, "Jetpack Compose," https://developer.android.com/compose, 2024, [Online; accessed 27-September-2024].

[4] tiangolo, "FastAPI," https://fastapi.tiangolo.com/, 2024, [Online; accessed 19-December-2024].

[5] HashiCorp, "Terraform AWS Provider," https://registry.terraform.io/providers/hashicorp/aws/latest/docs, 2024, [Online; accessed 19-December-2024].

[6] Traefik, "Traefik Proxy," https://doc.traefik.io/traefik/, 2024, [Online; accessed 19-December-2024].

[7] containrrr, "watchtower," https://github.com/containrrr/watchtower, 2024, [Online; accessed 19-December-2024].

[8] AWS, "Amazon S3," https://aws.amazon.com/de/s3/, 2024, [Online; accessed 19-December-2024].

[9] ——, "AWS Lambda," https://aws.amazon.com/de/lambda/, 2024, [Online; accessed 19-December-2024].

[10] ——, "Amazon Textract," https://aws.amazon.com/textract/, 2024, [Online; accessed 19-December-2024].

[11] ——, "Amazon DynamoDB," https://aws.amazon.com/dynamodb/, 2024, [Online; accessed 19-December-2024].

[12] Firebase, "Firebase Cloud Messaging," https://firebase.google.com/docs/cloud-messaging, 2024, [Online; accessed 19-December-2024].

[13] Google, “App Architecture Overview,” https://developer.android.com/topic/architecture, 2024, [Online; accessed 30-September-2024].

[14] Google, “Hilt,” https://dagger.dev/hilt/, 2024, [Online; accessed 27-September-2024].

[15] AWS, “Amazon Elastic Container Service,” https://aws.amazon.com/ecs/, 2024, [Online; accessed 19-December-2024].

[16] ——, “Amazon Elastic Kubernetes Service,” https://aws.amazon.com/de/eks/, 2024, [Online; accessed 19-December-2024].

[17] Figma, “Figma,” https://www.figma.com/, 2024, [Online; accessed 30-September-2024].

[18] Material Design, “Material 3 Design Kit Plugin,” https://www.figma.com/community/file/1035203688168086460/material-3-design-kit, 2024, [Online; accessed 30-September-2024].

[19] ——, “Material Theme Builder Plugin,” https://www.figma.com/community/plugin/1034969338659738588/material-theme-builder, 2024, [Online; accessed 30-September-2024].

[20] AWS, “AWS SDK für Python (Boto3),” https://aws.amazon.com/de/sdk-for-python/, 2024, [Online; accessed 19-December-2024].

[21] J. Leitschuh, “ktlint-gradle,” https://github.com/JLLeitschuh/ktlint-gradle, 2024, [Online; accessed 26-September-2024].

[22] P. C. Q. Authority, “Flake8: Your Tool For Style Guide Enforcement,” https://flake8.pycqa.org/en/latest/#, 2024, [Online; accessed 26-September-2024].

[23] P. S. Foundation, “black,” https://github.com/psf/black, 2024, [Online; accessed 26-September-2024].

[24] JUnit, “JUnit 4,” https://junit.org/junit4/, 2024, [Online; accessed 26-September-2024].

[25] pytest, “pytest: helps you write better programs,” https://docs.pytest.org/en/stable/, 2024, [Online; accessed 26-September-2024].

# List of Figures

# List of Tables

# Listings

# Part VI

# Appendix

# Task Description

# Rechnungs-Scanner App für Android oder iOS

Studienarbeit Informatik HS24

## 1. Betreuer

Diese Studienarbeit wird von Martin Seelhofer (martin.seelhofer@ost.ch) betreut.

## 2. Studierende

Diese Arbeit wird als Studienarbeit an der Abteilung Informatik durchgeführt von:

- Tseten Emjee, tseten.emjee@ost.ch
- Roger Marty, roger.marty@ost.ch

## 3. Beschreibung

Im Rahmen dieser Studienarbeit soll eine Android oder iOS App entwickelt werden, die es Nutzern ermöglicht, Rechnungen zu fotografieren. Die Inhalte der fotografierten Rechnungen sollen anschliessend in einem Backend mittels Künstlicher Intelligenz analysiert und in Form von strukturierten Daten in einer Datenbank abgelegt werden. In der App sollen die Daten als JSON aus dem Backend abgerufen und in einer tabellarischen Übersicht angezeigt werden.

## 4. Aufgabenstellung

Folgende Aktivitäten sollen im Rahmen der App Entwicklung mittels einer agilen Vorgehensweise durchgeführt werden:

- Anforderungsanalyse: Definition der Anforderungen und Spezifikationen der App.
- App-Entwicklung: Implementierung der App mit einer benutzerfreundlichen Oberfläche zur Rechnungserfassung (unbedingt vorgefertigte Libraries nutzen!).
- KI-Integration: Integration eines KI-Moduls zur Auswertung und Datenextraktion. (Eigenentwicklung aus Aufwandsgründen nicht empfohlen, besser APIs nutzen, z.B. OpenAI)
- Datenstrukturierung: Ablegen der extrahierten Rechnungsdaten in einer Datenbank und Bereitstellung als JSON-Struktur.
- Test und Validierung: Testen der App auf einem Gerät und Validierung der KI-Ergebnisse.
- Dokumentation: Ausführliche Dokumentation des Entwicklungsprozesses, der eingesetzten Technologien und der erzielten Ergebnisse.

Der genaue Funktionsumfang der App wird in Absprache mit dem Betreuer festgelegt, bzw. ergibt sich aus dem Projektfortschritt.

