OST
Eastern Switzerland
University of Applied Sciences

Digitaler Briefversand

Autumn Term 2023

Department of Computer Science

OST - University of Applied Sciences

Campus Rapperswil-Jona

Author(s)	Gian-Luca Vogel & Marc Kissling & Andrew Willi
Advisor	Frank Koch
Project Partner	Michael Güntensperger

December 22, 2023

Abstract

Introduction

Communication in the modern business world is characterized by remarkable complexity. In addition to conventional communication methods, such as postal mail, innovative approaches have emerged that businesses can leverage. Email and messenger services like Microsoft Teams or WhatsApp have become widely adopted communication channels. The clear trend indicates a significant preference for digital communication forms. The trend shows a clear preference for digital forms of communication, which is at the expense of the still important, paper-based postal correspondence. In this work, analogue postal traffic is to be digitally enriched in order to combine the advantages of both worlds.

Objective

Within the scope of this project, a prototype of an application is being developed with the aim of digitizing postal mail. The primary goal is to create a user-friendly application that makes it easier for the sender to digitally send a letter to a recipient. If the digital route is not feasible, the application will automatically send the letter through traditional means. The prototype is designed to be easily expandable to facilitate the continuous development and implementation of new features. Another crucial aspect of the application is its broad device compatibility to ensure maximum user reach.

Conclusion

In the creation of the prototype for this project, we utilized the following technologies: Node.js and TypeScript were employed for the backend. Electron, in conjunction with React, also in TypeScript, was used for the desktop application. For the mobile application, we opted for Flutter. The application allows an organization to digitally send a letter to a customer via the desktop application, who then receives it through the mobile app. The application autonomously decides whether the letter should be digitally sent to the recipient after verifying their address or if it should be physically sent if verification is not successful. In further development, the desktop application is planned to be enhanced with automatic address recognition, and the manual handling of physical mail delivery will be improved as needed.

Management Summary

Introduction

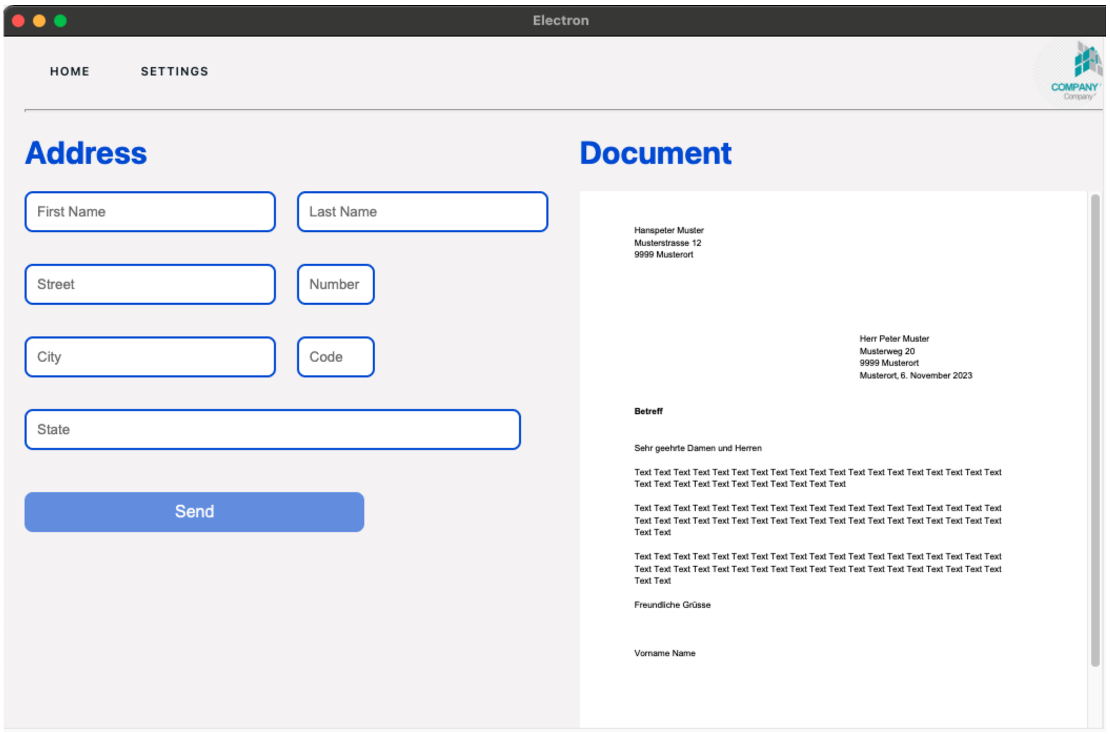
The rise of digital channels such as email and messaging services has altered communication in the modern working environment. By creating a prototype application that digitises postal mail and combines the benefits of digital communication with the traditional postal system, this project responds to the rapidly changing environment. Especially for small companies it would be a great tool to streamline their communication.

Collaboration and Team Dynamics

Our team, consisting of three dedicated members, adopted a technology driven approach to the project implementation. We divided the project workload into desktop, mobile and backend. Motivated by a common goal, our group worked incredibly well together to complete the project. The dynamic interaction of the team members' skills and knowledge enabled a smooth workflow that fostered innovation and creativity. In order to develop the frontend in parallel, we were able to start working on the backend early on. This allowed us to quickly identify and fix any problems that arose during the development process.

Achievements

Throughout the development process we successfully achieved several milestones. We implemented all the functional requirements, and due to time constraints we were only able to partially implement two optional requirements. All but three of the non-functional requirements have now been implemented. And here is a result of the desktop application:



Hanspeter Muster
Musterstrasse 12
9999 Musterort

Herr Peter Muster
Musterweg 20
9999 Musterort
Musterort, 6. November 2023

Betreff

Sehr geehrte Damen und Herren

Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text

Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text

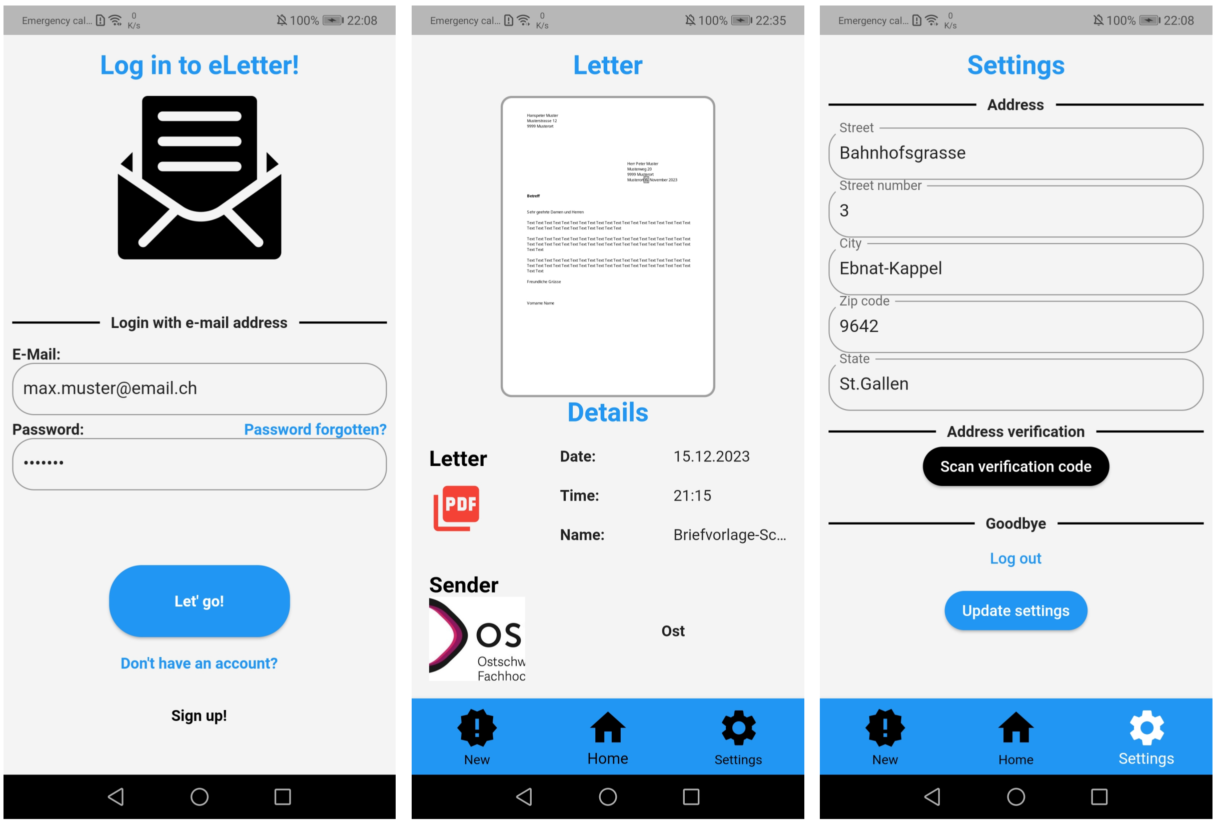
Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text

Freundliche Grüsse

Vorname Name

Figure 0.1: Desktop Application

Here are three screenshots of the mobile application:



E-Mail: max.muster@email.ch
Password: •••••••

Letter	Date:	15.12.2023
	Time:	21:15
	Name:	Briefvorlage-Sc...
Sender		Ost

Street: Bahnhofsgrasse
Street number: 3
City: Ebnat-Kappel
Zip code: 9642
State: St.Gallen

Figure 0.2: Mobile Application

Challenges and Solutions

We faced a number of challenges which were overcome during the implementation of the project:

- **New Technology Exploration:** Understanding the new Flutter framework posed an initial challenge. To overcome this, we invested time in thoroughly exploring and learning its features and capabilities.
- **Project Time Management Misunderstanding:** We thought it would be sufficient to simply track the time spent on the project. However, there is a need for estimation and issue tracking, which we solved by using a github project board and the 7pace plugin.
- **Deployment Process Complexity:** We just had to spend more time on the deployment. The desktop application was also a bit of a challenge. We were not familiar with the process of deploying an Electron application. We solved this by using the Electron builder package and creating an artefact.

Lessons learned

In retrospect, it was a good decision to start the backend early in the development process. This allowed us to quickly adapt and address any issues that arose during the development process. As everyone had their own main area with interfaces to the others, we were able to effectively contain the effects of blocking each other.
We also learned why Latex is such a powerful tool for parallel development. Despite a slight learning curve, the benefits of using LaTeX for documentation became apparent. It allowed us to collaborate on professional and structured documents, ensuring version control and consistency.

Closing words

All in all the project has been a success. We are happy with the outcome and the way we worked together. The experience of working collaboratively and communicating effectively not only enriched the project outcome but also contributed to the personal and professional growth of each team member. These lessons in teamwork and communication are invaluable takeaways that will undoubtedly influence our future collaborations.

Contents

1 Starting Position

This project piqued our interest and continues to do so. When we first looked at all the topics, this one stood out the most. We have worked with similar technologies which was our main reason why we wanted this project. Furthermore, digging deeper into Electron and Flutter gave an interesting opportunity for additional study and skill development. What further motivated us was the realization that we were actively digitizing the process.

2 Conceptual Formulation

The aim of this project is to develop a solution to bridge the gap between the user-friendliness of digital communication and the security of traditional physical mail. To this end, a software system is being created that consists of a desktop application, a mobile application and a backend. The desktop application is used to send letters in PDF format to recipients. These recipients can then receive and display the letters via the mobile application. If this is not possible, the letter is automatically delivered physically by conventional means. The entire logic of the process is handled by the backend application, which is implemented as a cloud application.

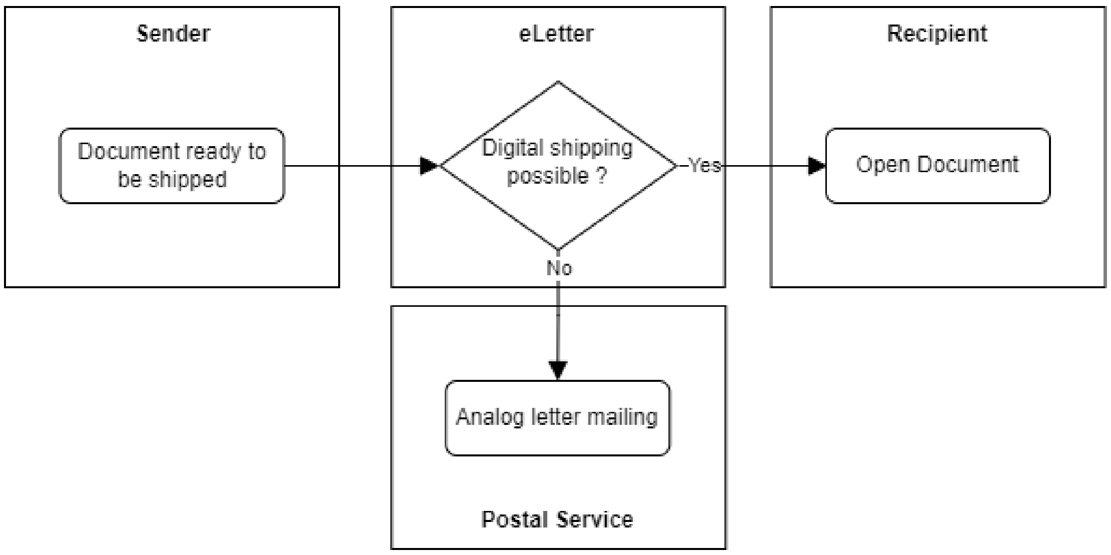


Figure 2.1: Letter sending process

The following diagram illustrates the process when a letter is sent physically. As you can see, it's a fairly simple process. The application saves the metadata and the letter in PDF format, and then sends it to the postal service. The postal service is responsible for printing and mailing the letter.

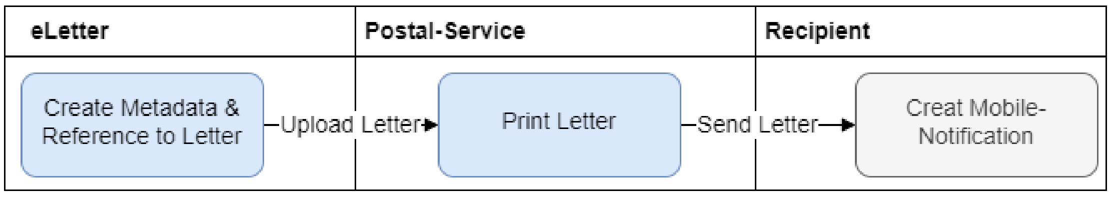


Figure 2.2: Letter sending physical

In contrast, the digital route requires more effort on the part of the application. First, the letter must be saved together with the metadata. The recipient is then notified and receives the letter's metadata via this notification. This metadata in turn contains a link that the recipient can use to download the letter.

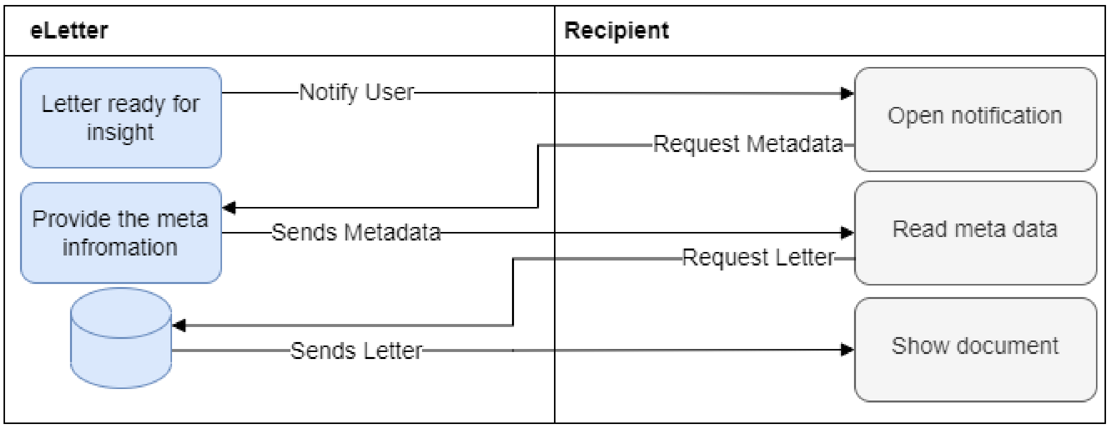


Figure 2.3: Letter sending digital

For security purposes, the user needs to confirm his address; the procedure is shown in the diagram below.

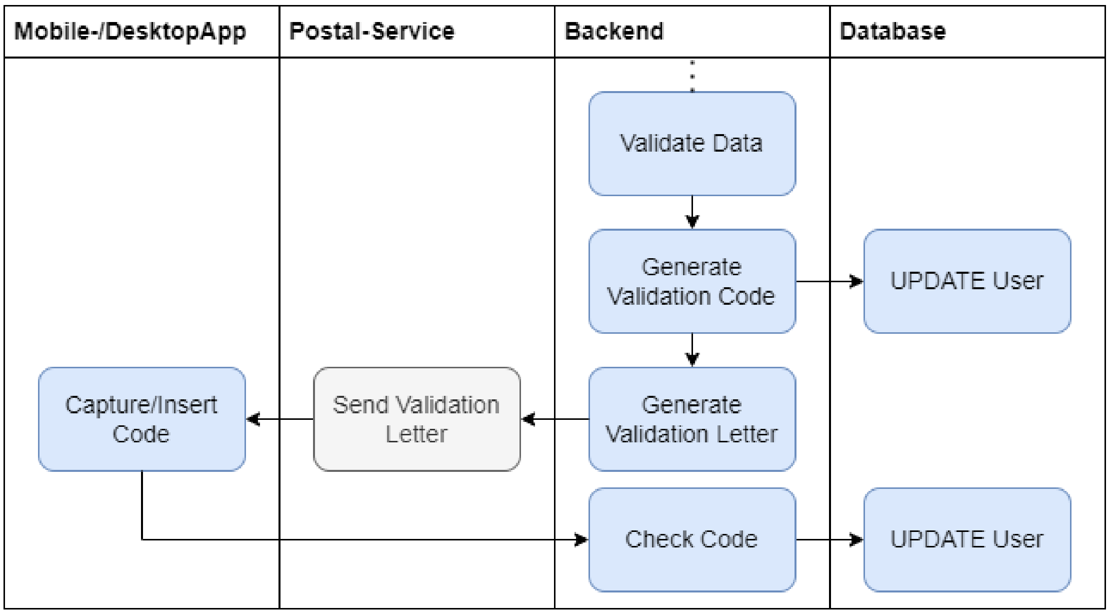


Figure 2.4: Process chart address verification

3 Requirements

To ensure agility, we track functional needs on GitHub and 7pace using user stories. (More information about 7pace in the chapter Project and Time Management under the section "*Time Tracking*".) Our primary goal is to make letter sending and receiving as simple as possible. Restricted mobile access to specific papers, user-friendly registration for new recipients, address verification by QR-Code via mail, expedited registration for companies or senders, and rapid document upload for dispatch are all key features.

We're also considering cloud deployment to improve accessibility and scalability. These criteria, which are managed via GitHub and 7pace, represent our commitment to digitising and optimising the letter exchange process.