## 5. Zur Durchführung

In der Studienarbeit geht es darum, das in den verschiedenen Modulen der HSR/OST gelernte Wissen in einem Projekt anzuwenden. Insbesondere Ihre Software Engineering Fähigkeiten werden dabei gefordert sein. Es wird erwartet, dass Sie dieses Wissen anwenden und Methoden wie zum Beispiel Unit Testing, Clean Code und Continuous Integration, wenn immer möglich anwenden. Vorkenntnisse in den Bereichen KI, App-Entwicklung und UX-Design werden Ihnen dabei helfen, die Umsetzungsrisiken tief zu halten.

Während der Bearbeitung findet normalerweise einmal wöchentlich eine Besprechung mit dem Betreuer statt. Zusätzliche Besprechungen können nach Bedarf der Studierenden individuell veranlasst werden.

Alle Besprechungen (ausser Kick-off) sind von den Studierenden mit einer **Traktandenliste** (Was wurde gemacht? Was wurde erreicht? Was nicht? Welche Fragen haben wir?) vorzubereiten und zu leiten. Im Regelfall sollte eine Besprechung maximal eine Stunde dauern. Die Ergebnisse der Besprechungen (also die getroffenen Entscheidungen) sind durch die Studierenden zu protokollieren und an den Betreuer anschliessend zuzustellen oder an einem definierten Ort (z.B. Wiki) abzulegen.

## 6. Werkzeuge

Sofern nicht in der Aufgabenstellung vorgeben sind die Studierenden für die Auswahl Ihrer Werkzeuge, Libraries, Frameworks, SaaS-Angebote, etc. selbst verantwortlich. Auf Wunsch kann eine an der OST gehostete virtuelle Maschine zur Verfügung gestellt werden.

## 7. Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen (siehe Sharepoint, hier finden Sie auch weitere Reglemente und Anleitungen). Alle Dokumente sind nachzuführen, d.h. sie sollten den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren. Es ist eine Zeiterfassung zu führen und im Bericht auszuwerten.

## 8. Termine

Eine Übersicht der Termine finden Sie auf Sharepoint.

| | |
|---|---|
| **16.09.2024** | Beginn der Studienarbeit, Ausgabe der Aufgabenstellung durch den Betreuer. |
| **Okt/Nov** | Regelmässige Abstimmunsmeetings zum Projektfortschritt |
| **16.12.2024** | Die Studierenden erfassen den Broschüren-Abstract im Online Tool http://abstract.rj.ost.ch/ und geben den Broschüren-Abstract zur Kontrolle an ihren Betreuer frei. |
| **19.12.2024** | Der Betreuer gibt das Dokument mit dem korrekten und vollständigen Broschüren-Abstract zur Weiterverarbeitung an das Studiengangsekretariat frei. |
| **20.12.2024** | Abgabe des Berichts an den Betreuer und hochladen aller Dokumente auf https://avt.i.ost.ch/ bis 17 Uhr |
| **Bis zum 10.02.2025, 12:00** | Abgabe SA-Note |

## 9. Beurteilung

Eine erfolgreiche Studienarbeit zählt 8 ECTS Punkte pro Studierenden. Für 1 ECTS Punkt soll mit einer Arbeitsleistung von ~30 Stunden gerechnet werden. Für die Beurteilung ist die Betreuungsperson zuständig.

| Gesichtspunkt | Gewicht |
|---|---|
| 1. Organisation und Durchführung | 20% |
| 2. Formale Qualität des Berichts | 20% |
| 3. Analyse, Entwurf und Auswertung | 20% |
| 4. Technische Umsetzung | 40% |

Die Note wird den Studierenden über unterricht.ost.ch bekannt gegeben. Die Betreuungsperson kann/darf angeben, ob die Arbeit als bestanden/nicht bestanden gilt, er darf auch vorwarnen, wenn das Risiko auf eine ungenügende Note besteht.

Im Übrigen gelten die Bestimmungen der Abteilung Informatik für Studienarbeiten gemäss «Leitfaden für Bachelor- und Studienarbeiten» in Version 1.2, gültig ab HS 23.

St. Gallen, den 16. September 2024

Martin Seelhofer

Dozent für Informatik
OST – Ostschweizer Fachhochschule

# Usability Tests

This chapter contains the usability testing protocol as well as the results of all conducted usability tests.

Table 12.2: Usability Testing Protocol

| Nr. | What | Description |
|---|---|---|
| 1 | First impression | Open the Invoice Scanner app on the smartphone.<br>• Is the use case of the app identifiable?<br>• How do you rate the design of the app?<br>• Do you know what your next step would be? |
| 2 | Upload invoice | Upload an new invoice using the app.<br>• How do you rate the step of uploading a new invoice (use of camera/file explorer)?<br>• Did you see the different options available during this process?<br>• From 1 to 10, how easy and intuitive was the process?<br>• How could this process be improved? |

Continued on next page

Table 12.2 – continued from previous page

| Nr. | What | Description |
|---|---|---|
| 3 | Overview page | Check the result of the invoice you just uploaded using the overview page.<br>• Do you understand what is happening with the invoice (processing status)?<br>• Is the overview compact enough and does it still show everything important?<br>• From 1 to 10, how easy and intuitive was the process?<br>• How could this process be improved? |
| 4 | Invoice entry | Check the details of the processed invoice and the original image.<br>• Are the fields relevant to you?<br>• Are fields you deem important missing?<br>• From 1 to 10, how easy and intuitive was the process?<br>• How could this process be improved? |
| 5 | View invoice file | Open the original invoice file that was uploaded.<br>• From 1 to 10, how easy and intuitive was the process?<br>• How could this process be improved? |
| 6 | Edit entry | Edit the amount field of the scanned invoice, mark it as paid and save the changes.<br>• From 1 to 10, how easy and intuitive was the process?<br>• How could this process be improved? |
| 7 | Delete invoice file | Delete the invoice file from the created invoice.<br>• From 1 to 10, how easy and intuitive was the process?<br>• How could this process be improved? |