3.1 Functional Requirements

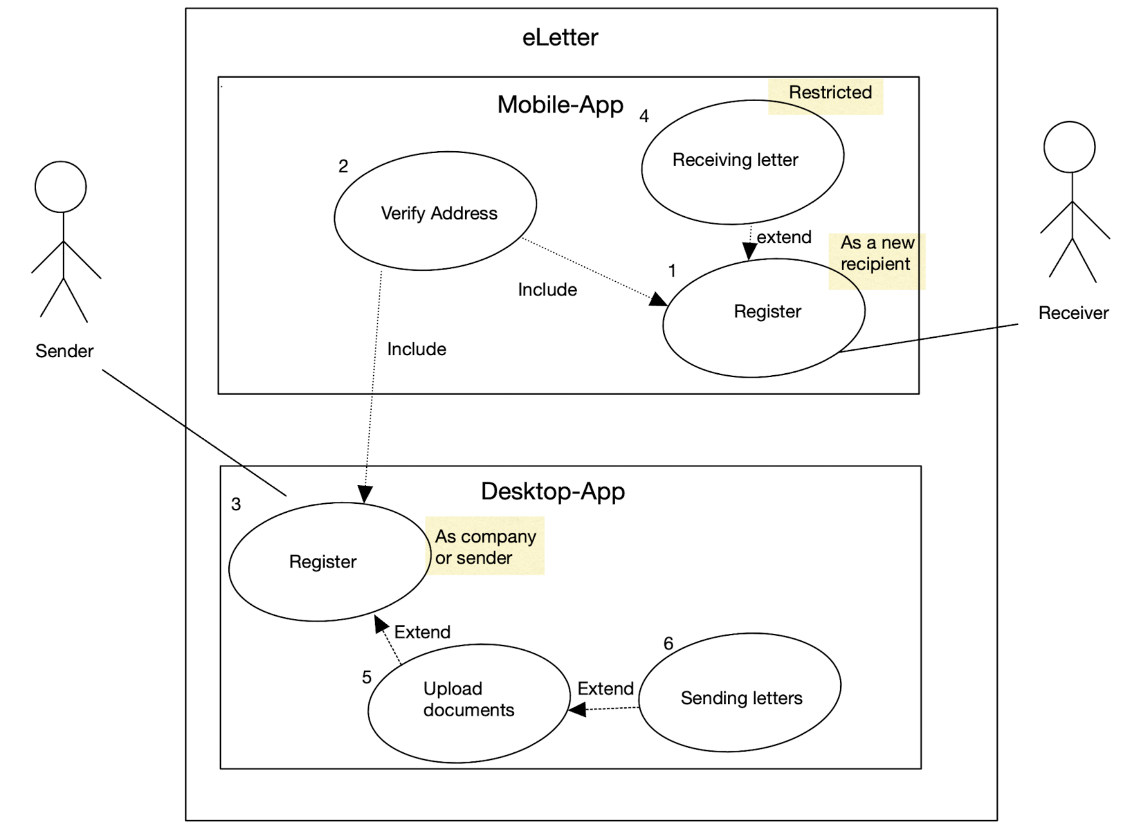


Figure 3.1: Use Case Diagram

Here we see two systems for the mobile and desktop application and two types of users:

- **Sender:** A user, who can be a company or an individual, who sends digital letters and has access to the desktop application.
- **Receiver:** A user, usually a person, who receives digital letters and has access to the mobile application.

Following the use case diagram, each will be explained in detail:

1. User wants to register as new recipients
2. User needs address verification with QR code Adress verification with QR code which will be sent through postal mail.
3. System should provide a way for companies/sender to register as new recipients.
4. Mobile application should have a feature that restricts access to a user's own documents while the desktop application has no restrictions.
5. User wants to upload document(s).
6. User wants to send letters after uploading document(s).

3.2 Non-Functional Requirements

Our Non-functional requirements are defined using FURPS+'s definition. Unless otherwise specified, all of the NFRs mentioned below are requirements.

Type	Description	Acceptance Criteria
Feature	Implement features as per agreed-upon priority.	Features implemented in the agreed priority order.
Performance	Backend handles 1000 requests per minute.	Backend maintains performance with 1000 requests per minute.
Performance	Pages load within 200ms.	All pages load within 200ms for responsive user experience.
Compatibility	Desktop application runs on Windows.	Desktop application is Windows-compatible.
Usability	Application is easy to install and use.	Users can easily install and use the application.
Usability	Mobile UI should rate at least 8 out of 10 by three out of four test users (layout, responsiveness, color, content).	Chapter Test Concept
Performance	Database manages up to 10,000 documents/PDFs without issues.	Tested with Performance Testing Framework.
Reliability	Errors display messages, revert the system, and avoid system failures.	Errors show messages, restore the system, and prevent failures.
Logging	Every error is logged for monitoring and troubleshooting.	All errors are logged for monitoring and troubleshooting.
Security	Encryption with SSL certificates for frontend-backend communication.	SSL certificates secure frontend-backend communication.
Security	Input validation and no SQL injection vulnerabilities.	Input data is validated, and no SQL injection vulnerabilities exist.
Security	User passwords securely hashed in the database.	User passwords hashed securely, not stored in plain text.
Flexibility	Modular backend logic for easy expansion.	Backend logic designed modularly for easy expansion.
Testing	Comprehensive backend API testing with appropriate tools.	Backend API thoroughly tested with suitable tools, ensuring functionality and reliability.
Deployment	Database, backend, and frontend deployed on separate instances for isolation and scalability.	CI/CD
Deployment	Deployment inside the Cloud	CI/CD

Table 3.2: Non-Functional Requirements

Additional information: In our opinion, the functional requirement "Deployment inside the Cloud" is a Non-Functional Requirement, especially the backend.

3.3 Optional Requirements

These optional features address various aspects of document management, user interaction and overall system customisation. The table below outlines these optional requirements:

Nr	Description
1	Automatic address recognition in PDF.
2	Categorization of documents in the mobile application.
3	Selection of shipping method in the desktop application.
4	Cost display for shipping via the desktop application.
5	Dashboard for managing employees for letter shipment, including permissions.
6	Company branding for the desktop app.
7	Web application for recipients.

Table 3.3: Optional Requirements

4 Architecture and Design

4.1 Visualizing the Architecture

Based on the Conceptual Formulation and the Requirements, we have created the architectures and designs of the application described in the following chapters. To illustrate these architectures, we have chosen to adopt the C4-Model standard. It is worth mentioning that we implemented only the first three levels of the model, as the fourth level is too detailed and would not provide any added value to the reader.

System Context (C4 Model Level 1)

The diagram illustrates how the sender and the recipient interact with the eLetter application. The eLetter application, in turn, interacts with two additional software systems. One is a Notification Service required for sending push notifications to users on mobile devices, and the other is a Postal Service necessary for physically sending letters in a traditional manner.

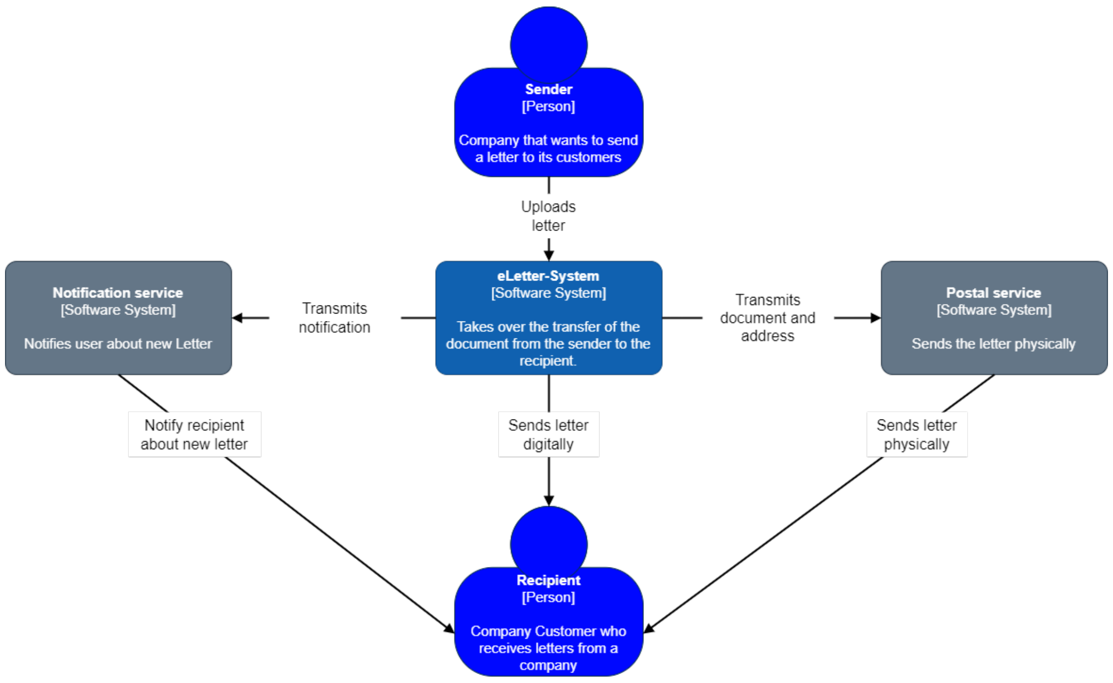

Sender
[Person]
Company that wants to send a letter to its customers
Uploads letter
eLetter-System
[Software System]
Takes over the transfer of the document from the sender to the recipient.
Transmits notification
Notification service
[Software System]
Notifies user about new Letter
Transmits document and address
Postal service
[Software System]
Sends the letter physically
Notify recipient about new letter
Sends letter digitally
Sends letter physically
Recipient
[Person]
Company Customer who receives letters from a company

Figure 4.1: C4 Model Level 1

Container Diagram (C4 Model Level 2)

The focus in the following diagram is to provide a more detailed explanation of the eLetter application. As depicted, it consists of five components: a desktop application enabling users to send letters, a mobile application facilitating letter reception, the backend serving as the central hub for the entire application logic, along with a database storing metadata, and an object storage handling the storage of letters as PDFs.

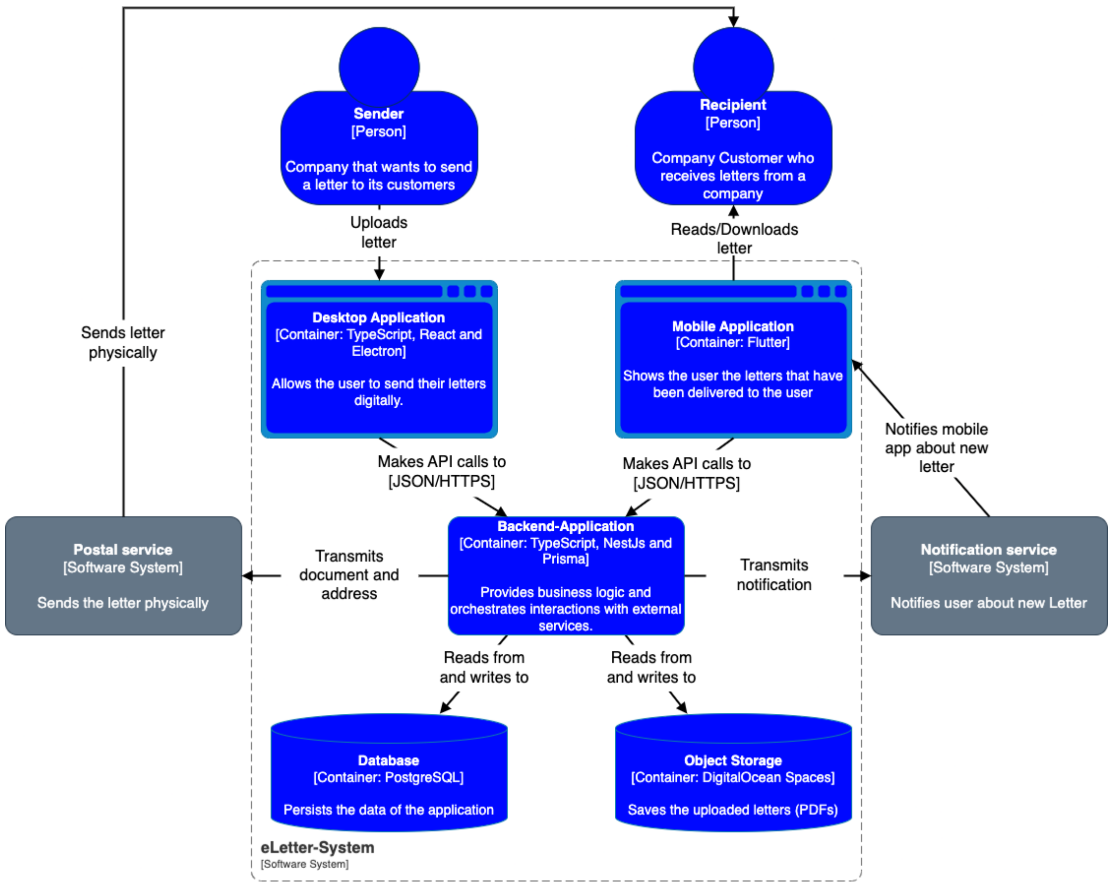

Sender
[Person]
Company that wants to send a letter to its customers
Recipient
[Person]
Company Customer who receives letters from a company
Uploads letter
Reads/Downloads letter
Sends letter physically
Desktop Application
[Container: TypeScript, React and Electron]
Allows the user to send their letters digitally.
Mobile Application
[Container: Flutter]
Shows the user the letters that have been delivered to the user
Notifies mobile app about new letter
Makes API calls to [JSON/HTTPS]
Makes API calls to [JSON/HTTPS]
Postal service
[Software System]
Sends the letter physically
Transmits document and address
Backend-Application
[Container: TypeScript, NestJs and Prisma]
Provides business logic and orchestrates interactions with external services.
Transmits notification
Notification service
[Software System]
Notifies user about new Letter
Reads from and writes to
Reads from and writes to
Database
[Container: PostgreSQL]
Persists the data of the application
Object Storage
[Container: DigitalOcean Spaces]
Saves the uploaded letters (PDFs)
eLetter-System
[Software System]

Figure 4.2: C4 Model Level 2

4.2 Architecture in Detail

In this section we examine the specific components and their interrelationships that make up the design of our system. The preceding C4 model provides the third level for a more in-depth examination of critical issues.

4.2.1 Desktop Application

Component diagram (C4 Model Level 3)

Since the desktop application is designed to be straightforward, we opted for a simple architecture. In this setup, the user interacts with the application through a view, which, in turn, communicates with the backend via REST services. This interaction is illustrated in the diagram below.

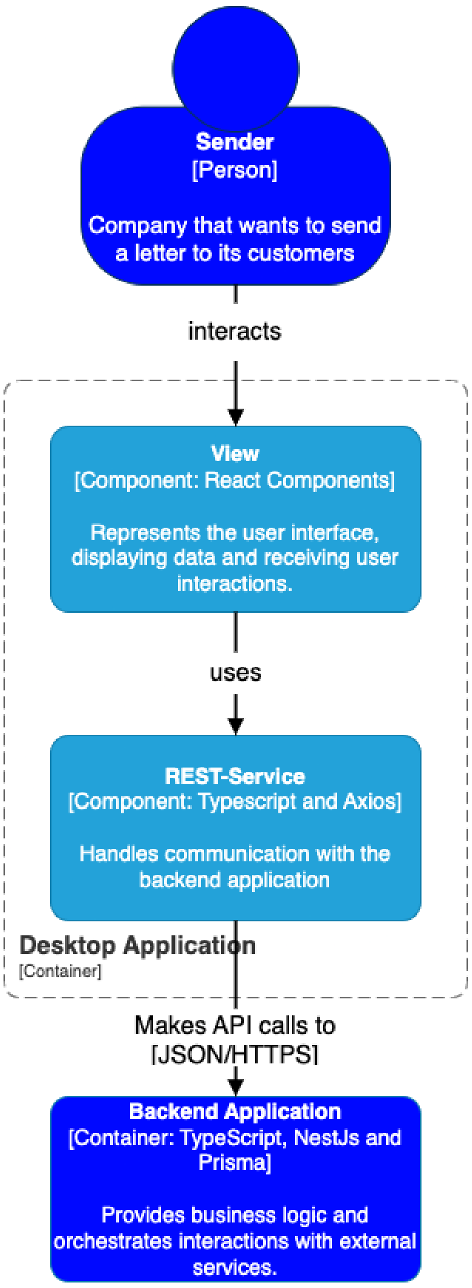

Sender
[Person]
Company that wants to send a letter to its customers
interacts
View
[Component: React Components]
Represents the user interface, displaying data and receiving user interactions.
uses
REST-Service
[Component: Typescript and Axios]
Handles communication with the backend application
Desktop Application
[Container]
Makes API calls to [JSON/HTTPS]
Backend Application
[Container: TypeScript, NestJs and Prisma]
Provides business logic and orchestrates interactions with external services.

Figure 4.3: C4 Model Level 3 Desktop Application

UI Mock-ups

The next chapter contains the initial drafts of the desktop application's mock-up, inspired by Microsoft Outlook and Apple Mail to provide a familiar user experience from the beginning.

The first two images display the welcome page presented to the user when they open the desktop app. The second image shows the page the user navigates to when clicking on "Sign up."

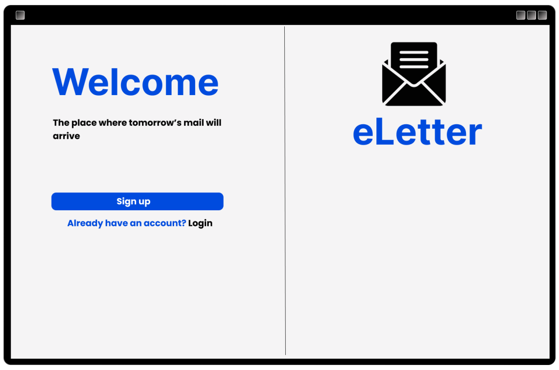

Welcome
The place where tomorrow's mail will arrive
Sign up
Already have an account? Login
eLetter

Figure 4.4: Welcome Page

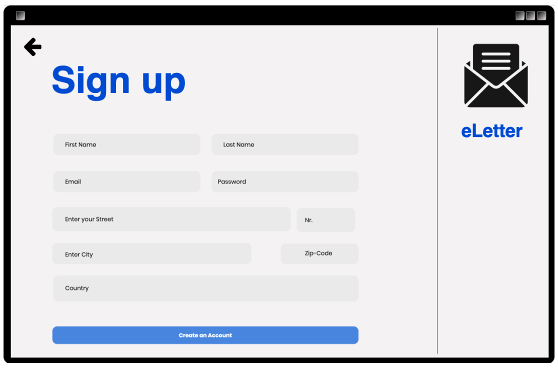


Figure 4.5: Sign up page

The next two images showcase the login page and the settings page, where the user can configure profile settings.

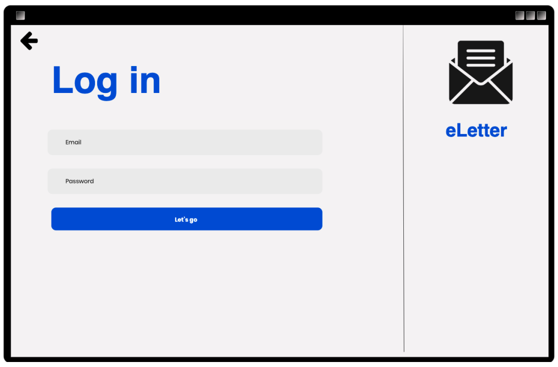


Figure 4.6: Log in page

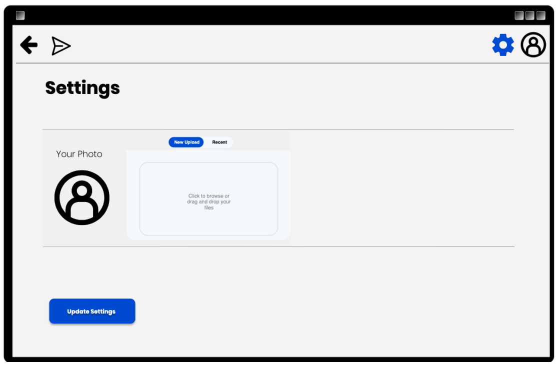


Figure 4.7: Letters page

The final two images show the application's necessary pages for sending a letter. The first image presents the upload page, where users can drag and drop a letter for upload.
The second page previews the uploaded letter and requires users to enter the address details before submission.

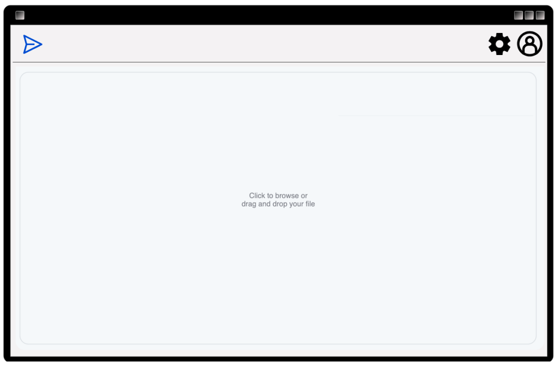


Figure 4.8: Sending and uploading page 1

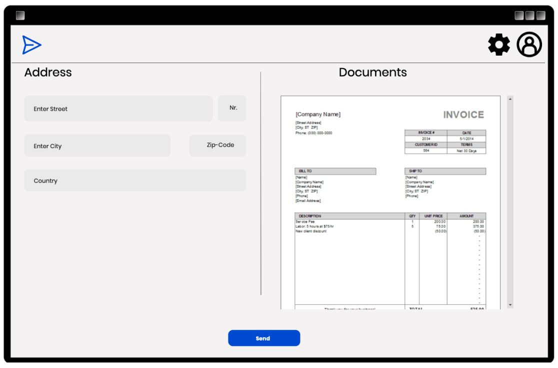


Figure 4.9: Sending and uploading page 2

4.2.2 Mobile Application

Component diagram (C4 Model Level 3)

The following diagram shows that the mobile application contains slightly more UI logic than the desktop app. This is due to the fact that, in addition to loading data from the backend, it is also necessary to process notifications that are sent to the mobile application when the user receives a new letter. For this reason, we have decided to abstract the view from the logic using a ViewModel (MVVM pattern). This offers the advantage that the view only contains the pure display logic, while the ViewModel handles state management and data loading.

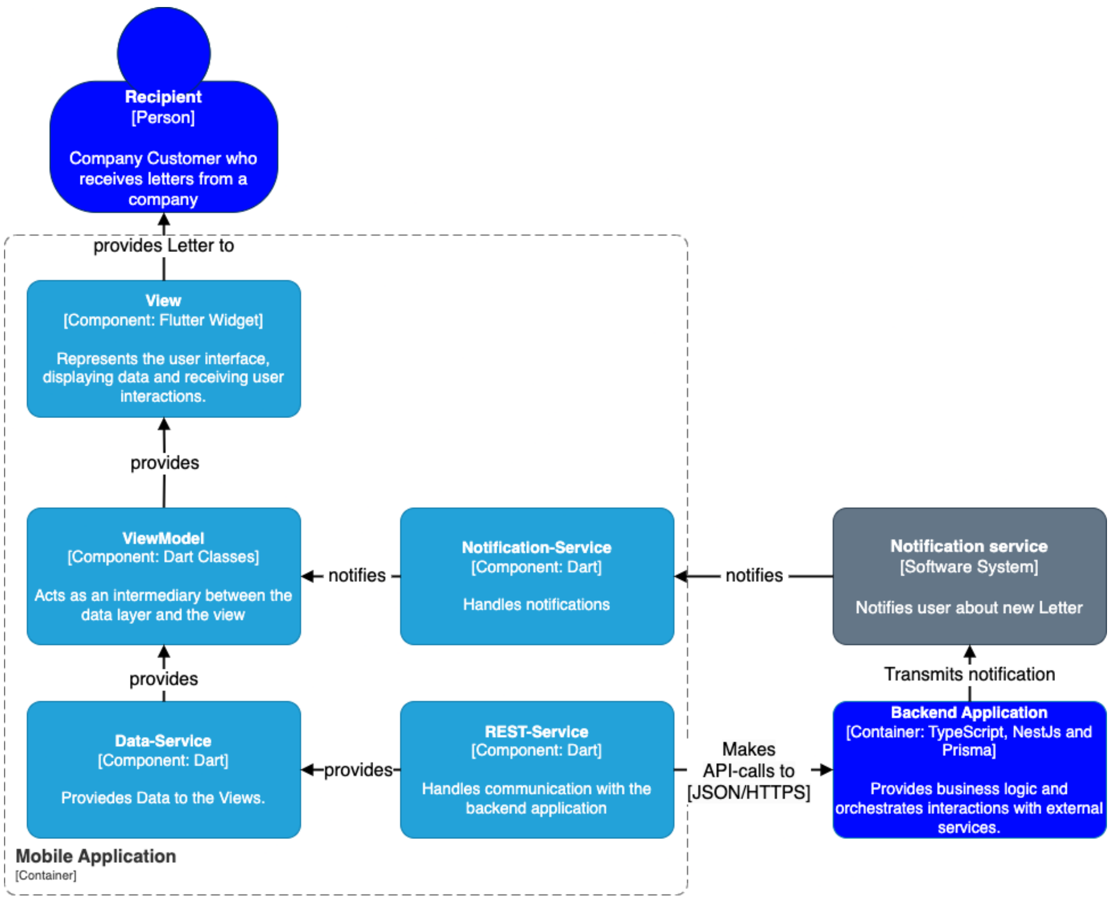

Recipient
[Person]
Company Customer who receives letters from a company
provides Letter to
View
[Component: Flutter Widget]
Represents the user interface, displaying data and receiving user interactions.
provides
ViewModel
[Component: Dart Classes]
Acts as an intermediary between the data layer and the view
notifies
Notification-Service
[Component: Dart]
Handles notifications
notifies
Notification service
[Software System]
Notifies user about new Letter
Transmits notification
provides
Data-Service
[Component: Dart]
Proviedes Data to the Views.
provides
REST-Service
[Component: Dart]
Handles communication with the backend application
Makes API-calls to [JSON/HTTPS]
Backend Application
[Container: TypeScript, NestJs and Prisma]
Provides business logic and orchestrates interactions with external services.
Mobile Application
[Container]

Figure 4.10: C4 Model Level 3 Mobile-Application

UI Mock-ups

The first three images show all the pages a user needs to register and log in to the app. From right to left, these are the Welcome Page, displayed when the application is launched, the Sign Up Page needed for user registration, and finally, the Login Page.

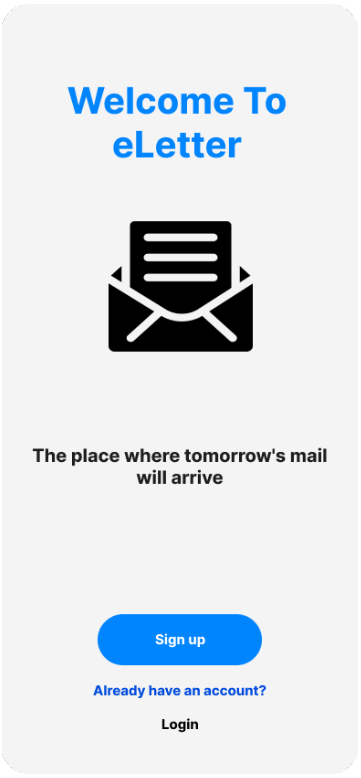


Figure 4.11: Welcome page

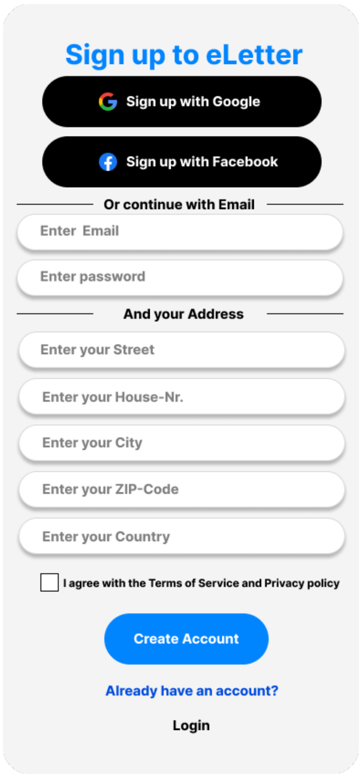


Figure 4.12: Sign up page

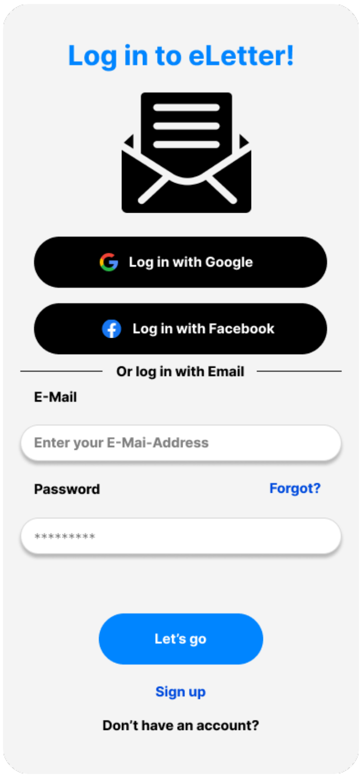
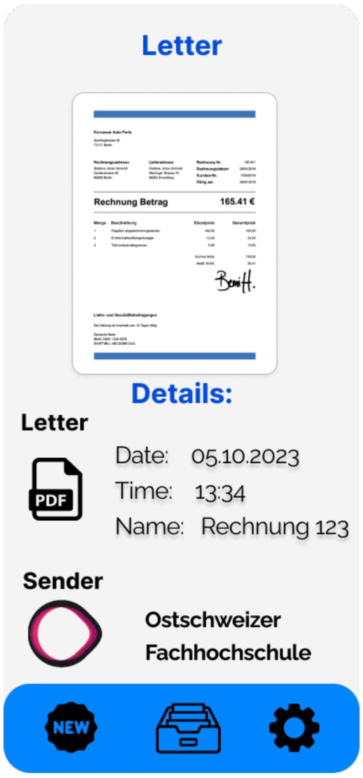


Figure 4.13: Sign in page

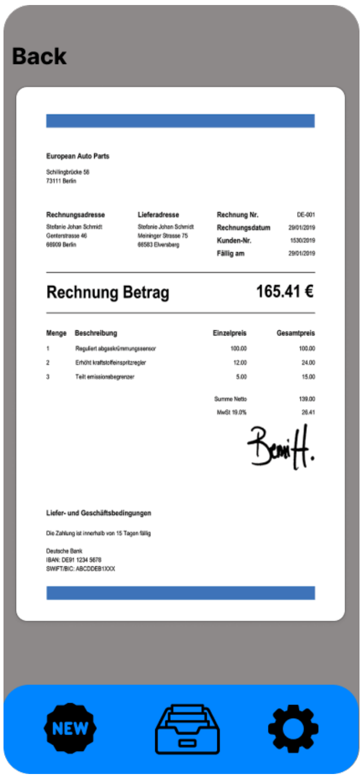
The next three pictures show the "Letter Details Page" (on the left), which is displayed when the user clicks "Open" in the mailbox. Here, the user can view all the important meta-information about the letter, and additionally, a preview of the first page is displayed.
If the user clicks on this preview, they will access the PDF view, depicted in the middle image.
The third image shows the "Settings Page," where the user can manage their information, verify their address, and log out.



Date: 05.10.2023
Time: 13:34
Name: Rechnung 123

Ostschweizer Fachhochschule

Figure 4.14: Detail page



European Auto Parts
Schillingbrücke 58
73111 Berlin

Rechnungsadresse	Lieferadresse		
Stefanie Johan Schmidt	Stefanie Johan Schmidt	Rechnung Nr.	DE-001
Genterstrasse 46	Meininger Strasse 75	Rechnungsdatum	29/01/2019
66909 Berlin	66583 Elversberg	Kunden-Nr.	1530/2019
		Fällig am	29/01/2019

Rechnung Betrag 165.41 €

Menge	Beschreibung	Einzelpreis	Gesamtpreis
1	Reguliert abgaskrümmungssensor	100.00	100.00
2	Erhöht kraftstoffeinspritzregler	12.00	24.00
3	Teilt emissionsbegrenzer	5.00	15.00
		Summe Netto	139.00
		MwSt 19.0%	26.41

[illegible]

Liefer- und Geschäftsbedingungen

Die Zahlung ist innerhalb von 15 Tagen fällig

Deutsche Bank
IBAN: DE91 1234 5678
SWIFT/BIC: ABCDDEB1XXX

Figure 4.15: PDF view page

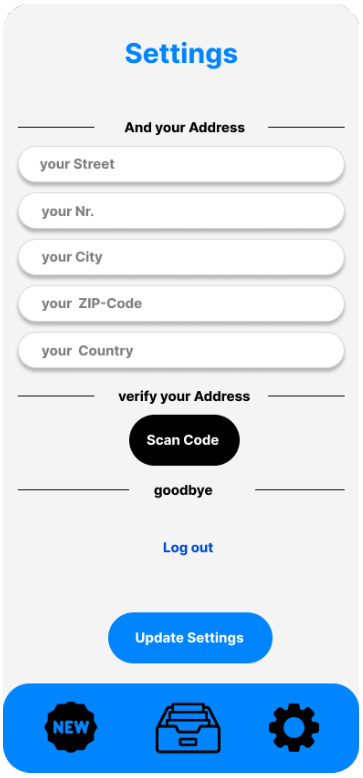


Figure 4.16: Settings page

The last two pictures show the user's mailbox, displaying all the letters they have received (on the right). The "New Mail Page" shows all the letters that the user has not yet read. Switching between these two views is done through the navigation bar at the bottom of the screen. Clicking on the middle icon takes the user to the mailbox, clicking on the far left icon takes them to the "New Mail Page," and clicking on the far right icon takes them to the "Settings Page."

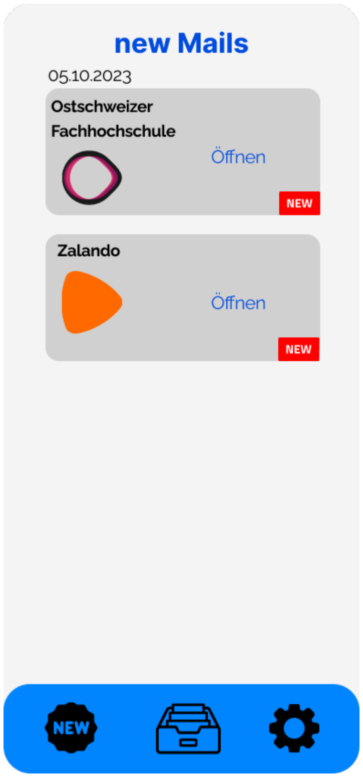


Figure 4.17: New letters

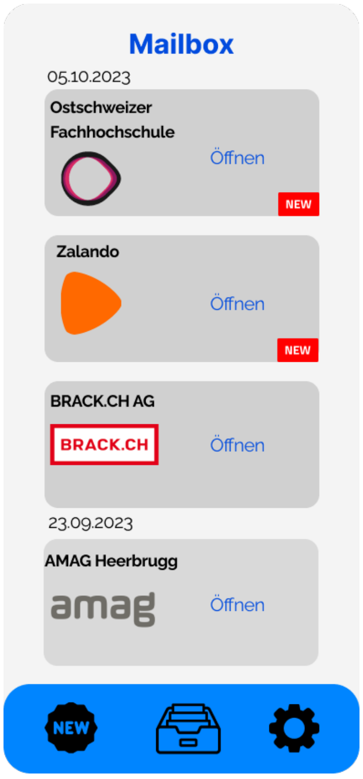


Figure 4.18: All letters page

4.2.3 Backend

Component diagram (C4 Model Level 3)

The diagram below illustrates the planned architecture for the backend. As can be seen, all required functions are encapsulated in individual modules that share services among themselves. This modular approach aims to make the code easily maintainable and extendable. Furthermore, this architecture is intended to simplify the testability of the code.

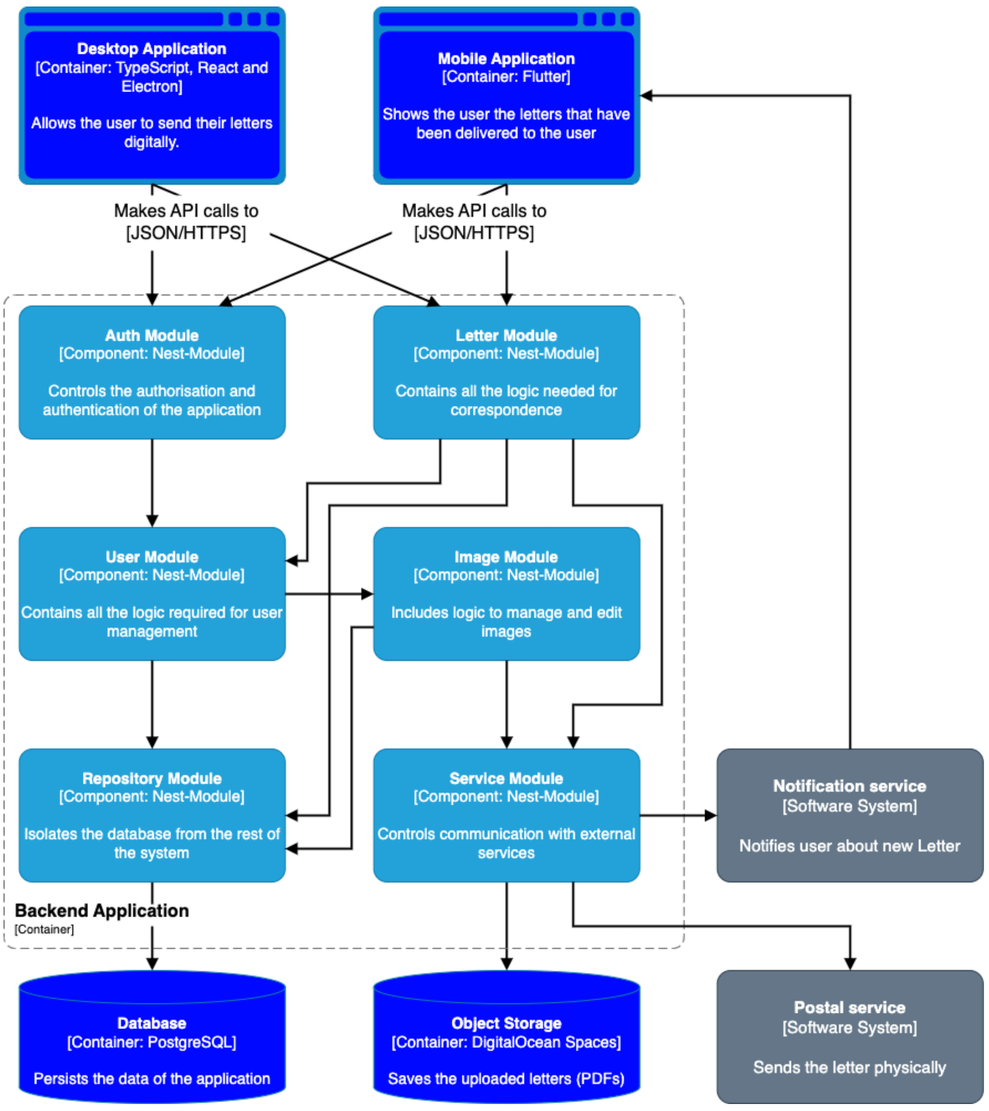

Desktop Application
[Container: TypeScript, React and Electron]
Allows the user to send their letters digitally.
Mobile Application
[Container: Flutter]
Shows the user the letters that have been delivered to the user
Makes API calls to [JSON/HTTPS]
Makes API calls to [JSON/HTTPS]
Auth Module
[Component: Nest-Module]
Controls the authorisation and authentication of the application
Letter Module
[Component: Nest-Module]
Contains all the logic needed for correspondence
User Module
[Component: Nest-Module]
Contains all the logic required for user management
Image Module
[Component: Nest-Module]
Includes logic to manage and edit images
Repository Module
[Component: Nest-Module]
Isolates the database from the rest of the system
Service Module
[Component: Nest-Module]
Controls communication with external services
Notification service
[Software System]
Notifies user about new Letter
Backend Application
[Container]
Database
[Container: PostgreSQL]
Persists the data of the application
Object Storage
[Container: DigitalOcean Spaces]
Saves the uploaded letters (PDFs)
Postal service
[Software System]
Sends the letter physically

Figure 4.19: C4 Model Level 3 Backend

Data Model

The following graphic displays our data model. As mentioned in previous chapters, we use a relational database to persist the data, where each rectangle in the diagram represents a table. The decision to create a separate table for addresses was made deliberately to simplify maintenance, for instance, if the "Country" field needs to be extended in the future.
Additionally, it is worth noting that we made a deliberate choice to allow each user of the application to have only one address. Given that the application is currently in the prototype stage, this decision adequately encompasses the full range of functionalities.
When our application sends a letter, it can be done in two ways: physically or digitally. This is represented in the "Letter" table with the "type" field, which can take either the value "PHYSICAL" or "DIGITAL". The "recipientName" field is redundant in the digital case compared to the "firstName" and "lastName" fields in the "User" table but is required when sending a physical letter.