Continued on next page

Table 12.2 – continued from previous page

| Nr. | What | Description |
|---|---|---|
| 8 | Delete invoice | Delete the whole invoice entry.<br>• From 1 to 10, how easy and intuitive was the process?<br>• How could this process be improved? |
| 9 | Create entry | Create a manual entry instead of uploading an invoice.<br>• From 1 to 10, how easy and intuitive was the process?<br>• How could this process be improved? |
| 10 | Add file to entry | Add an invoice to the manually created entry.<br>• From 1 to 10, how easy and intuitive was the process?<br>• How could this process be improved? |
| 11 | Upload non invoice | Take a photo of something that is not an invoice and upload it to get it scanned.<br>• Is the information displayed understandable?<br>• From 1 to 10, how easy and intuitive was the process?<br>• How could this process be improved? |
| 12 | Information | Find information about the last mobile carrier invoice.<br>• Who was the biller?<br>• How much was the invoice?<br>• What was the date of the invoice?<br>• What is the invoice number?<br>• From 1 to 10, how easy and intuitive was the process?<br>• How could this process be improved? |

Continued on next page

Table 12.2 – continued from previous page

| Nr. | What | Description |
|---|---|---|
| 13 | Accessibility | Answer the following questions.<br>• Was the text size easy to read accross the app?<br>• Was the contrast good accross the app?<br>• Were all buttons and descriptions clear and helpful?<br>• Was it easy to navigate inside the app? |
| 14 | Performance | Answer the following questions.<br>• Did the app load quickly when opening the app or switching between pages?<br>• Did you notice any delays or lags when performing certain actions?<br>• From 1 to 10, how do you rate the speed of the app and computations (e.g. uploading invoice)? |
| 15 | General feedback | Answer the following questions.<br>• What would you change/improve?<br>• What additional functionality would you like to have?<br>• Overall would you say the app is easy to use? |

Table 12.3: Usability Testing Results: Beta Release

| Nr. | Test 1 | Test 2 |
|---|---|---|
| 1 | • Use case identifiable in title<br>• Simple, functional<br>• Yes, the plus button | • Yes, something with invoices and scanning<br>• Plain, simple, straightforward<br>• Click on the plus |

Continued on next page

Table 12.3 – continued from previous page

| Nr. | Test 1 | Test 2 |
|---|---|---|
| 2 | • Easy to understand<br>• Yes<br>• 10<br>• - | • Unclear what upload does but steps are understandable<br>• Yes, take photo, choose a photo or a document<br>• 9<br>• "Document" text is on two lines, initially unclear what happens next |
| 3 | • Yes, the uploaded file is being processed<br>• Yes<br>• 9<br>• Entries are not sorted in creation order | • It's being processed somehow<br>• Yes, perfect<br>• 8<br>• Automatic refresh/tooltip, placeholder when value not found, entries are not sorted in order |
| 12 | • All information found<br>• 10<br>• - | • All information found<br>• 10<br>• Search function or similar |
| 15 | • Pay-by date instead of issue date<br>• Detail page with more information<br>• Overall: App is easy to use | • "Open" state unclear at this stage, delete action had to be shown<br>• More information about the invoice<br>• Overall: App is easy to use |

Table 12.4: Usability Testing Results: MVP Release

| Nr. | Test 1 | Test 2 |
|---|---|---|
| 1 | • Yes, in title<br>• Simple, functional<br>• Yes, the plus button<br>• Small ”i” in title is weird | • Yes, clear from title<br>• Nice theme, clean<br>• Click on the plus (Hinted) |
| 2 | • Easy to understand<br>• Yes<br>• 10<br>• - | • Very nice, easy<br>• Yes, but what files are allowed?<br>• 10<br>• Put camera buttons inside the camera view and make it fullscreen |
| 3 | • Yes, the uploaded file is being processed<br>• Yes<br>• 9<br>• Menu button does nothing | • Yes, very clear and nice animation<br>• Most important things are shown<br>• 10<br>• Filter or sort functionality |
| 4 | • Yes the fields are relevant<br>• Pay-by or due date<br>• 10<br>• - | • All the fields are important<br>• Not really<br>• 10<br>• - |
| 5 | • 10<br>• - | • 10<br>• - |
| 6 | • 10<br>• - | • 9<br>• Ask if changes want to be saved on back navigation without saving |

Continued on next page

Table 12.4 – continued from previous page

| Nr. | Test 1 | Test 2 |
|---|---|---|
| 7 | • 10<br>• - | • 10<br>• - |
| 8 | • 10<br>• Swipe to delete function not really visible | • 9<br>• Took a few seconds, wasn't aware of both delete methods |
| 9 | • 9<br>• Currency label is on two lines, big numbers get converted to scientific notation | • 9<br>• Currency field wasn't formatted nicely |
| 10 | • 10<br>• - | • 9<br>• Add option for image gallery |
| 11 | • Yes, its clear something went wrong<br>• 9<br>• Popup dialog needs bigger margin for content | • Yes, nice popup with good information<br>• 9<br>• Add placeholders instead of empty fields |
| 12 | • All information found<br>• 10<br>• - | • All information found<br>• 10<br>• - |
| 13 | • Yes, text was readable<br>• Yes, contrast was good<br>• Yes, button functionality was clear<br>• Yes, navigation was easy | • Yes<br>• Yes, dark and light theme had good contrast<br>• Yes, except menu button in Overview screen<br>• Yes |