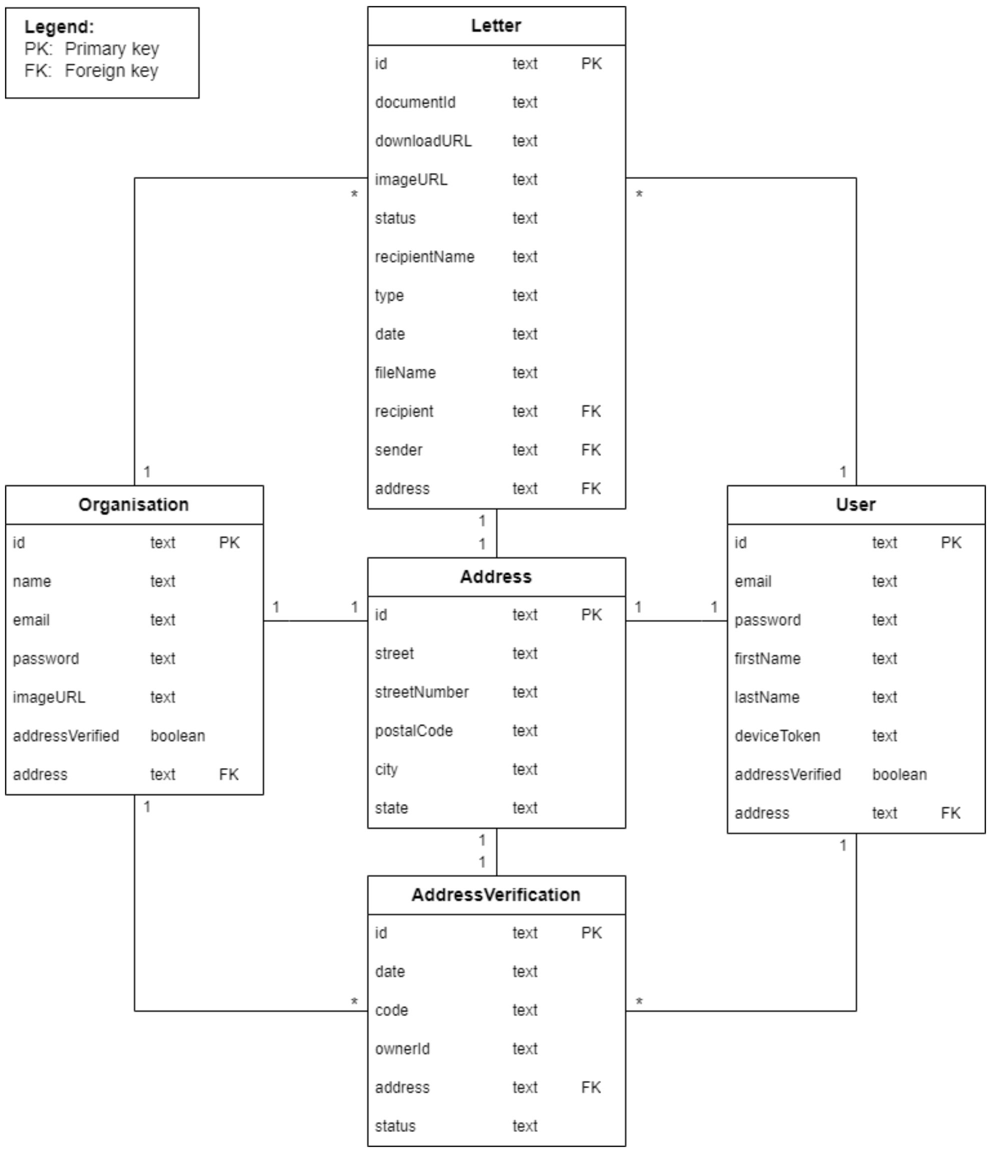


Figure 4.20: Data Model

5 Implementation

5.1 Technology

Our decision process was led by the project requirements, with some technologies prescribed to ensure a standardised approach and others chosen to correspond with our team's skills and preferences. Each technology plays an important part in various portions of the project, resulting in a comprehensive and versatile solution. More information on each technology choice can be found in the sections that follow.

Programming Language

Since our industry partner specified that the backend must be a Node application and that we also had to choose internet technologies for the desktop and mobile applications, the only programming languages we considered were JavaScript, TypeScript and Dart.
Since we decided to write the mobile application in Flutter, we chose Dart as the programming language for the mobile application.
For the backend and the desktop applicationlication we chose TypeScript because it is a superset of JavaScript and is a strongly typed language. And for team collaboration it is better to use the same language if we have clear interfaces and expectations.

Frontend Framework and Libraries

Electron

As mentioned, this is a requirement from our industry partner and we have chosen Electron-vite. A tool that simplifies the setup process and speeds up the creation of our application. Electron-vite includes pre-configured settings that reduce the complexity associated with setting up Electron projects, allowing us to focus more on the application logic and features.

React

We chose React as our frontend framework because it is a popular framework with a large community and lots of documentation. It is also a component-based framework which makes it easy to reuse components and create a consistent user interface. But the main reason we chose React is because we have experience with it and we know how to use it.

Flutter

We chose Flutter as our frontend framework for the mobile application, because it is a cross-platform framework that allows us to write one codebase for both Android and iOS. It is also a component-based framework which makes it easy to reuse components and create a consistent user interface. But the main reason we chose Flutter over React Native is the fact that it is faster and has a better performance.

PDF-Viewer Desktop Application

To view the uploaded PDF, we decided to use react-pdf as the PDF viewer. We looked at the available options such as simply using `<iframe src="files/example.pdf"/>` which is easy to implement, but we only wanted to have the first page of a PDF.
Another option was to use pdf.js which is a JavaScript library for rendering PDF documents in the browser. But we decided to use react-pdf because it is a React component that allows us to render PDF documents in the browser and it is easy to use.

PDF-Viewer Mobile Application

For the integration of PDF viewing functionality in our application, we chose to utilize a pre-built package within the Flutter framework. Our primary criterion for selecting this package was its compatibility with the required platforms, specifically Android and iOS.
After evaluating various options, we decided on the PDFx package, which is available on pub.dev. PDFx is a comprehensive Flutter plugin, designed to render and display PDF documents as images. It supports a wide range of platforms including Web, MacOS, Windows, Android, and iOS, making it an ideal choice for our cross-platform application needs.

QR-Code Scanner

In addition to PDF viewing, our application required the capability to scan QR codes. To achieve this, we opted for a well-established solution within the Flutter ecosystem. Our choice fell on the ’`qr_code_scanner`’ plugin, a popular tool among Flutter developers. This plugin is known for its reliability and widespread use in the community, making it a trustworthy choice for our application.

Backend Framework and Libraries

Nestjs

Although Express is widely used and better suited for small applications, offering more flexibility, we deliberately chose NestJS for the following reasons:

- **TypeScript-based:** Nest is built in TypeScript, leading to increased stability. Static typing allows for early error detection and a clearer codebase.
- **Convention over Configuration:** In contrast to Express, NestJS follows the "Convention over Configuration" paradigm. This means that developers can avoid much of the configuration by adhering to clear conventions. This promotes a unified code structure and accelerates the development process by relieving developers of certain decisions.
- **Modular Architecture:** NestJS provides a modular architecture that facilitates development and maintenance. By using modules, different functionalities can be cleanly separated and reused.
- **Dependency Injection:** NestJS integrates the concept of Dependency Injection, contributing to well-structured and testable code. This also encourages loose coupling between different components.

Orm Prisma

We have chosen Prisma as our ORM tool for the following three reasons:

- **Type-Safety and Code Auto-Generation:** Prisma fully leverages the advantages of TypeScript by providing strong typing for database queries. This means that queries can be statically checked during development, reducing runtime errors. Prisma automatically generates part of the code based on the database schema, reducing development time and ensuring code synchronization with the schema.
- **Performance and Optimizations:** Prisma is designed to create efficient database queries and offers optimizations such as batch data loading. This allows for the generation of efficient queries and minimizes the number of database accesses, enhancing the overall performance of the application.
- **Data Modeling and Simplicity:** Prisma enables clear and declarative data modeling through the schema.prisma file, leading to a simpler and more understandable codebase.

Localization i18n

At this point, it is worth noting that we utilize the Node package nestjs-i18n to extract all message strings from the code files to a single point. This will simplify the management of different messages and facilitate the implementation of localization in the next phase of this project.
Currently, this may seem a bit overhead, but in our opinion, it is the correct approach, especially considering that when this application is intended for use throughout Switzerland, it must support at least three languages. Therefore, this implementation will establish a solid foundation for the localization process.

PDF to PNG Converter

After deciding to show the user an image of the first page of the letter, we had to implement this logic in the backend. What initially seemed simple turned out to be a greater challenge, as most libraries we evaluated faced a common issue: dependency on external scripts, which was impractical for us due to maintenance reasons. Nevertheless, after extensive research, we found a solution to this problem with the Node package pdf-to-png-converter.

Database

In the decision-making process regarding the choice of the database, we opted for PostgreSQL. This decision was driven by straightforward considerations. Firstly, our industrial partner expressed a preference for a relational database. Secondly, all team members possessed prior experience with PostgreSQL.

For local development environments, we implemented a Dockerized PostgreSQL database to ensure consistency across development machines. Dockerization facilitates the seamless setup and teardown of the database environment, ensuring that all team members can work with a standardized PostgreSQL instance. The use of Docker containers ensures that dependencies are isolated, making it easier to manage and reproduce the development environment.

In the production environment, we chose to leverage Digital Ocean's managed PostgreSQL service for its robust features and ease of administration. Digital Ocean's managed PostgreSQL service provides automated backups, scaling capabilities, and optimized performance, reducing the operational overhead for our team. The decision to utilize a managed service aligns with our goal of ensuring a reliable and scalable production database, allowing us to focus more on application development and less on database management.

External Services

Shared Preferences Mobile App

Since we set up the endpoints in the backend and their paths dynamically, we had to provide functionality to store some data on the mobile device. To implement this, we used shared preferences. This feature is particularly useful for storing user settings or preferences. It allows applications to remember choices made by the user, such as login details, theme preferences, or language settings. Moreover, it is easier to implement than a database connection on a mobile device.

Object-Storage Digital Ocean

As the main task of the application is to digitally send letters, it is logical to utilize Object-Storage for file storage. Our initial choice was Azure Blob Storage from Microsoft, as the team already had experience working with it. However, the industrial partner requested the entire application to be hosted on Digital Ocean. Consequently, we opted for the Object-Storage offered by Digital Ocean, specifically Digital Ocean Spaces.

Digital Ocean Spaces provides us with a reliable means of storing and managing files. In comparison to the setup of Azure Blob Storage, the onboarding and implementation of Spaces proved significantly more straightforward.

The seamless integration of the Amazon S3 SDK for Spaces connectivity was a crucial part of this procedure. We achieved the flawless integration with Spaces by using Amazon's AWS S3 SDK. This SDK provides a uniform interface to access many Object-Storage services, which greatly simplifies implementation and adds a ton of functionality.

The following code snippets from the project illustrates the simplicity of the implementation:

```
export class FileService {
 client: any;
 constructor(private readonly i18n: I18nService) {
  // Initialize the S3 client with the credentials and endpoint details
  this.client = new S3({
   forcePathStyle: false,
   endpoint: process.env.DO_SPACES_ENDPOINT,
   region: process.env.DO_SPACES_REGION,
   credentials: {
    accessKeyId: process.env.DO_SPACES_KEY as string,
    secretAccessKey: process.env.DO_SPACES_SECRET as string,
   },
  });
 }
```

Listing 5.1: Code to instanciate the S3 Client

```
async uploadFile(file: Buffer, Key: string, fileName: string) {
 try {
  // Send the file to the S3 bucket using the PutObjectCommand
  await this.client.send(
   new PutObjectCommand({
    Bucket: process.env.DO_SPACES_NAME,
    Key,
    Body: file,
    Metadata: { 'file-name': `${fileName}` },
   }),
  );
 } catch (err) {
  console.error(err);
  throw new BadRequestException(this.i18n.t('messages.errorSaveFile'));
 }
}
...
```

Listing 5.2: Code for uploading file to Digital Ocean Spaces

Pingen Postal Service

Pingen is an online mail service that dispatches physical letters. This functionality is essential within the application, as previously mentioned, to enable the sending of letters even when the recipient is not registered on the platform or the address has not yet been verified.
The application itself relies on the Pingen service to send users a letter for address verification. Communication takes place through a REST API with the Pingen service. The following details the exact process:

The first step involves the application authenticating with Pingen.

```
private async authenticateOnPingen(): Promise<PingenGetAccessTokenResponse> {
 try {
  return (
   await this.httpService.axiosRef.post(
    process.env.PINGEN_AUTH_URL as string,
    {
     grant_type: 'client_credentials',
     client_id: process.env.PINGEN_CLIENT_ID,
     client_secret: process.env.PINGEN_CLIENT_CREDENTIALS,
    },
    {
     headers: {
      'Content-Type': 'application/x-www-form-urlencoded',
     },},)
  ).data as PingenGetAccessTokenResponse;
 } catch (err) {
  console.error(err);
  throw new InternalServerErrorException(...);
 }
 }
```

Listing 5.3: Code for authenticate the application on pingen

Subsequently, another request is made to obtain an upload URL for the letter.

```
private async getUploadUrl(access_toke: string) {
 try {
  const result = (
   await this.httpService.axiosRef.get(`${process.env.PINGEN_URL}/file-upload`,
    {
      headers: { Authorization: `Bearer ${access_toke}` },
   })
  ).data;
  return result.data as PingenGetUploadUrlResponse;
 } catch (err) {
  console.error(err);
  throw new InternalServerErrorException(...);
 }
}
```

Listing 5.4: Code to get the upload URL from pingen

The letter can then be uploaded to Pingen using this URL.

```
private async uploadLetterToPingen(url: string, file: Buffer) {
 try {
  await this.httpService.axiosRef.put(url, file);
 } catch (err) {
  console.log(err);
  throw new InternalServerErrorException(...);
 }
}
```

Listing 5.5: Code for uploading a file to pingen

Pingen automatically verifies if the letter has a valid format and subsequently dispatches it.

Notification Service

Due to the fact that the majority of eLetter users utilize the application through a mobile application, we have chosen to integrate a notification service for sending push notifications to the application.
Initially, we evaluated OneSignal, as their documentation suggested it to be straightforward to implement. However, it turned out that, for sending notifications to Android devices, a Firebase account is required, which is then linked with OneSignal. As a result, we opted to simply leverage the SDKs and tools that Firebase inherently utilizes. This allowed us to bypass the overhead that would have been introduced by OneSignal.
This decision proved to be correct, as the implementation in both the backend and frontend was straightforward. The following code snippets showcase the entire code needed to send a notification to the user.

```
export class NotificationService {
 ...
 public async sendNotificationToDevice(token: string, organisationName: string)
   {
  try {
    await this.firebase.messaging().send({
    token,
    notification: {
     title: this.i18n.t('messages.notificationTitle'),
     body: `${this.i18n.t('messages.notificationText')} ${organisationName}`,
    },
   });
  } catch (err) {
   console.error(err);
  }
 }
}
```

Listing 5.6: Code for sending push notification to a mobile device

5.2 Test Concept

5.2.1 Frontend

Unit-Tests
We initially planned to use unit testing, but after talking to our industry partner we decided that it would not make sense and that the features were more important than the testing.

Usability-Tests
Although the end-user test is not required for the Desktop application, we thought it would be nice to present it to users and test it. The non-functional requirement was that three test users would rate our mobile application eight or higher out of ten. So we decided to do the desktop application with 2 more people, for a total of 6 people.

Result Desktop Application

The results for the Usabilty-Tests were the following:

- **Participant 1:** Overall rating of 7
- **Participant 2:** Overall rating of 8
- **Overall rating:** 7 + 8 => **7.5**

In the context of our project we consider these ratings to be good, but they show us that there is still work to be done in our UI. Detailed information can be found in the chapter appendix under the section "*Usablity Test Desktop Application*" and under the the section "*Testing code snippet*".

Result Mobile Application

The results for the Usabilty-Tests, broken down by user, were the following:

- **Participant 1:** 9 + 8 + 9 + 7 => 8.25
- **Participant 2:** 9 + 10 + 10 + 9 => 9.50
- **Participant 3:** 9 + 9 + 10 + 8 => 9.00
- **Participant 4:** 9 + 8 + 9 + 8 => 8.50
- **Overall rating:** 8.25 + 9.50 + 9.00 + 8.50 => **8.8125**

The results for the Usabilty-Tests, broken down by category, were the following:

- **Overall Experience:** 9 + 9 + 9 + 9 = 9.00
  (content, layout)
- **Usability:** 8 + 10 + 9 + 8 = 8.75
  (layout, responsiveness)
- **Dynamic-Content:** 9 + 10 + 10 + 9 = 9.50
  (responsiveness, content)
- **Look and feel:** 7 + 9 + 8 + 8 = 8.00
  (color)
- **Overall rating:** 9.00 + 8.75 + 9.50 + 8.00 => **8.8125**

Detailed information can be found in the chapter appendix under the section
"*Usablity Test Mobile Application*".

5.2.2 Backend

The decisions regarding the test concept for the backend are based on two key aspects. The backend application is characterized by a limited manifestation of business logic, with the main focus primarily on writing and reading data in the database. A central objective is to link these data and transmit them through corresponding APIs.

In this project, the prioritization of features by our industrial partner takes center stage, leading to the decision to emphasize integration tests for the backend and entirely omit unit tests.

The benefits of integration tests extend to the comprehensive verification of the entire application, ensuring smooth collaboration among all components. Through this extensive test coverage, potential weaknesses in the application can be identified and addressed early. An additional advantage of integration tests lies in their proximity to user behavior, enabling a more realistic simulation of user interaction. This contributes to a more effective validation of the application from the perspective of end-users and prepares for potential challenges.

Test implementation

For the execution of integration tests, we utilized the provided tools from NestJs. In conjunction with the Supertest library, these tools enable the sending of HTTP requests to endpoints, allowing for a comprehensive examination of the application from start to finish. To avoid API calls to external services, we employed NestJs's internal mocking mechanisms.
This allows for the straightforward replacement of affected modules or services with corresponding mocks during the instantiation of the application for testing purposes.

A more significant challenge arose with the database. The optimal solution would have been to use an in-memory database for testing to enhance test performance. Unfortunately, Prisma does not offer an interface for an in-memory database. Consequently, we were compelled to launch a Docker container containing a PostgreSQL database for the tests. However, this led to additional issues since the Jest test runner conducts tests in parallel by default. This resulted in unforeseen errors, as entities sometimes remained in the database when they should have been removed or data was deleted that shouldn't have been.

To address this issue, we opted to run the tests sequentially. It turned out that this had no significant impact on runtime since the number of tests was not substantial. The overhead incurred by parallelization had a greater influence on the test runtime.

Result

The tests run successfully. You can see a screenshot of a test run in the chapter appendix under the section *Screenshots*.

5.3 Deployment

User Applications Deployment

The deployment of the desktop and mobile applications is relatively straightforward. Since we are not publishing the applications as part of this work, deployment simply involves creating (building) the respective application. For this purpose, we used GitHub Actions, which builds the applications directly on GitHub. It is worth mentioning that we only create the desktop application for Windows, as our industrial partner expressed the preference for the application to primarily run on Windows. Similarly, we create the mobile application only for Android devices and not for Apple devices. This is because creating an iOS application requires an Apple device. To save resources, we have therefore decided to deploy only for Android devices. If the mobile application build process is successful, we can download the APK-file as an artifact.
The desktop application is also available as an artifact, which contains the windows exe-file.

Backend Deployment

We also use GitHub Actions for the backend deployment. However, deploying the backend application is somewhat more complex than deploying the user applications. It begins by conducting tests, and only after successful testing does the build step commence.
Additionally, for deployment, we had to set up a self-hosted runner on our Digital Ocean Droplet. The exact functioning of these two steps in the workflow is explained in the following.

End-to-End Tests (e2e-tests):

The first job in our workflow performs End-to-End tests. These tests run in an Ubuntu Latest environment on GitHub, with a specially configured PostgreSQL service provided. The PostgreSQL database is configured on a port of our choice, ensuring that our tests run against a clean database environment. The individual steps of this job include checking out the repository, setting up Node.js, installing dependencies, and finally, running the End-to-End tests.

Build and Deployment (build):

The second job in our workflow is responsible for the actual building and deployment. This job runs on a self-hosted runner located on the same droplet as the production environment. The steps of this job include checking out the code, setting up Node.js, installing dependencies, building the application, stopping the existing API, starting the new changes, creating a snapshot of the process, and finally restarting the server to activate the updated application. The second job not only deploys the backend it also applies the database migrations to the production database.

The diagram below illustrates the deployment workflow

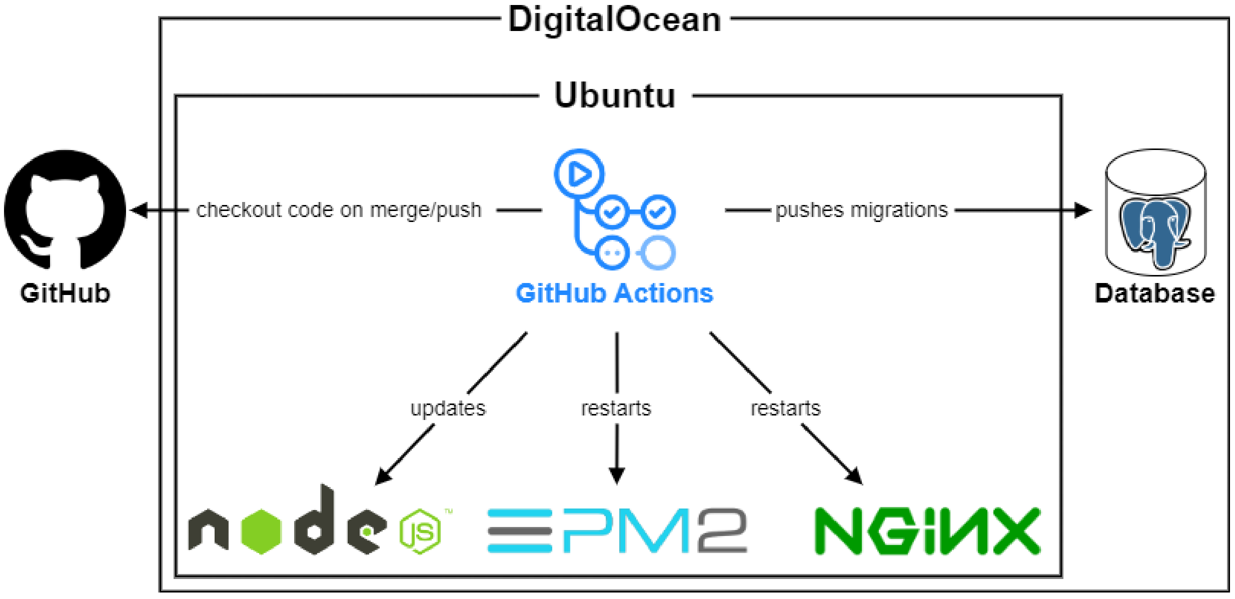


Figure 5.1: Backend Deployment

If you need more information about the deployment steps you can find screenshots of the GitHub Actions in the appendix under the chapter *Screenshots*.

5.4 Production Environment Backend

The production environment is hosted on a Digital Ocean Droplet, equipped with 1vCPU, 2GB RAM, 25GB disk space, and running Ubuntu 23.10 x64. This Droplet represents the second smallest option offered by Digital Ocean. Initial attempts to host the application on the smallest Droplet were unsuccessful due to insufficient memory. The Droplet is located in the Frankfurt data center, chosen for its proximity to Switzerland. The application is deployed on the Droplet using a self-hosted runner from GitHub Actions, details on this process can be found in the following chapter (*Deployment*).

For the database, we utilize the managed PostgreSQL database service from Digital Ocean, as mentioned in the *Database* section. The database is situated in the same data center as the Droplet. We opted for the cheapest plan, named single node, offering 1 GB RAM and 10 GB disk space. While Digital Ocean does not recommend this plan for production, we decided to use it as it aligns with the resource requirements of our project.

As discussed in the *External Services* chapter, we employ Digital Ocean Spaces for file storage. Additionally, for the notification service, we use Firebase Cloud Messaging, and Pingen serves as the postal service provider.

To serve the application on the web, we use the NGINX web server as a reverse proxy, forwarding all requests. NGINX was selected for its popularity and ease of configuration. Meeting a non-functional requirement requiring HTTPS encryption for all requests, we have blocked HTTP port 80 and redirected all traffic to HTTPS port 443. Due to the absence of a dedicated domain, a self-signed certificate is used for secure communication.

To run the application in the background, we utilize the PM2 process manager. Which is a production process manager for Node.js applications which allows to demonize node application so that they can run in the background as a service.

The following image shows the architecture of the production environment.

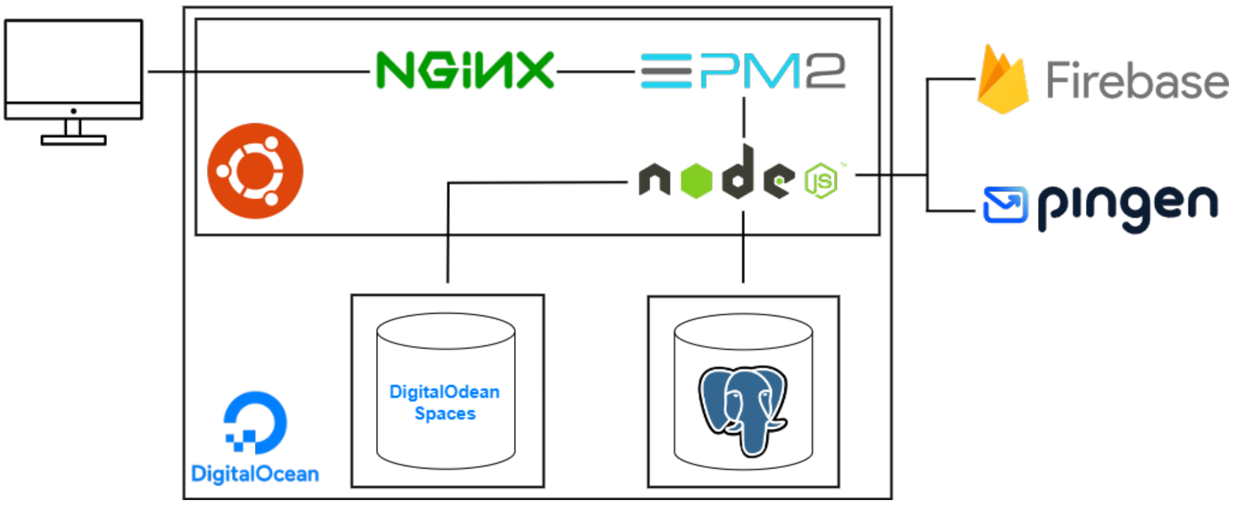


Figure 5.2: Backend Production Environment

5.5 Function Implementation

Before starting the implementation, we divided the application based on functionality and prioritized the implementation according to importance we also had to setup the development environment before we could start the implementation. We initiated with the functionality of sending and receiving letters and viewing them in the mobile application since it constitutes the core task of the application and should thus be included in a Minimal Viable Product (MVP). Following that, we introduced the capability for users to register and login. Recognizing the significance of address verification as a crucial security feature, we decided to implement this prior to the authentication functionality. Lastly, with the lowest priority, we developed the Settings functionality, encompassing all aspects related to modifying user data.
How we implemented each of these functionalities is described in the following sections.

5.5.1 Setup Development Environment

Before setting up a development environment, we decided to build simple applications for both front-ends to gain hands-on experience with the chosen technologies. For the simple applications we need to setup locally on our computer.

For the desktop application we needed to have Node.js version 14.18+ and Vite version 3.0+ before installing Electron-Vite, which will scaffold the application.

For the mobile application we needed to have Windows PowerShell 5.0 and Git for Windows before Flutter SDK can be installed.

For the backend, we needed Node.js before we could use NestJS and Docker. We first implemented a simple REST endpoint that integrated all the technologies, from the controller to the database. This ensured that our approach would work.

After gaining hands-on experience with the technologies, we decided to move on to the development environment. To do this, we created a monorepo with the following structure:

```
eLetter/
├── code
│   ├── backend/
│   ├── desktopApp/
│   └── mobileApp/
└── documentation/
```

After that each team member can start the development. For the desktop application it is the following command:

```
yarn create @quick-start/electron
```

Then follows the prompts:

```
Project name: ... <app-name>
Select a framework: > react
Add TypeScript? ... No / Yes
Add Electron updater plugin? ... No / Yes
Enable Electron download mirror proxy? ... No / Yes

Scaffolding project in ./<app-name>...
Done.
```

For the mobile application it is the following command:

```
flutter create app-name
cd my_app
flutter analyze
flutter test
flutter run lib/main.dart
```

For the backend it is the following command:

```
npm i -g @nestjs/cli
nest new project-name
npx tsc --init
npm install prisma --save-dev
npx prisma init --datasource-provider sqlite
```

The backend is equipped with a pre-existing REST endpoint that benefits from NestJS's "Convention over Configuration" paradigm. To maintain a consistent approach across both desktop and mobile applications, we have implemented a organized set of principles for using the REST handler
At the core of this strategy is a basic service class responsible for facilitating REST calls to the backend. In the desktop application we use the Axios base service, while in the mobile application we use the RestBaseHandler. This approach ensures consistency and efficiency when communicating with the backend. Further information is available in following chapter Authentication.

5.5.2 Sending and Receiving Letters

Sending and receiving letters constitute the central function of this application. The application is designed in such a way that an organization, through the desktop application, can digitally send a letter to a customer, i.e., a user of the eLetter mobile application, provided that the user has verified their address.

The implementation of sending in the desktop application is relatively straightforward. We have created two views for this purpose. In the first view, the user can upload a letter in the form of a PDF using drag and drop. It should be noted that the letter must adhere to the letter format of Switzerland (see *Conclusion*). After the PDF has been loaded into the application, the user moves to the second view, where they can provide the corresponding metadata (address and name of the recipient). Once these details are provided, the user can send the letter, meaning the letter is sent to the backend via a POST request.

In the backend, data is first validated, and it is checked whether a file exists and whether the file has the correct format. Subsequently, the database is queried for the user with the specified address provided by the organization. This search is performed through a simple string comparison of various values at the database level. If the user is not found or if the user exists in the database but has not verified their address, the letter is delivered to them in the conventional way. For this purpose, we have integrated a Post Service in the backend, namely Pingen. Details about integrating this service can be found in the *Pingen Postal Service* chapter. If the transmission is successful, the metadata of the letter is written to the database (for more information, see the *Database* section).

In the event that the identified user has verified their address, the letter is sent digitally to them. This means that the backend initially creates a preview image of the first page and then stores it along with the letter in an Object Storage (see *Object-Storage Digital Ocean* section). If the files are successfully stored, a preSigned URL is generated for them, along with the metadata of the letter, and written to the database.

If this process is successful, the recipient of the letter can view it on their smartphone. To notify them, the backend utilizes the Notification Service from Firebase to send a push notification to the user's smartphone (see Notification Service).

The following code snippet shows the implementation of the function which is responsible for the sending process in the backend:

```
async sendLetter(file: Express.Multer.File, dto: LettersRequestDto) {
 const user = await
   this.userRepo.findUserByAddress(this.createUserAddressFromDto(dto));
 const organisation = await this.organisationRepo.findeById(dto.organisationId);

 if (!organisation) throw new
   BadRequestException(this.i18n.t('messages.dataError'));
 if (!organisation.addressVerified) throw new
   UnauthorizedException(this.i18n.t('messages.errorNotVerified'));

 if (!user || !user.addressVerified) {
   await this.lettersRepo.createPhysicalLetter(dto, 'NOT_YET', organisation.id,
   file.originalname);
   this.postalService.sendLetterOverPingen(file.buffer, file.originalname);
   return this.i18n.t('messages.sendLetterPhysically');
 }

 const documentId = `${organisation.id}/${uuid()}_${file.originalname}`;
 await this.fileService.uploadFile(file.buffer, documentId, file.originalname);
 const url = await this.fileService.generatePreSignedUrl(documentId);

 const firstPageImage = await this.createImageFromFirstPage(file);
 const imageId = `${organisation.id}/${uuid()}_${firstPageImage.name}`;
 await this.fileService.uploadFile(firstPageImage.content, imageId,
   firstPageImage.name);
 const imageURL = await this.fileService.generatePreSignedUrl(imageId);

 await this.lettersRepo.createDigitalLetter(
  dto,
  ocumentId,
  organisation.id,
  user.id,
  url,
  file.originalname,
  imageURL,
 );
 if (user.deviceToken) {
  await this.notificationService.sendNotificationToDevice(user.deviceToken,
   organisation.name);
 }
 return this.i18n.t('messages.sendLetterDigitally');
}
```

Listing 5.7: Backend code responsible to send letters

5.5.3 Viewing Letters

For the display of letters, we have implemented four views in the mobile application. These include the mailbox, which shows all the mails the user has received, the "New Mail Page," which only displays the letters the user has not yet read, the "Letter Details Page," which displays the metadata and a preview of the letter, and finally, the "PDF View Page," where a letter can be read, meaning the PDF is displayed.

When a user wants to open a letter, they can do so either in the mailbox or on the "New Mail Page" by clicking on the corresponding letter. Once this is done, a REST call is sent to the backend, which returns the metadata of the letter as a response. In the metadata, in addition to the sender's name and time, there is also the filename and a pre-signed URL for the image of the first page. This is needed to display the image of the first page to the user. Additionally, there is a pre-signed URL through which the document can be downloaded directly from Digital Ocean. This is required if the user wants to view the PDF file in the application.

We opted for this approach with the pre-signed URLs on the advice of our industry partner because it avoids the detour through our own backend when downloading letters, leading to resource savings on our part.

5.5.4 Register

The registration process is largely identical for both the user and the organization. Instead of a first and last name, the organization needs to provide its name and additionally upload an image of the organizational logo, which will be displayed to the user in the mailbox.

In the backend, during the registration process, validation is performed on the entered data, and it is checked whether the email address provided by the organization or the user has already been used, using a Unique Constraint in the database. After the successful registration of the user, meaning after the data has been successfully stored in the database, it is worth noting that the passwords are hashed before they are stored in the database. The verification process is then initiated.

Further information can be found in the *Verification* section. Relevant UI images of the applications can be found in the *Screenshots* section of the appendix.

It should be noted that, in addition to the backend validations, a validation for the fields email, password, and postal code has been implemented in the desktop application. This helps avoid unnecessary requests and optimizes the response time in the user interface. Particularly in the registration of organizations, more data processing is required in the backend due to the logo that needs to be stored. For this reason, we have opted for this approach and the necessary code snippet can be seen below.

```
class inputValidator {
 public email = (email: string): string => {
  // This regex therm is shortend in this documentation for better readability.
  const re = /^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*)|(".+"))...
  if (!re.test(email)) {
   return 'Invalid email address'
  }
  return ''
 }

 public password = (password: string): string => {
  if (password.length < 3 || password.length > 20) {
   return 'Password has to be between 3 and 20 characters'
  }
  return ''
 }

 public postalcode = (postalCode: string): string => {
  const numericPostalCode = parseInt(postalCode, 10)
  if (isNaN(numericPostalCode) || numericPostalCode < 1000 || numericPostalCode
    > 9999) {
   return 'Postal code has to be a number between 1000 and 9999'
  }
  return ''
 }
}
```

Listing 5.8: Desktop Application code to verifi input values

5.5.5 Login

A user can use their email address and password to log in after successfully registering. The desktop and mobile applications use the same implementation: we've made a view where users can input their email address and password. After a successful registration, the organization's email is kept in the local storage to improve the desktop application's usability. It loads automatically into the email input area upon reopening the login window, requiring only the password to be typed.

When a user logs in through the mobile application, in addition to the email and password, the device token generated by Firebase is also sent. This token is necessary to send push notifications through Firebase.

The following code snippet shows the login view code in the desktop application:

```
const LoginPage: React.FC = () => {
 ...
 return (
  <div className={`login-container ${loading ? 'loading' : ''}`}>
   <div className="return-button-login" onClick={handleReturn}>
    <i className="fa fa-arrow-left"></i>
   </div>
   <h1 className="login-title">Login</h1>
   <form onSubmit={handleSubmit} className="login-form">
    <div className="login-group">
     <label>
       E-Mail:
       <input
        type="text"
        value={email}
        onBlur={handleEmailBlur}
        onChange={handleEmailChange}
        className="login-input"
       />
     </label>
     </div>
     <div className="login-group">
      <label>
        Password:
        <input
         type="password"
         value={password}
         onChange={handlePasswordChange}
         className="login-input"
        />
       </label>
     </div>
     <button type="submit" className="login-button">
      Login
     </button>
   </form>
   <div className="vertical-line-login"></div>
   <div className="email-icon"><LetterIcon/></div>
   {loading && (
     <div className="loader-wrapper">
      <BarLoader loading={loading} aria-label="Loading Spinner"
    data-testid="loader" />
     </div>
   )}
  </div>
 )
}
```

Listing 5.9: UI code for logine desktop application view

To better understand the login process in the backend you can look at the following code snippet which shows the function that is called when the user wants to login.

```
public async signInUser({ email, password, deviceToken }: SignInUserRequestDto,
   res: Response) {
 const foundUser = await this.userRepo.findUserByEmail(email);
 if (!foundUser) throw new
   BadRequestException(this.i18n.t('messages.wrongCredentials'));
 if (!(await this.comparePasswords(password, foundUser.hashedPassword))) {
  throw new ForbiddenException(this.i18n.t('messages.wrongPassword'));
 }

 if (!foundUser.deviceToken || foundUser.deviceToken !== deviceToken) {
  this.userRepo.updateDeviceToken(foundUser.id, deviceToken);
 }

 // Add auth cookie to response
 const token = await this.signToken(foundUser.id, foundUser.email);
 if (!token) throw new ForbiddenException();
 res.cookie('token', token, {
  expires: new Date(Date.now() + 3600 * 1000 * 24),
  secure: true,
  sameSite: 'none',
  httpOnly: true,
 });
 return res.status(200).send(foundUser.id);
}
```

Listing 5.10: Backend code responsible for user login

5.5.6 Verification

Start a new verification process:

After a user successfully registered or updating his address, which requires a new verification, the following steps follow:

1. **Generate Verification Code:**
   - After a user successfully registers or updates their address, a new verification code is generated.
2. **Insert Code into PDF Template:**
   - The generated code is inserted into a predefined PDF template.
3. **Distinguish Between Desktop and Mobile Applications:**
   - If the user registers via the desktop application, the code is inserted into the letter as a character string.
   - If the user registers via the mobile application, the code is integrated into the letter as a QR code.
4. **Use Pingen Postal Service for Sending Letters:**
   - The letters, containing the verification code, are sent using the same methods as physical letters. (See section *Pingen Postal Service*.)
5. **Enter Code into Database:**
   - Once the PDF with the verification code is successfully created and sent to the postal service, the code is entered into the database.
   - The status in the database is set to "LETTER_SEND."
6. **Database Unit Represents Link Between User/Organization and Address:**
   - The database unit serves as a link between the user/organization and the respective address. (See section *Backend*.)

Completion of the verification process:

To complete the verification process, we have created an input field in the desktop application (see screenshots in the appendix section *Screenshots*) that is displayed when the user is not verified and clicks on "Settings". They will now need to enter the code from the letter into the field. Once they have done this, the code is sent to the backend via a REST call along with their user ID, which then checks that the code is still valid and that the code and user ID match.
In the mobile application, the process is slightly different. Instead of entering the code manually, the user can scan a QR code by clicking on the button in the "Settings" area (see screenshots in the appendix section *Screenshots*). The part in the backend remains identical.

The following code snippet shows the verification logic in the backend:

```
...
// This method is used to verify an User
public async verifyUser(ownerId: string, code: string) {
 const verification = await this.verify(ownerId, code);
 await this.userRepo.updateVerificationStatus(verification.ownerId, true);
 return this.i18n.t('messages.userVerified');
}

// This method is used to verify an Organisation
public async verifyOrganisation(ownerId: string, code: string) {
 const verification = await this.verify(ownerId, code);
 await this.organisationRepo.updateVerificationStatus(verification.ownerId,
   true);
 return this.i18n.t('messages.organisationVerified');
}
...
// This methods contains the logic of verifing an user or an organisation
private async verify(ownerId: string, code: string):
   Promise<AddressVerfication> {
 if (!ownerId || !code) throw new ForbiddenException();
 const verification = await
   this.verificationRepo.getVerificationByOwnerIdAndCode(ownerId, code);
 if (!verification) throw new
   ForbiddenException(this.i18n.t('messages.verificationNoMatchOwnerCode'));
 if (verification.status === 'DONE') throw new
   BadRequestException(this.i18n.t('messages.verificationCodeUsed'));
 await this.verificationRepo.updateVerificationStatus(verification.id, 'DONE');
 return verification;
}
...
```

Listing 5.11: Backend code for verifing user/organisation address

5.5.7 Authentication

In the authentication process, we opted for a cookie-based approach. The use of NestJS for implementation facilitated this choice, as Nest already provides pre-implemented functions for integrating cookie-based authentication in projects.

The authentication process for users and organizations is identical during a request. After a successful account creation, the user must log in with an email and password. Upon success, a cookie with a token is sent back to the user client, which is included in every subsequent request requiring authentication.
If the user clicks "Logout" in the mobile appication or desktop application, the cookie becomes invalid, preventing access to protected API endpoints. Regarding authorization, we structured it so that each user can only access entities in the database linked to their respective user/organization ID, eliminating the need for a role system. The following diagram illustrates the authentication process.

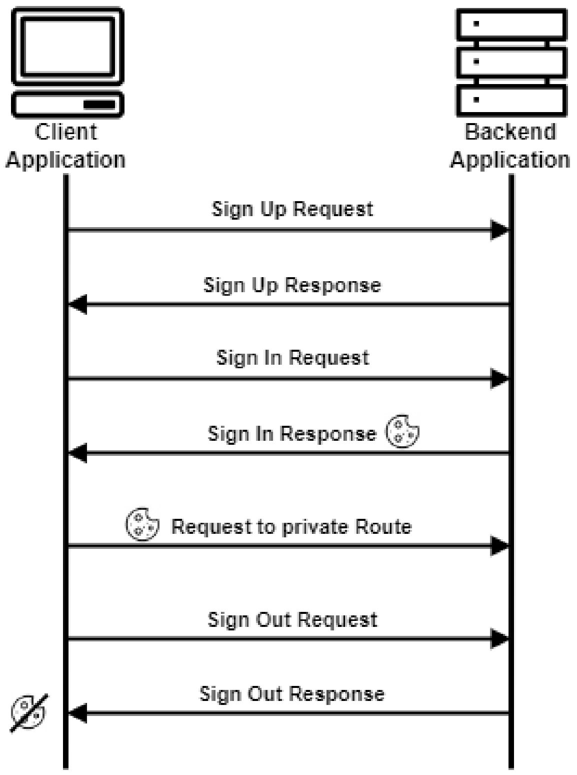


Figure 5.3: Authentication Process

For the implementation of the authentication process in the desktop application, we did not need to do much. This is because Electron is a browser-based technology, and the browser manages cookies. So we only need to whether the user has permission to access the requested page and react accordingly if not.