Continued on next page

Table 12.4 – continued from previous page

| Nr. | Test 1 | Test 2 |
|---|---|---|
| 14 | • Yes, the app loads quickly<br>• Camera takes a second to take a picture<br>• 9 | • Loading times weren't too long<br>• Only a tiny bit when opening invoice file<br>• 9 |
| 15 | • Pay-by date would be nice, title "i" is small<br>• -<br>• Overall: App is easy to use | • Unnecessarily big gaps in Detail screen at the top<br>• More functionality in Overview page (filter, sort)<br>• Overall: App is easy to use |

# Designs

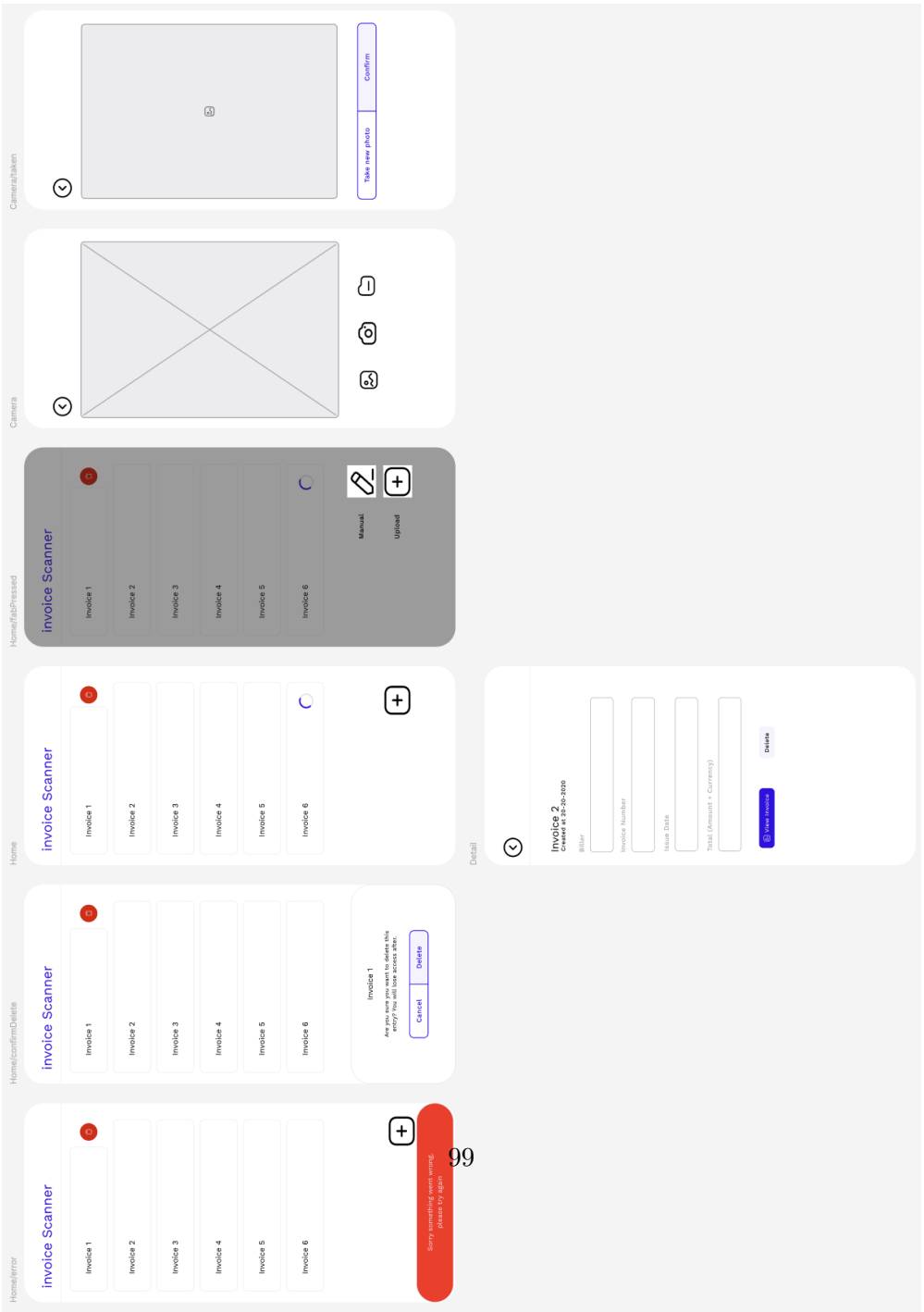


Figure 12.1: Low-Fidelity Designs

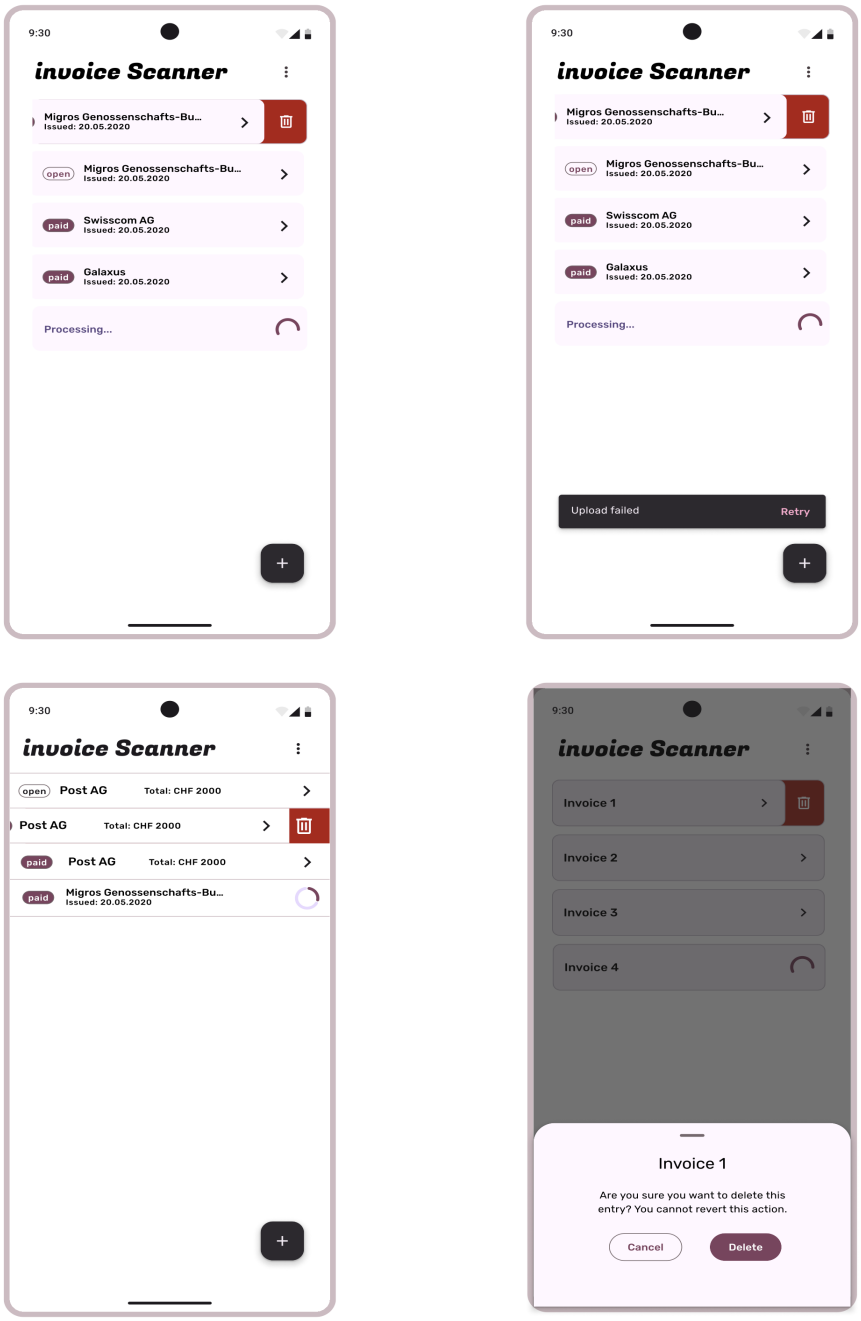


Figure 12.2: High-Fidelity Designs - Light 1

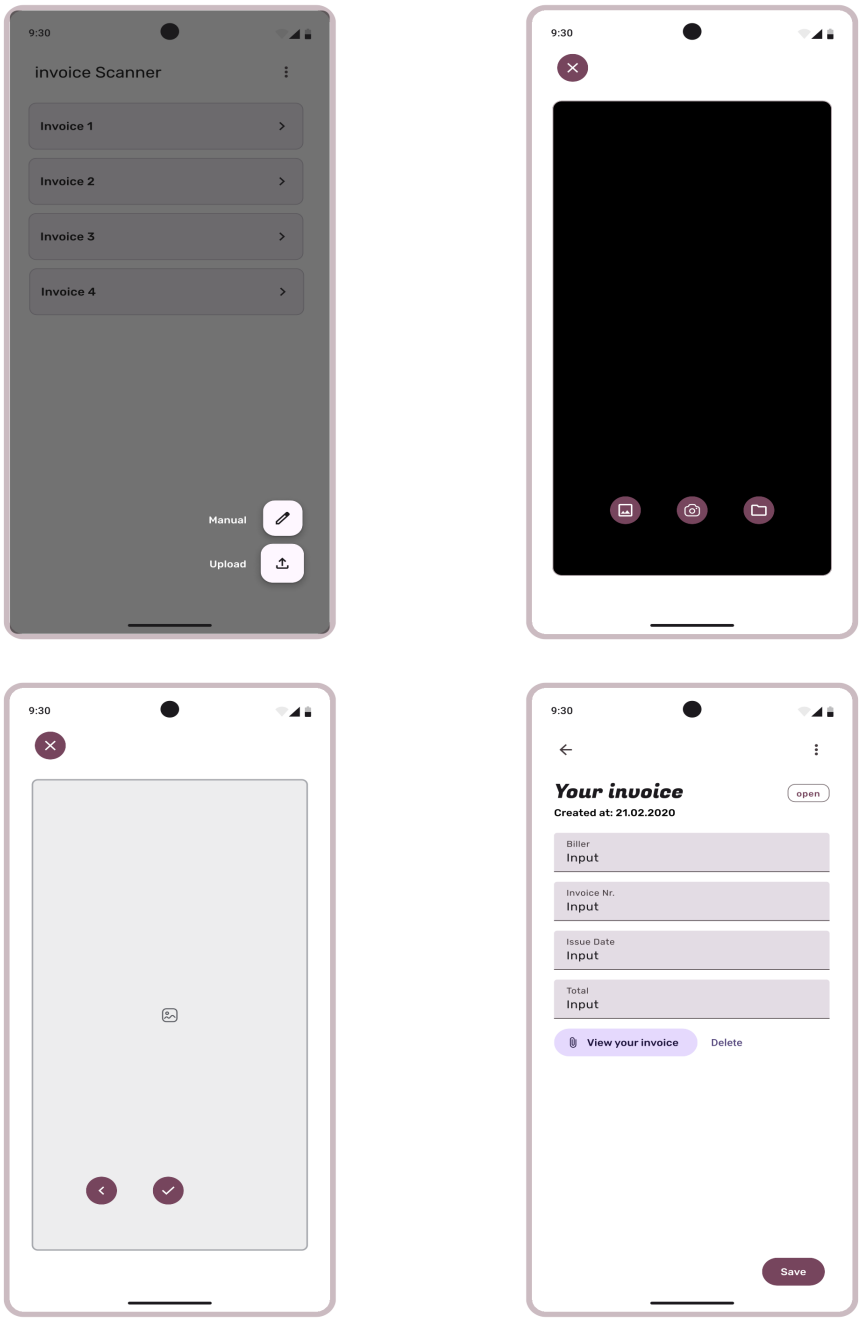