In the mobile application, a bit more implementation was required for cookie handling. Since Flutter does not handle cookie management by default, we implemented it ourselves. We wrote the RestBaseHandler to handle communication with the backend.
For specific areas such as sending letters and authentication, we defined our own handlers that call the RestBaseHandler through dependency injection. When a user logs in through the mobile app, the RestBaseHandler sends a REST call to the backend. If authentication is successful, it receives the response along with the cookie. It reads the token from the cookie and stores it in the shared preferences. When another endpoint, such as the Settings endpoint, is called, the RestBaseHandler reads the token from the shared preferences and creates the cookie, which is then sent to the backend with the subsequent REST call.

The following code snippet shows the implementation of the cookie handling in the RestBase-Handler:

```
class RestBaseHandler {
 ...
 RestBaseHandler({required bool withCookie, required this.baseUrl})
  : dio = Dio(BaseOptions(
   baseUrl: baseUrl,
   headers: {
    'Content-Type': 'application/json',
    'Accept': 'application/json',
   },
  )) {
  if (withCookie) {
   _loadCookie();
  }
  _configureDio();
 }
 Future<void> _loadCookie() async {
  String? token = await _loadToken();
  // This line loads the token from the shared preferences
  String cookieToken = "token=" + token! + ";";
  await headers.remove("Cookie" ?? '');
  await headers.putIfAbsent("Cookie", () => cookieToken ?? '');
 }
 Future<String?> _loadToken() async {
  return await storage.read(key: 'bearerToken');
 }
 ....
 Future<void> _saveToken(String cookie) async {
  var match = RegExp(r'token=([^;]+)').firstMatch(cookie);
  if (match != null) {
   String token = match.group(1)!;
   await storage.write(key: 'bearerToken', value: token);
  }
 }
 Future<void> _saveCookie(String cookie) async {
  var match = RegExp(r'token=([^;]+)').firstMatch(cookie);
  if (match != null) {
   String token = match.group(1)!;
   // This line saves the token in the shared preferences
   await storage.write(key: 'cookie', value: token);
  }
 }
}
```

Listing 5.12: Mobile application RestBaseHandler with save and load cookie function

The function below is part of the RestBaseHandler and is used when a post call is sent to the backend.

```
Future<dynamic> post(String endpoint, Object data) async {
try {
 Response response = await dio.post(endpoint, data: data);
 String? cookie = response.headers.value('set-cookie');
 if (cookie != null) {
  _saveCookie(cookie);
  _saveToken(cookie);
 }
 return response.data;
 } on DioError catch (e) {
  throw Exception('Failed to update data: ${e.message}');
 }
}
```

Listing 5.13: Mobile application RestBaseHandler post method

The code snippet below contains a part of the AuthHandler the the method which is shows uses the method form the code snipped above to send a post request to the backend when the user wants to login.

```
class AuthHandler {
 ...
 Future<String> signIn(Login credentials) async {
  String jsonBody = json.encode(credentials.toJson());
  final api = RestBaseHandler(
      baseUrl: 'https://209.38.244.175/auth/signIn/user', withCookie: false);
  return await api.post('', jsonBody);
 }
 ...
}
```

Listing 5.14: Mobile application AuthHandler post method

5.5.8 Settings

Customizing the settings is a relatively straightforward process. However, during the implementation, we had to distinguish between a user and an organization. This was necessary because an organization can have a profile picture, while a user cannot. This implies that an organization can change its picture. However, this change should not trigger a new address verification process, a consideration we had to address when implementing the logic in the backend.
In implementing the logic, we kept it straightforward: when a user clicks on "Update Settings" in one of the applications, the entire form is sent to the backend. The backend then checks whether the content has changed compared to the stored values and whether a new address verification is needed. If this is the case, the new data is written to the database, and a new address verification letter is sent to the user. If the address has not changed, the new data is written to the database, and the user is notified that the settings have been updated.

5.6 Documentation

5.6.1 REST-API Documentation

The comprehensive documentation of the REST API was crucial for the entire development process, as it serves as the interface for both the desktop application and the mobile application. Therefore, we decided to document all REST endpoints using Postman. The use of Postman not only allows effective sharing of the current REST endpoints among all team members but also permits the manual execution of requests. An export file of the Postman collections can be found in the code repository in the "code/backend/postman-collections" folder.
An Detailed documentation of the REST endpoints can be found in the appendix in the Rest API documentation chapter. At this point it is worth mentioning that we have not created an endpoint for downloading the letters. This is due to the fact that the PDF file of the letter is stored on Digital Ocean Spaces, and to allow secure access, we store a pre-signed URL in the database. This pre-signed URL is then included in the response of the GET /detailsOf/<LETTER_ID> endpoint. Consequently, the client application can download the PDF letter file directly from Digital Ocean Spaces by using this provided, pre-signed URL.

5.6.2 Code Documentation

For readability and consistency we decided to use TSDoc. TSDoc is a TypeScript documentation standardisation proposal that looks like this:

```
export class Statistics {
  /**
   * Returns the average of two numbers.
   *
   * @remarks
   * This method is part of the {@link core-library#Statistics | Statistics
   subsystem}.
   *
   * @param x - The first input number
   * @param y - The second input number
   * @returns The arithmetic mean of `x` and `y`
   *
   * @beta
   */
  public static getAverage(x: number, y: number): number {
    return (x + y) / 2.0;
  }
}
```

Listing 5.15: TSDoc Example

6 Result

In this project, we developed an application that enables companies to digitize their correspondence using a desktop application. The prerequisite is that the customer has installed the corresponding application on their mobile phone and has already verified their address. Once this is done, they can digitally receive all their letters from companies that also use the application. The backend of this application is hosted on Digital Ocean and managed by the publisher of the application. Thus, we were able to fulfill all previously defined functional requirements.

Due to time constraints, it was not possible to implement the optional requirements. However, we were able to partially implement two of them:

1. This concerns the web application for recipients. Since we decided to develop the mobile application using Flutter, it would be fundamentally possible to deploy it as a web application. However, this would still require some additional effort. To ensure a good user experience, we would need to adjust the styling, and additionally, the deployment would need to be set up.

2. This concerns the branding of the desktop application with the companys own design. The application requires companies to add their own logo to the application, which is then displayed. However, for a complete corporate design, colors and fonts should also be customizable.

In contrast to the optional requirements, we were able to fulfill almost all non-functional requirements. The following lists all the requirements we achieved, and those we did not achieve are listed at the end with an explanation of why they were not achieved.

Fulfilled Non-Functional Requirements:

- The development team implemented the features according to the agreed priority with the customer.
- Each page should not take longer than 200 ms to load.
- The desktop application should run on Windows.
- Three out of four test users should rate the UI (categories: layout, responsiveness, color, content) of the application with a minimum score of 8 out of 10 using a mobile phone, where 10 is the best.
- Errors should not generate system failures but show an error message and reset the system to the previous state.
- Every error should be logged in the system.
- Each communication between the front and backend should be encrypted with an SSL certificate.
- Data filled in input fields should be validated before being processed by the system. SQL injection tests of the input fields should not reveal vulnerabilities.
- User passwords are not stored in plain text in the database.
- Business logic in the backend should be modular for easy expansion.
- The backend API should be tested using API testing tools.
- Database, backend, and frontend should be deployed on different instances.

Non-Functional Requirements that were not fulfilled:

In the course of this work, we could not test the requirement that the backend can handle 1000 requests per minute. This was not prioritized highly by us, as the application is intended to be an initial prototype that demonstrates functionality. Therefore, due to time constraints, it could not be tested.

The second requirement that was not fulfilled is the ease of installation. It is fundamentally a simple process for an Electron application to be installed via an exe file, which we successfully created and installed. However, the corresponding exe file also needs to be made available to a user, which was not done in the course of this work. The same applies to the APK file for installing the mobile application.

It is also worth mentioning that the following NFR was not tested: "The database should be able to manage up to 10,000 documents/PDFs." However, in contrast to the other two, this should have been fundamentally achieved. Digital Ocean Spaces provides the necessary resources and capacities to meet the specified requirement. Nevertheless, capacity is always a matter of cost, as additional storage space is associated with higher expenses.

Important to note:

It should be noted that we discovered a flaw in our implementation at the end of the project that affects the long-term usability of the application. Specifically, we incorrectly assumed that the pre-signed URL for file downloads would remain valid indefinitely in our database. However, it has been discovered that a pre-signed URL is only valid for a maximum of 7 days, which was realized too late in the project.

To address this issue in future work, it is recommended to generate the pre-signed URL each time the user views or downloads the document instead of saving it in the database.

7 Conclusion

As mentioned in the "Results" section, we have developed an application as part of this project, enabling companies to digitize their communication with customers. We successfully met all requirements, with the exception of three specified by our industry partner. Consequently, the application we developed serves as a robust foundation for a future application that can significantly simplify correspondence management for companies.
In the ongoing development of the application, particular attention should be given to improving the interface with the postal service. Currently, the application relies on the assumption that the company sending a letter also uploads it in the correct format. If this is not the case, it must be corrected by the application administrator. Unfortunately, time and resource constraints prevented us from addressing this issue during the project. In the following sections, we will discuss the most important points that need to be addressed in the future development of the application.

7.1 Needs to be addressed

Desktop Application

The desktop application has a lot of potential for future development. Looking at the optional requirements, there could be features such as

- cost display for sending
- dashboard for managing employees for sending letters
- automatic address entry for the recipient
- improved company branding

We also have some additional input that can guide further development:

- Creating a website where you can download the application.
- Integration with third-party authentication services for added user security.
- Conducting regular user feedback sessions to identify areas for improvement and user satisfaction.
- Multi-language support.
- Integration with calendars and scheduling.

Mobile Application

In the context of a semester's work, what we have achieved is, in our opinion, a broadly supported feature scope. If you want to take the idea beyond the level of a proof-of-concept, the existing code base should be expanded. This could include the following features.

- Categorization of documents using tags
- Sharing documents with each other
- Functionality to print the documents
- Scanning of documents after they have been filled out
- Creating responses to documents
- Web version for display on devices with larger screens

However, existing features and the code base should also be made more robust if the decision is made to bring the idea to market.

For example, when displaying error messages, only a very coarse-grained distinction was made as to the cause of the error that occurred.

Although we have already carried out some user tests during the creation of the mock-ups, this would be an iterative process that should now go into the next round.

Backend

In the backend application, there are three points that, in our opinion, have room for improvement.

Firstly, there is the library used to generate the preview image of the first page of the letter. The chosen library sometimes encounters difficulties in generating an image from a PDF because it struggles to recognize certain characters.

The second point concerns authentication, which is currently only possible through email and password. For an enhanced user experience, it would be beneficial to allow users to sign in using platforms such as Google, Microsoft, etc. To enhance the security of the application, it would also be advisable to verify not only the user's address but also their email address.

The third point concerns the address comparison. When the user is searched in the sending process of a letter, at the moment we implemented a simple value comparison. On the database side, this is not a problem, but it would be better to use a more sophisticated comparison algorithm which also takes into account the similarity of the values and not only the equality.

7.2 Future Vision

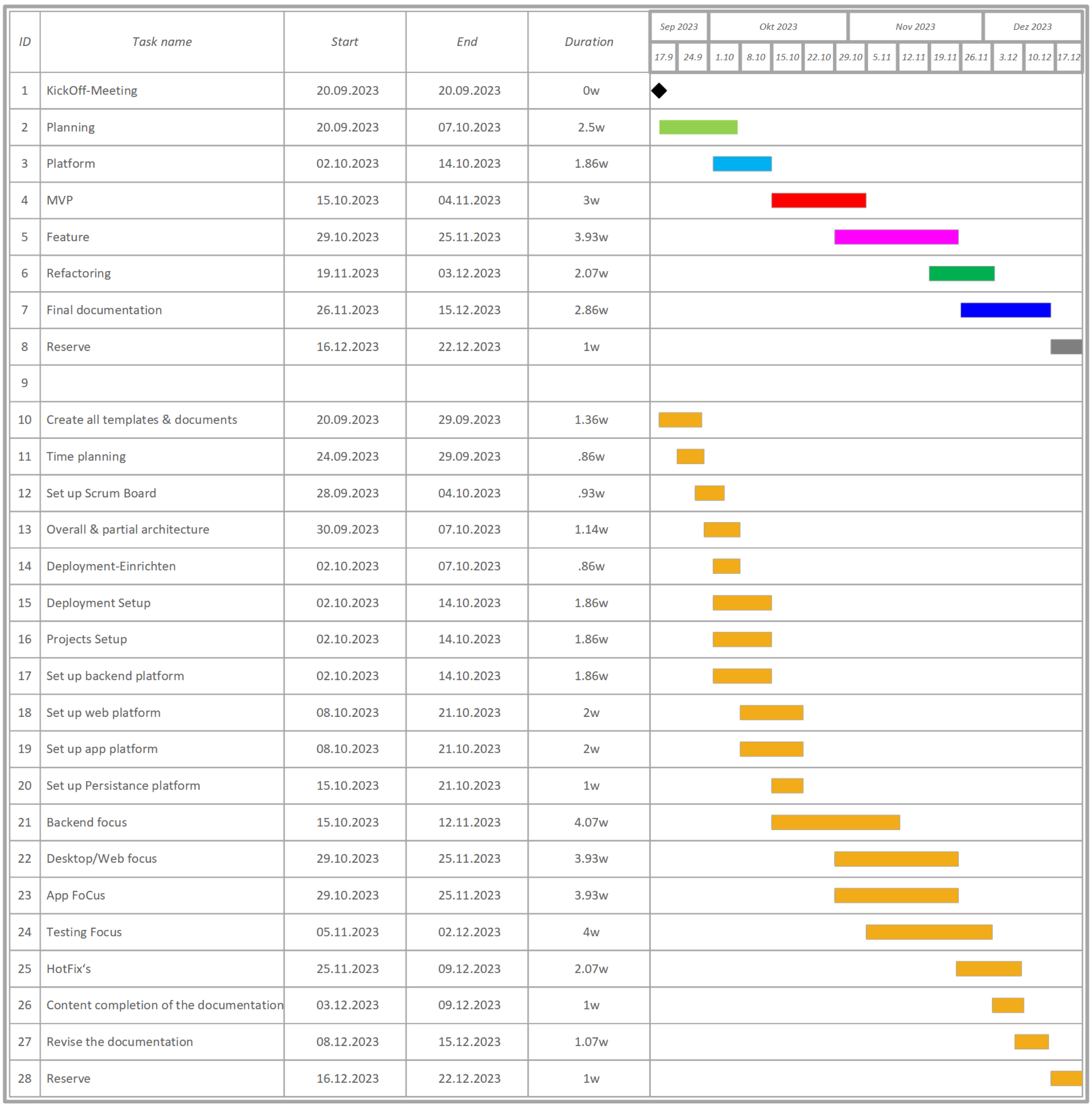
In our opinion, the application has great potential, as there is currently no truly effective solution to digitize mail correspondence in the manner that this application does. For further development, we have the following thoughts and questions:

- It might be beneficial to facilitate mail communication not only for business customer relationships but also for person-to-person interactions.
- A web application would complement the existing application well, making it accessible to an even larger audience.

8 Project and Time Management

8.1 Project Plan

This is the content of the Project Plan section.



ID	Task name	Start	End	Duration
1	KickOff-Meeting	20.09.2023	20.09.2023	0w
2	Planning	20.09.2023	07.10.2023	2.5w
3	Platform	02.10.2023	14.10.2023	1.86w
4	MVP	15.10.2023	04.11.2023	3w
5	Feature	29.10.2023	25.11.2023	3.93w
6	Refactoring	19.11.2023	03.12.2023	2.07w
7	Final documentation	26.11.2023	15.12.2023	2.86w
8	Reserve	16.12.2023	22.12.2023	1w
9				
10	Create all templates & documents	20.09.2023	29.09.2023	1.36w
11	Time planning	24.09.2023	29.09.2023	.86w
12	Set up Scrum Board	28.09.2023	04.10.2023	.93w
13	Overall & partial architecture	30.09.2023	07.10.2023	1.14w
14	Deployment-Einrichten	02.10.2023	07.10.2023	.86w
15	Deployment Setup	02.10.2023	14.10.2023	1.86w
16	Projects Setup	02.10.2023	14.10.2023	1.86w
17	Set up backend platform	02.10.2023	14.10.2023	1.86w
18	Set up web platform	08.10.2023	21.10.2023	2w
19	Set up app platform	08.10.2023	21.10.2023	2w
20	Set up Persistance platform	15.10.2023	21.10.2023	1w
21	Backend focus	15.10.2023	12.11.2023	4.07w
22	Desktop/Web focus	29.10.2023	25.11.2023	3.93w
23	App FoCus	29.10.2023	25.11.2023	3.93w
24	Testing Focus	05.11.2023	02.12.2023	4w
25	HotFix's	25.11.2023	09.12.2023	2.07w
26	Content completion of the documentation	03.12.2023	09.12.2023	1w
27	Revise the documentation	08.12.2023	15.12.2023	1.07w
28	Reserve	16.12.2023	22.12.2023	1w

Figure 8.1: Initial Project Plan

Planning

During the planning phase, all necessary documents and platforms are created. Additionally, the solution design is developed in this phase.

Platform

In the platform phase, the focus is on integrating the frameworks and establishing the connections and interfaces. The primary goal of this phase is not to fully define the interfaces or elaborate on all tables, but rather to implement the frameworks and test their functionalities.

MVP

During the MVP phase, the aim is to implement all functionalities to the extent that the features can be developed. For instance, by the end of this phase, all central objects should be storable in the tables, and basic push notifications should be sendable.

Feature

In the feature phase, the primary objective is to ensure that all functional requirements are meticulously and comprehensively implemented. This phase is crucial as it brings the solution to its fullest potential, ensuring that every specified functionality is translated into tangible features. If, during this process, we find that we have additional capacity—whether in terms of time, we won't just stop at the baseline requirements. Instead, we'll seize the opportunity to integrate further features, enhancing the overall value and capability of the solution. This proactive approach ensures that we maximize the output and efficiency of this phase.

Refactoring

The refactoring phase is intended to review the entire code again, ensuring that potential improvements can still be incorporated. Moreover, during this phase, it is checked whether all tests cover all edge cases.

Final Documentation

During the 'Final Documentation' phase, the primary focus is on adjusting the documentation materials to the newly created solution. While documentation is continuously updated alongside development, the emphasis in previous phases was on the creation of the solution. This phase ensures that the documentation maintains a consistent thread and allows all interested parties to trace the entire development process.

Reserve

Given that unforeseen delays can occur during the development of such a solution, we have allocated a one-week buffer. If we determine that this buffer is not needed, we will, of course, adjust the scope in a timely manner.

8.2 Project organization

8.2.1 Roles

Although, as described in the chapter, we have chosen a more or less traditional SCRUM approach in issue management, we did not allocate roles in the project according to the classical SCRUM roles. This decision was made because, given the team's size of three people, strict adherence to traditional roles is not practical. Instead, we opted for a role distribution based on the specific areas of the application: Marc Kissling for the Mobile Application, Andrew Willi for the Desktop Application, and Gian-Luca Vogel for the Backend.

8.2.2 GitHub Project Board

Our GitHub board is organized into five categories: Backlog, Sprint Backlog, In Progress, In Review, and Done. When a new issue is created, it is initially placed in the Backlog. Issues slated for specific sprints are transferred to the Sprint Backlog. Once a developer begins working on an issue, it transitions to the In Progress category. Subsequently, it moves to the In Review stage. When the issue is successfully merged into the main branch, it is then marked as Done.

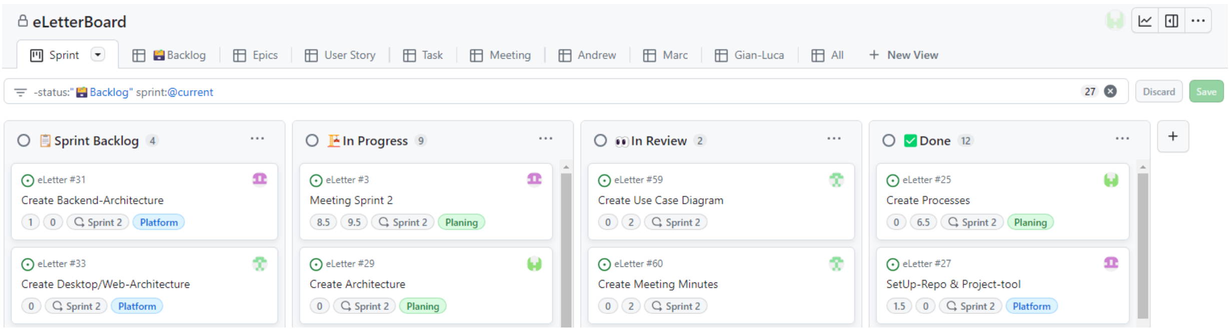


Figure 8.2: GitHub Project Board

8.2.3 Code Repository

Source code management is conducted through a GitHub repository which we set up by our self. We have chosen to centralize both the code and documentation within a monorepo, which is structured into two main folders: one for the code and another for the documentation. The code folder is further divided into three subfolders: desktop application, mobile application, and backend.

There are several reasons we decided to use a monorepo

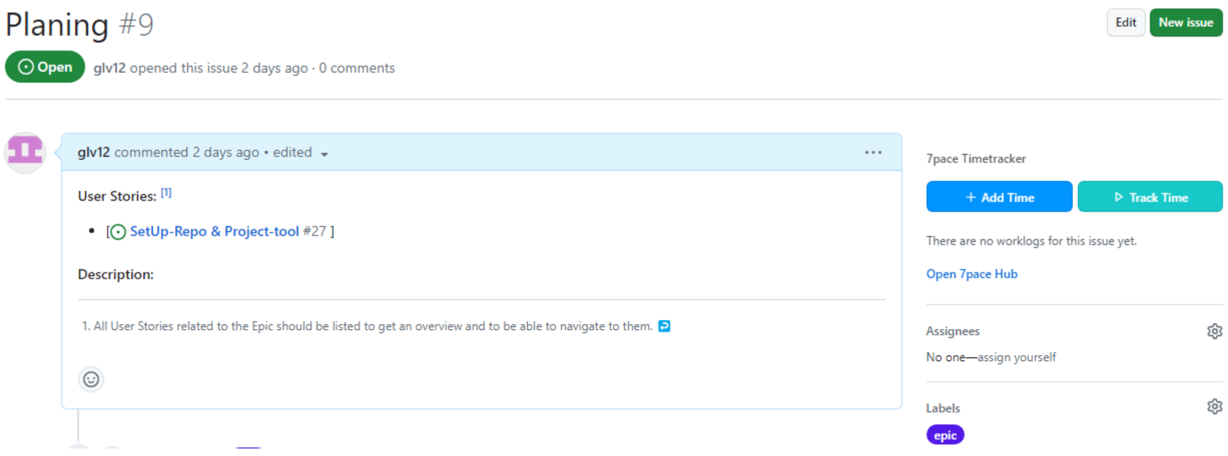
1. Simplifies keeping all code up to date with the latest version.
2. It's a lot easier to work on tasks that affect all three applications.
3. Issue management needs to be done within one repository (limitation of GitHub)

8.2.4 Issue Managment

We've chosen to organize our tasks using a classic Scrum hierarchy, which includes Epics, User Stories, and Tasks. In addition to this traditional structure, we've introduced a fourth component called "Meetings."
To address GitHub's lack of native support for hierarchical issue structures, we've implemented a solution that involves linking within the issue descriptions. This linking establishes connections between tasks, user stories, and epics, enabling smooth navigation between related issues in all directions.
To maintain consistency, we've created dedicated templates for each issue type within the repository. These templates ensure that the desired hierarchical structure is upheld across all issues. More details on the precise structure and functionality of these four elements will be provided below.

Epic

During the Planning phase, each process step is identified and defined as an Epic. An Epic is comprised of multiple user stories and can extend beyond the sprint boundaries.


Planing #9

Open glv12 opened this issue 2 days ago · 0 comments

glv12 commented 2 days ago • edited

User Stories: [1]

- [SetUp-Repo & Project-tool #27]

Description:

1. All User Stories related to the Epic should be listed to get an overview and to be able to navigate to them.

7pace Timetracker

There are no worklogs for this issue yet.

Open 7pace Hub

Assignees

No one—assign yourself

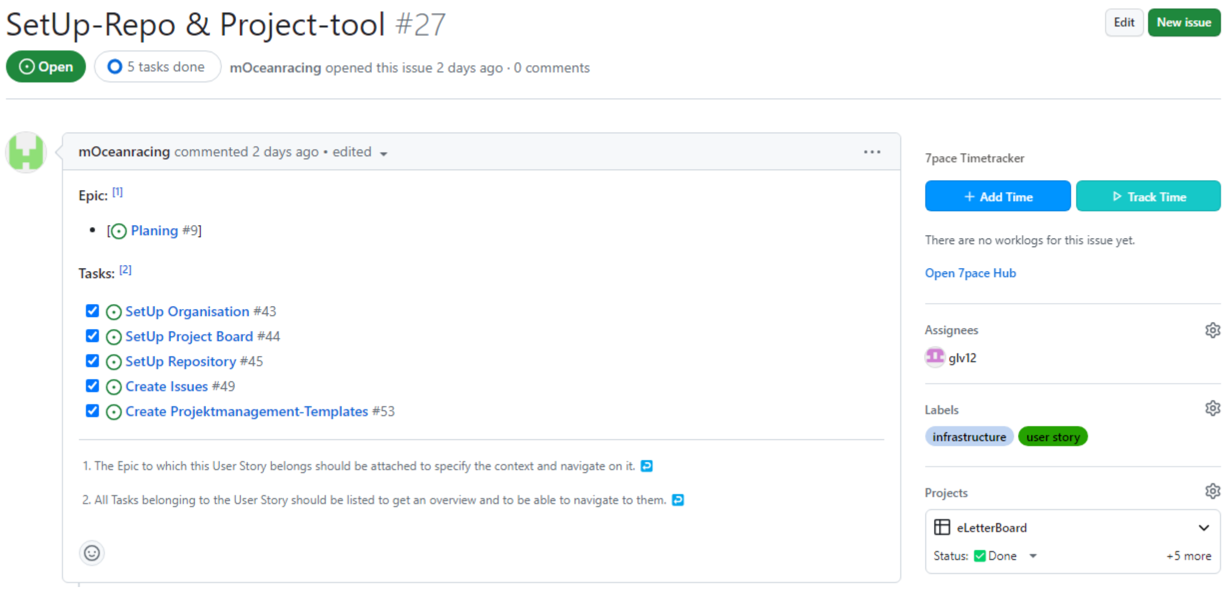
Labels

epic

Figure 8.3: GitHub Epic

User Story

A user story describes a single functionality that needs to be implemented. It itself consists of individual tasks that must be completed for successful completion. A User Story should not exceed the scope of a sprint. At the user story level, we will ensure functionality using integration testing. This differs from unit testing in that dependencies can also be tested.



SetUp-Repo & Project-tool #27

Open · 5 tasks done · mOceanracing opened this issue 2 days ago · 0 comments

mOceanracing commented 2 days ago • edited

Epic: [1]

- [Planing #9]

Tasks: [2]

- [x] SetUp Organisation #43
- [x] SetUp Project Board #44
- [x] SetUp Repository #45
- [x] Create Issues #49
- [x] Create Projektmanagement-Templates #53

1. The Epic to which this User Story belongs should be attached to specify the context and navigate on it.
2. All Tasks belonging to the User Story should be listed to get an overview and to be able to navigate to them.

7pace Timetracker
There are no worklogs for this issue yet.
Open 7pace Hub

Assignees: glv12

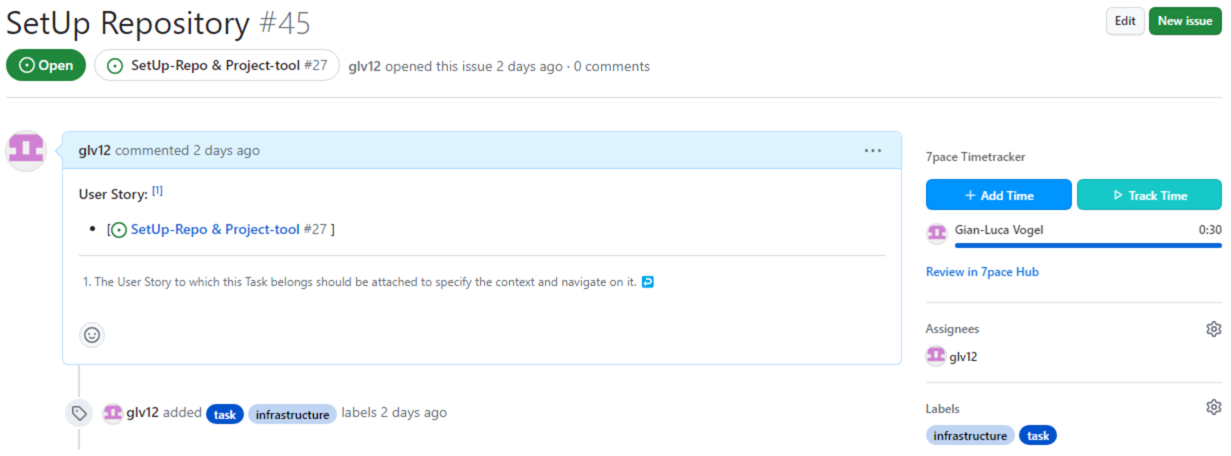
Labels: infrastructure, user story

Projects: eLetterBoard
Status: Done · +5 more

Figure 8.4: GitHub User Story

Task

A Task is a single task that cannot be further subdivided. It should be processed within a few hours and is part of a User Story. Since each task has a self-contained task, this can also be tested well using unit testing. These should not have any external dependencies.



SetUp Repository #45

Open · SetUp-Repo & Project-tool #27 · glv12 opened this issue 2 days ago · 0 comments

glv12 commented 2 days ago

User Story: [1]

- [SetUp-Repo & Project-tool #27]

1. The User Story to which this Task belongs should be attached to specify the context and navigate on it.

glv12 added task infrastructure labels 2 days ago

7pace Timetracker
Gian-Luca Vogel 0:30
Review in 7pace Hub

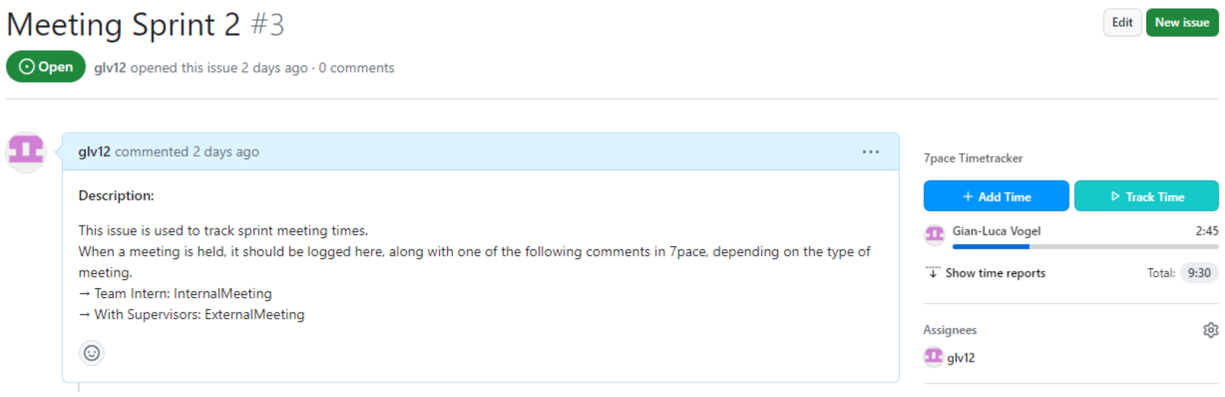
Assignees: glv12

Labels: infrastructure, task

Figure 8.5: GitHub Task

Meeting

Meetings serve the purpose of organizing and documenting our meeting time, also within the context of a sprint.


Meeting Sprint 2 #3

glv12 opened this issue 2 days ago · 0 comments

glv12 commented 2 days ago

Description:

This issue is used to track sprint meeting times.
When a meeting is held, it should be logged here, along with one of the following comments in 7pace, depending on the type of meeting.
→ Team Intern: InternalMeeting
→ With Supervisors: ExternalMeeting

7pace Timetracker

Gian-Luca Vogel 2:45

Show time reports Total: 9:30

Assignees

glv12

Figure 8.6: GitHub Meeting

8.2.5 Branching Strategy

Since each subdomain (Backend, Desktop Application, and Mobile Application) is located in its own folder, and each of us works exclusively in a specific area, we were able to apply a very simple branching strategy. Each story is divided into tasks for the respective area. A branch is then created from the main branch for this task, on which the changes for the specific area are made. Once the task is completed, the branch can be merged back into the main branch.

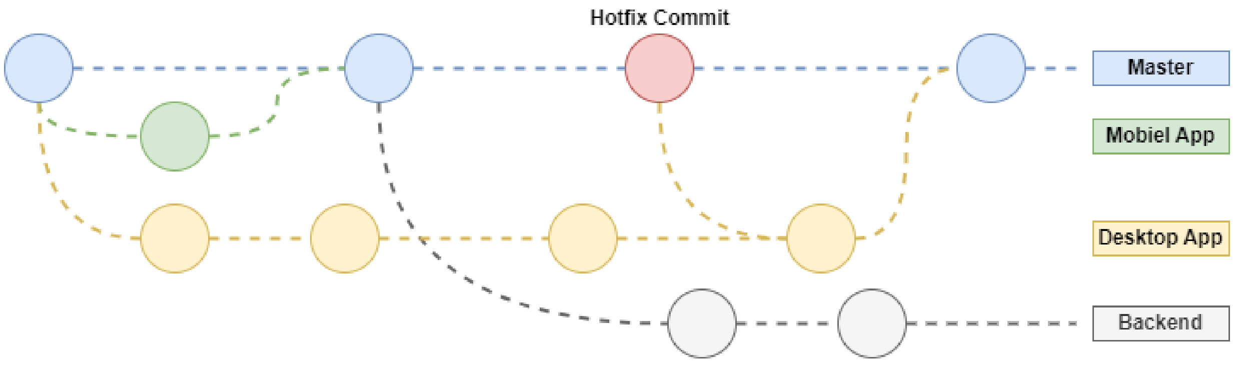


Figure 8.7: Branching Strategy

Definition of done

An issue is considered complete when it has gone through the development pipeline, passed all necessary tests, and been reviewed by at least one other member of the team. Once these steps have been completed, the branch associated with the issue must be merged. If the merge is successful and the time spent on the issue has been accurately recorded as outlined in this documentation the issue can then be marked as ”Done”.

8.2.6 Git-Workflow

The branching strategy described above results in the following workflow for all developers.

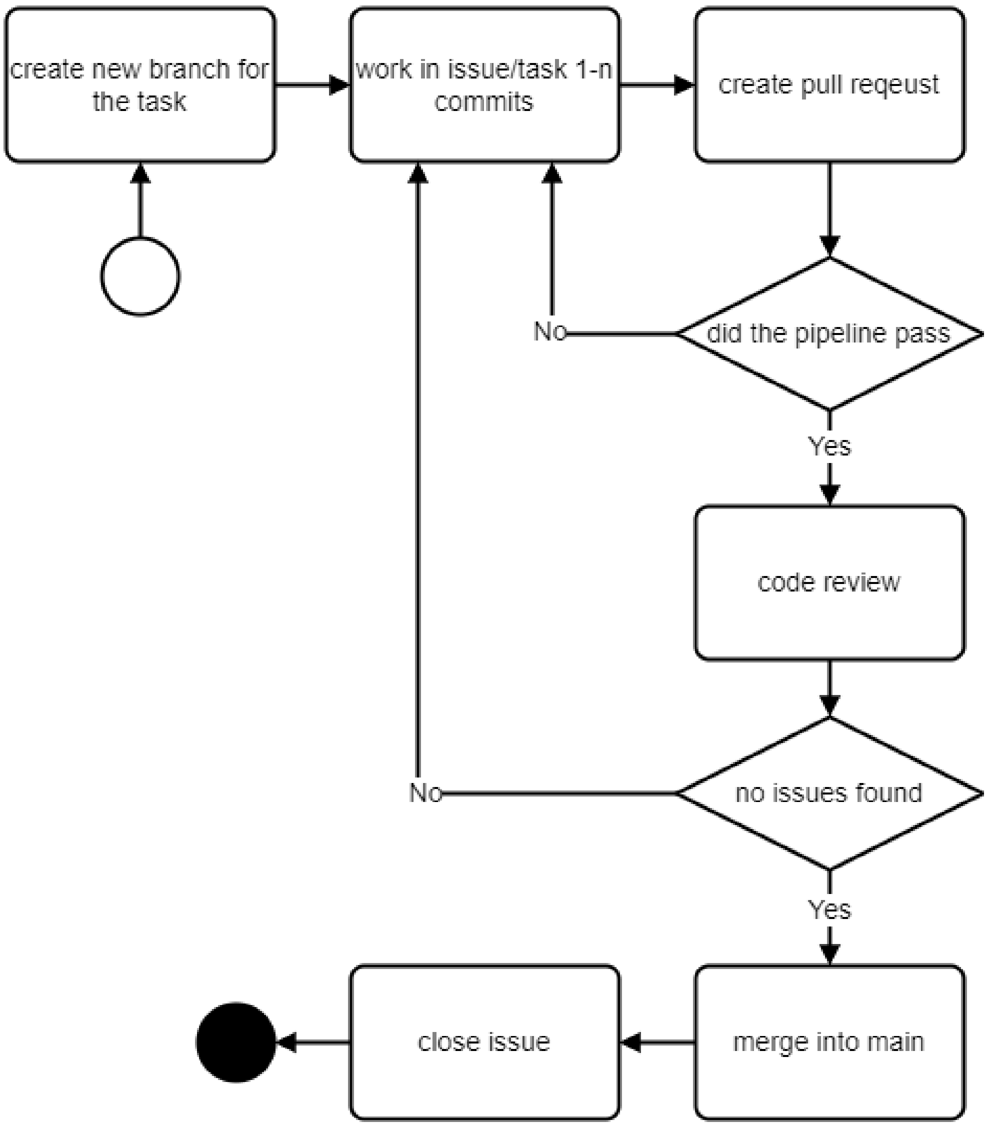


Figure 8.8: Git-Workflow

8.2.7 Risk Management

This section will address the risks to our project. If they happen, they are listed below the Riask Acceptance Graph with the associated likelihood and severity of the project. This section will be updated frequently to include new risks that arise during the project as well as dangers that have already been discovered. Additionally, the success of any countermeasures will be noted.