Figure 12.3: High-Fidelity Designs - Light 2

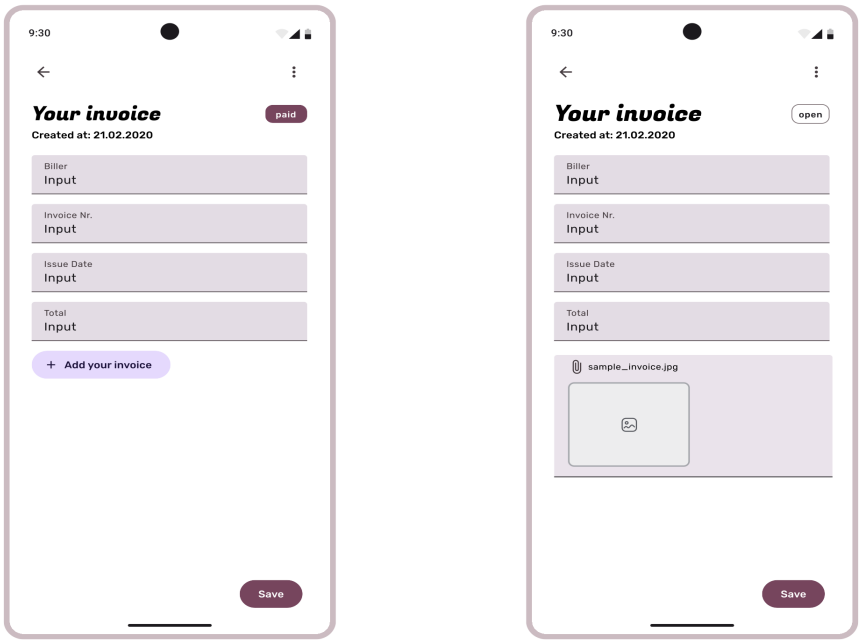


Figure 12.4: High-Fidelity Designs - Light 3

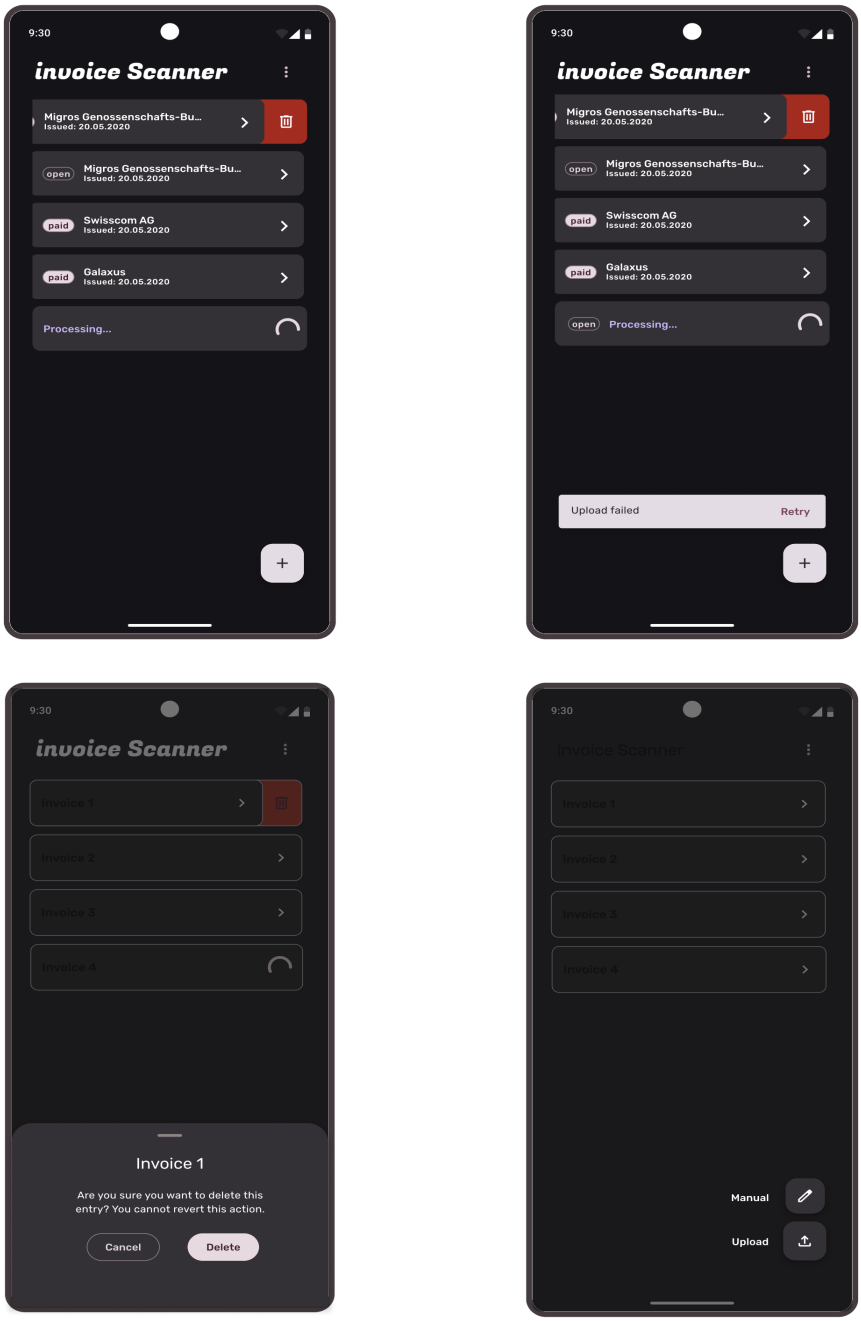


Figure 12.5: High-Fidelity Designs - Dark 1

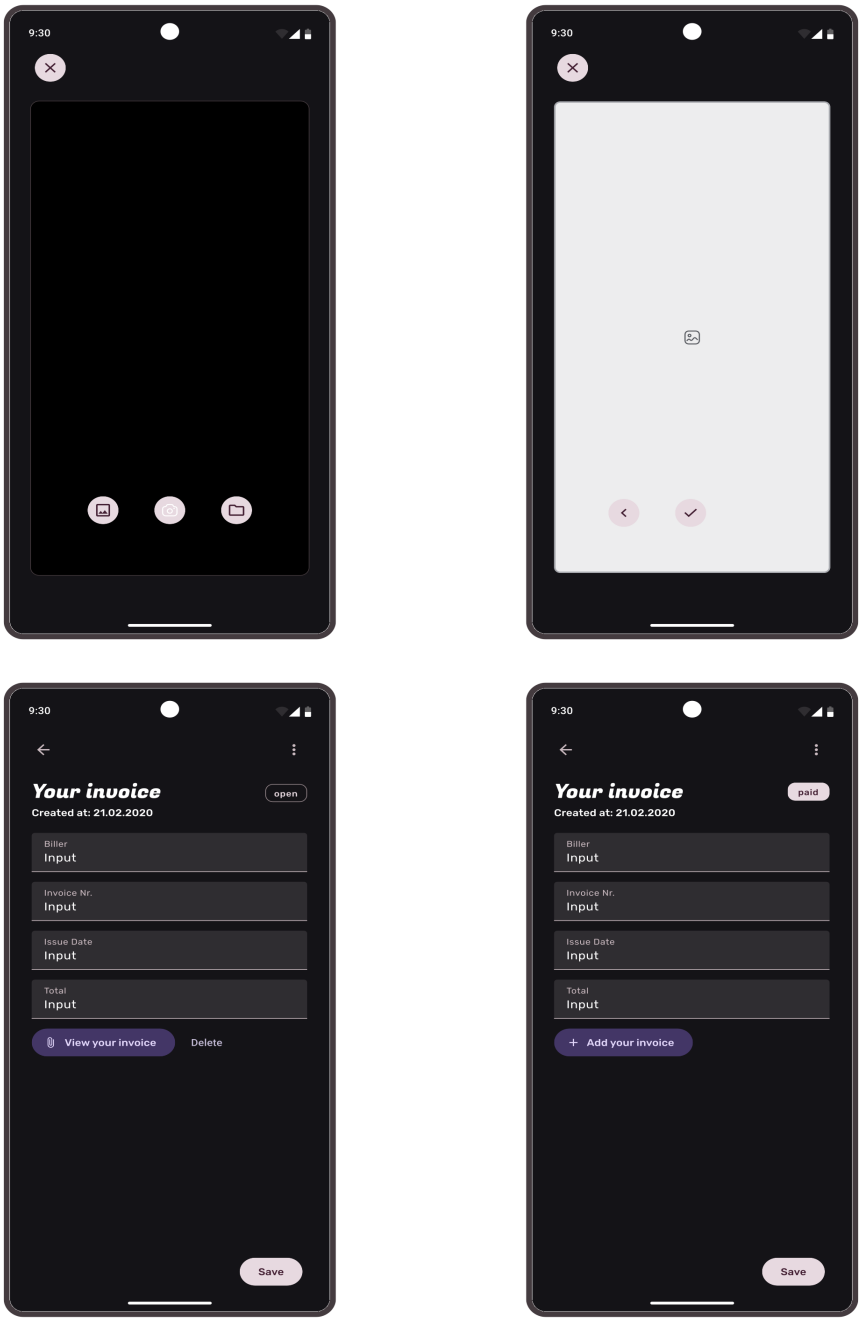


Figure 12.6: High-Fidelity Designs - Dark 2

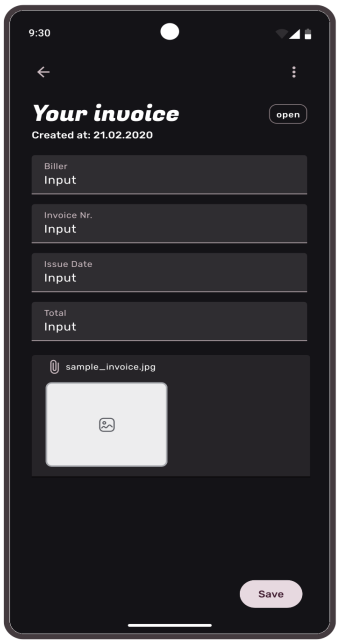


Figure 12.7: High-Fidelity Designs - Dark 3

# Meeting Minutes

The agendas and decisions of all the weekly meetings with the advisor Martin Seelhofer are summarised and noted here. For the full insight into the meeting notes check the Miro board. It is the primary medium to keep track of all meeting related things. Other meetings such as Sprint Planning etc. are not kept track off, except when big decisions are made.

## 19.09.2024

### Kick-Off Meeting

**Attendees:** Martin Seelhofer, Roger Marty, Tseten Emjee

**Agenda:**

- First meeting and introduction
- Clarification of the task

**Decisions:**

- We decided to focus on the scanning of invoices for the MVP. However, after initial research processing receipts should also be possible.
- Relevant fields for the MVP are invoice recipient, invoice number, date and a total amount. Other fields are outside the MVP scope.
- Deployment to the Google Play Store will not be implemented.
- Additional functionalities can be implemented but will be outside the MVP scope.

## 23.09.2024

### Weekly 1

**Attendees:** Martin Seelhofer, Roger Marty, Tseten Emjee

**Agenda:**

- Progress Update
- Questions

**Decisions:**

- Review and reevaluate risks after prototype / proof-of-concept.
- No regular submission of the documentation needed for advisor.

## 30.09.2024

### Weekly 2

**Skipped:** Skipped due to schedule conflicts.

## 07.10.2024

### Weekly 3

**Attendees:** Martin Seelhofer, Roger Marty, Tseten Emjee

**Agenda:**

- Progress Update
- Questions

**Decisions:**

- No release APK/AAB needed, but provide instructions for advisor to use app locally.
- Sequence diagrams can be high-level initially for planning. After implementation is complete there should be exemplary sequence diagrams of key use cases that reflect code closely.

## 14.10.2024

### Weekly 4

**Attendees:** Martin Seelhofer, Roger Marty, Tseten Emjee

**Agenda:**

- Progress Update
- Questions

**Decisions:**

- Component interaction diagrams will be used for the high-level sequence diagrams.
- The detail view of an invoice entry will have the option to CRUD the attached invoice and it will show a preview. However, the preview will have low priority.

### Sprint Planning

**Attendees:** Roger Marty, Tseten Emjee

**Decisions:**

- New field `isPaid` for `Invoice` entity domain model to define if an invoice has been paid or not which further extends the invoice management functionality.

## 21.10.2024

### Weekly 5

**Attendees:** Martin Seelhofer, Roger Marty, Tseten Emjee

**Agenda:**

- Progress Update
- Questions

**Decisions:**

- Limit file size of invoice uploads to 20MB, alternatively compress images before upload.
- Limit file type of invoice uploads to PNG, JPG and PDF.
- New field `fcmToken` for `User` entity in domain model for Firebase token.

## 28.10.2024

### Weekly 6

**Attendees:** Martin Seelhofer, Roger Marty, Tseten Emjee

**Agenda:**

- Progress Update
- Questions

**Decisions:**

- CID for the main use case is sufficient. More detailed sequence diagrams could cover more if needed.
- For additional query flexibility a secondary index will be created in the DynamoDB database.

## 04.11.2024

### Weekly 7

**Attendees:** Martin Seelhofer, Roger Marty, Tseten Emjee

**Agenda:**

- Progress Update
- Questions

**Decisions:**

- Research Notification Channels for Firebase notifications (Low priority).
- Return values for endpoints discussed and defined.

## 11.11.2024

### Weekly 8

**Attendees:** Martin Seelhofer, Roger Marty, Tseten Emjee

**Agenda:**

- Progress Update
- Questions

**Decisions:**

- Re-evaluation of risks and NFRs together with release of Beta.

## 18.11.2024

### Weekly 9

**Attendees:** Martin Seelhofer, Roger Marty, Tseten Emjee

**Agenda:**

- Progress Update
- Questions

**Decisions:**

- Of all optional features, authentication has the highest priority and will be the focus once everything else is done.
- A service like AWS Cognito or Firebase Authentication will be used.

## 25.11.2024

### Weekly 10

**Attendees:** Martin Seelhofer, Roger Marty, Tseten Emjee

**Agenda:**

- Progress Update
- Presentation of beta release
- Questions

**Decisions:**

- Use Javadoc or Docstrings for code that is not self explanatory.
- Adjustment of certain NFRs acceptable because they were not defined concise enough.

## 02.12.2024

### Weekly 11

**Attendees:** Martin Seelhofer, Roger Marty, Tseten Emjee

**Agenda:**

- Progress Update
- Questions

**Decisions:**

- Re-evaluated optional features and decided to use the remaining time to start with the authentication because all other FRs have been completed.

## 09.12.2024

### Weekly 12

**Attendees:** Martin Seelhofer, Roger Marty, Tseten Emjee

**Agenda:**

- Progress Update
- Presentation of MVP release
- Questions

**Decisions:**

- An A0 poster is not required.
- No important decisions have been made. Focus on MVP release and demonstration.

## 16.12.2024

### Weekly 13

**Attendees:** Martin Seelhofer, Roger Marty, Tseten Emjee

**Agenda:**

- Progress Update
- Submission Details
- Questions

**Decisions:**

- AI/LLMs declaration in documentation
- No need for complete sequence diagram of all systems