Risk ID	Risk	Countermeasure	Severity	Likelihood
1	Team member is unavailable	Communicate through channels to distribute tasks between other team members.	Moderate	Possible
2	Technical inexperience	Assign issues to the person best suited. Eliminate any inexperience as early as possible	Major	Likely
3	Poor project planning	Review project plan after every sprint	Major	Unlikely
4	Scope creep	Clearly communicate consequences.	Major	Possible
5	Stakeholder disagreements	Maintain transparent communication. Define decision-making protocols.	Moderate	Unlikely

Table 8.1: Project Risks

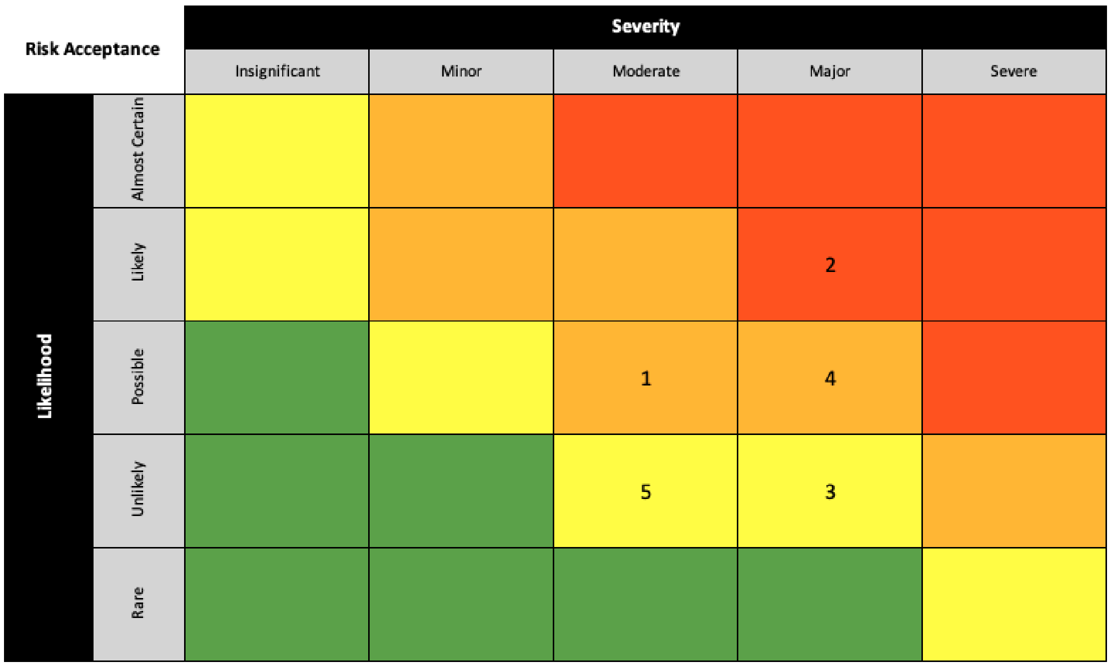


Figure 8.9: Risk Acceptance Graph

Realized risks

Team member is unavailable

A team member was unavailable for 3 days due to illness. This absence was covered by the other team members.
Severity: Moderate
Countermeasure: Successful

Technical inexperience

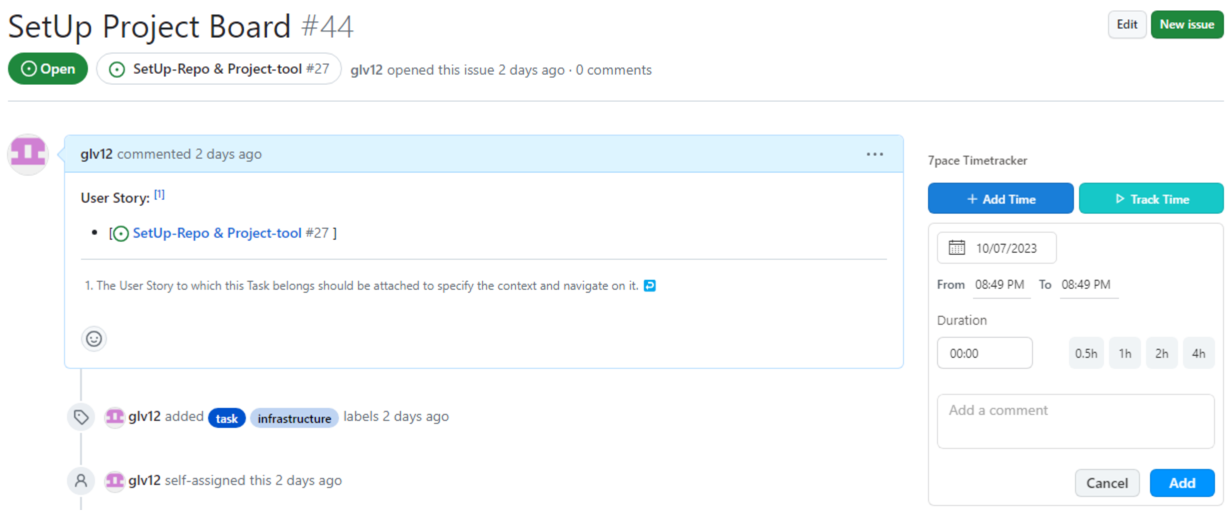
We were unfamiliar with flutter and dart. This was offset by the fact that we had plenty of time to learn the technology.
Severity: Major
Countermeasure: Successful

8.3 Time Management

Initially, we attempted time tracking using Excel, a method that our advisor and project partner did not endorse. This approach solely involved recording the actual time spent, without accounting for planned time on each issue.
Subsequently, we opted for the 7pace tool for our time tracking needs. In our view, 7pace offers three significant advantages compared to other tools:

1. The tool is free for the number of users we require.
2. It features excellent GitHub integration, allowing us to capture time directly on the issues.
3. When it comes to analysis, we can easily resort to a simple CSV format, enabling us to conduct detailed evaluations using Python as desired.

The image below shows the menu that allows you to track the time spent directly on the respective issue.


SetUp Project Board #44

Open · SetUp-Repo & Project-tool #27 · glv12 opened this issue 2 days ago · 0 comments

glv12 commented 2 days ago

User Story: [1]

- [SetUp-Repo & Project-tool #27]

1. The User Story to which this Task belongs should be attached to specify the context and navigate on it.

glv12 added task infrastructure labels 2 days ago

glv12 self-assigned this 2 days ago

7pace Timetracker

10/07/2023

From 08:49 PM To 08:49 PM

Duration 00:00

Figure 8.10: 7pace GitHub Addon

Time evaluation

The diagram below illustrates the workload of team members over the course of the project. We have divided the timeframe into the following 7 sprints:

- Sprint 1: 18.09.2023 - 01.10.2023
- Sprint 2: 02.10.2023 - 15.10.2023
- Sprint 3: 16.10.2023 - 29.10.2023
- Sprint 4: 30.10.2023 - 12.11.2023
- Sprint 5: 13.11.2023 - 26.11.2023
- Sprint 6: 27.11.2023 - 10.12.2023
- Sprint 7: 11.12.2023 - 22.12.2023

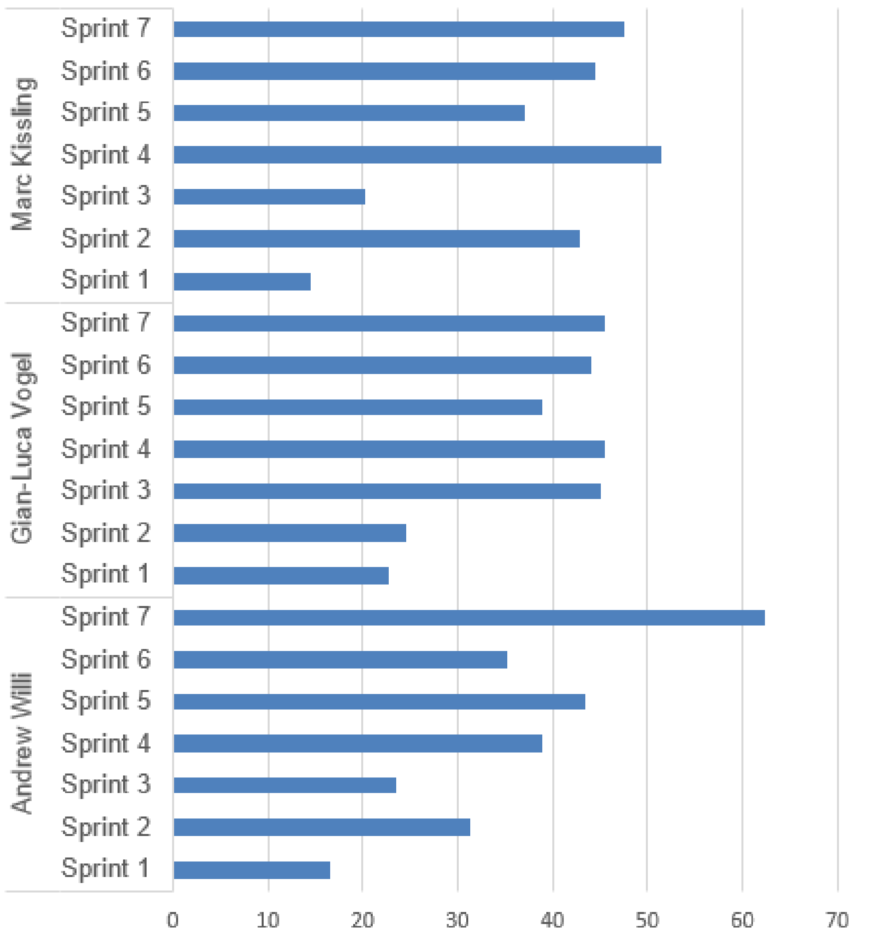


Figure 8.11: Bar chart distribution working hour per sprint area per sprint

The total workload per team member is as follows:

- Andrew Willi: 251.3 hours
- Marc Kissling: 258.05 hours
- Gian-Luca Vogel: 266.25 hours

The pie chart below displays the distribution of working hours in this project by area. As can be observed, the distribution is essentially equal across all sub-applications. The slightly greater share in the Mobile App category is owing to our lack of experience in this domain, which necessitated additional time for research. The database has a tiny percentage, denoted as 0% in the chart, because we used an ORM Mapper that successfully abstracts and maintains the database. As a result, database-related tasks are also included in the Backend part. At first look, the 9% share for meetings may appear excessive, however it should be emphasized that this corresponds to only 3% per team member.

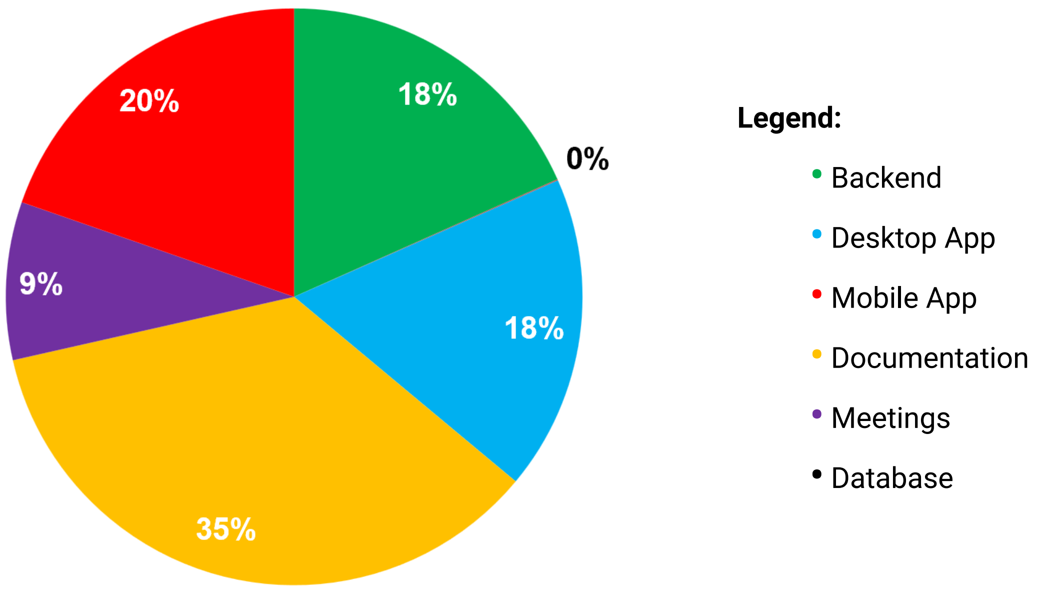


Figure 8.12: Pie chart distribution working hours per application area

The following bar chart shows the distribution of work per project area across various sprints. The first noticeable aspect is the significant effort invested in documentation during Sprint 7. This was planned from the outset, as it was crucial for us to implement features after planning the architecture before dedicating time to document everything. The second noteworthy observation is that substantial work on the mobile application occurred relatively late, specifically in Sprint 4. This is attributed to the initial need to acquaint ourselves with Flutter and the additional time required for creating mockups for the mobile app. As evident, in contrast to the other applications, we initiated work on the backend in the first sprint. This was necessary as the other two applications were dependent on the backend.

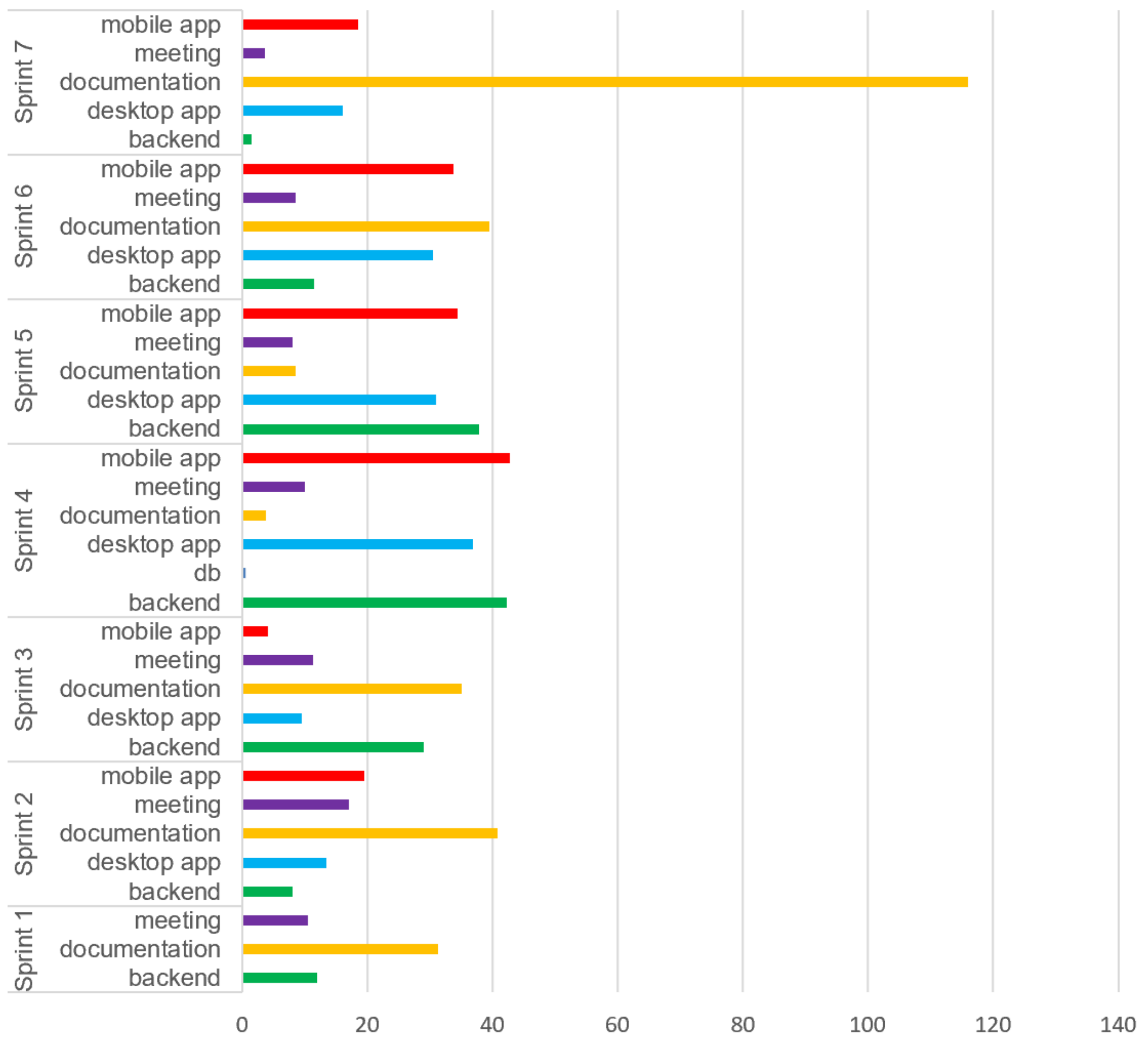


Figure 8.13: Bar chart distribution working hour per application area per sprint

The detailed figures for the time evaluations can be found in the appendix under the chapter time evaluation numbers

9 Acknowledgement

We would like to express our thanks to our supervisor Frank Koch, who took the time and was always available for questions. Furthermore, we would also like to thank our industry partner AdaptIT GmbH, specifically Michael Güntensperger, who granted us great freedom in developing the application and provided the necessary resources.

List of Figures

List of Tables

10 Appendix

10.1 API Documentation

Authentication Controller

The authentication controller is responsible for all authentication-related endpoints such as sign in, sign up, and sign out.
Base-URL: /auth
Endpoints:

- **POST /signIn/user**: Sign in an existing user with email and password

  Body: JSON

  ```
  {
   "email": String,
   "password": String,
   "deviceToken": String
  }
  ```

  Listing 10.1: Body of the signIn/user request

  Response: String

  USER_ID

- **POST /signUp/user**: Sign up an user with given data in the body

  Body: JSON

  ```
  {
   "firstName": String,
   "lastName": String,
   "email": String,
   "password": String,
   "street": String,
   "streetNumber": String,
   "postalCode": Number,
   "city": String,
   "state": String
  }
  ```

  Listing 10.2: Body of the signUp/user request

  Response: String

  Signe up was successful

- **GET /signOut/user**: Sign out an user

  Body: none

  Response: String

  Logged out successfully

- **POST /signIn/organisation**: Sign in an organisation with email and password

  Body: JSON

```
{
 "email": String,
 "password": String,
}
```

Listing 10.3: Body of the signIn/organisation request

  Response: String

  ORGANISATION_ID

- **POST /signUp/organisation**: Sign up an organisation with given data in the body

  Body: Form-Data

Key	Value
file	Buffer
name	String
email	String
password	String
street	String
streetNumber	String
postalCode	String
city	String
state	String

Table 10.1: Body of the signUp/organisation request

  Response: String

  Signe up was successful

- **GET /signOut/organisation**: Sign out an organisation

  Body: none

  Response: String

  Logged out successfully

Hello Controller

The hello controller contains only one endpoint, which is used to check whether the backend is running.
Base-URL: /hello
Endpoint:

- **GET /**: Check if the backend is running

  Body: none

  Response: String

  Hello World!

User Controller

The user controller is responsible for all endpoints related to the user, such as updating the user settings and the address verification process.
Base-URL: /users
Endpoints:

- **GET /settings/<USER_ID>**: Get user settings

  Body: none

  Response: JSON

```
{
 "id": String,
 "firstName": String,
 "lastName": String,
 "street": String,
 "streetNumber": String,
 "postalCode": Number,
 "city": String,
 "state": String
 "addressVerified": boolean;
}
```

Listing 10.4: Response of the users/settings/ get request

- **PUT /settings/**: Update user settings

  Body: JSON

```
{
 "id": String,
 "firstName": String,
 "lastName": String,
 "street": String,
 "streetNumber": String,
 "postalCode": Number,
 "city": String,
 "state": String
}
```

Listing 10.5: Body of the users/settings/ put request

Response: JSON

```
{
 "id": String,
 "firstName": String,
 "lastName": String,
 "street": String,
 "streetNumber": String,
 "postalCode": Number,
 "city": String,
 "state": String
 "addressVerified": boolean;
}
```

Listing 10.6: Body of the users/settings/ put request

- **Patch /verifications/?userId=<USER_ID>&verificationCode=<CODE>**: It verifies the address of the user

  Body: none

Response: String
User was successfully verified

Organisation Controller

The organisation controller is responsible for all endpoints related to the organisation, such as updating the organisation settings and the address verification process.
Base-URL: /organisations
Endpoints:

- **GET /settings/<ORGANISATION_ID>**: Get organisation settings

  Body: none

  Response: JSON

```
{
 "id": String,
 "name": String,
 "email": String,
 "street": String,
 "streetNumber": String,
 "postalCode": Number,
 "city": String,
 "state": String
 "addressVerified": Boolean
 "imageURL": String (pre-signed URL)
}
```

Listing 10.7: Response of the organisations/settings/ get request

- **PUT /settings/**: Update user settings

  Body: Form-Data

Key	**Value**
id	String
file	String
email	String
name	String
street	String
streetNumber	String
postalCode	String
city	String
state	String

Table 10.2: Body of the organisations/settings/ request

  Response: JSON

```
{
 "id": String,
 "name": String,
 "email": String,
 "street": String,
 "streetNumber": String,
 "postalCode": Number,
 "city": String,
 "state": String
 "addressVerified": Boolean
 "imageURL": String (pre-signed URL)
}
```

Listing 10.8: Response of the organisations/settings/ get request

- **Patch /verifications/?userId=<ORGANISATION_ID>&verificationCode=<CODE>**: It verifies the address of the organisation

  Body: none

  Response: String

  Organisation was successfully verified

Letter Controller

The letter controller is responsible for all endpoints related to the postal service.
Base-URL: /letters
Endpoints:

- **POST /send**: It sends a letter

  Body: Form-Data

Key	Value
file	Buffer
firstName	String
lastName	String
streetNumber	String
streetNumber	String
postalCode	String
city	String
state	String
organisationId	String

Table 10.3: Body of the letters/send request

  Response: String Send letter digitally or Send letter physically

- **GET /byUser/<USER_ID>**: It gets all letters of an user

  Body: none

  Response: JSON

```
[
 {
  "letterid": String,
  "date": Date,
  "senderName": String,
  "organisatonImage": String (pre-signed URL),
  "status": String,
 },
 ...
]
```

Listing 10.9: Response of the letters/byUser/ get request

- **GET /detailsOf/<LETTER_ID>**: It gets the letter data of a letter

  Body: none

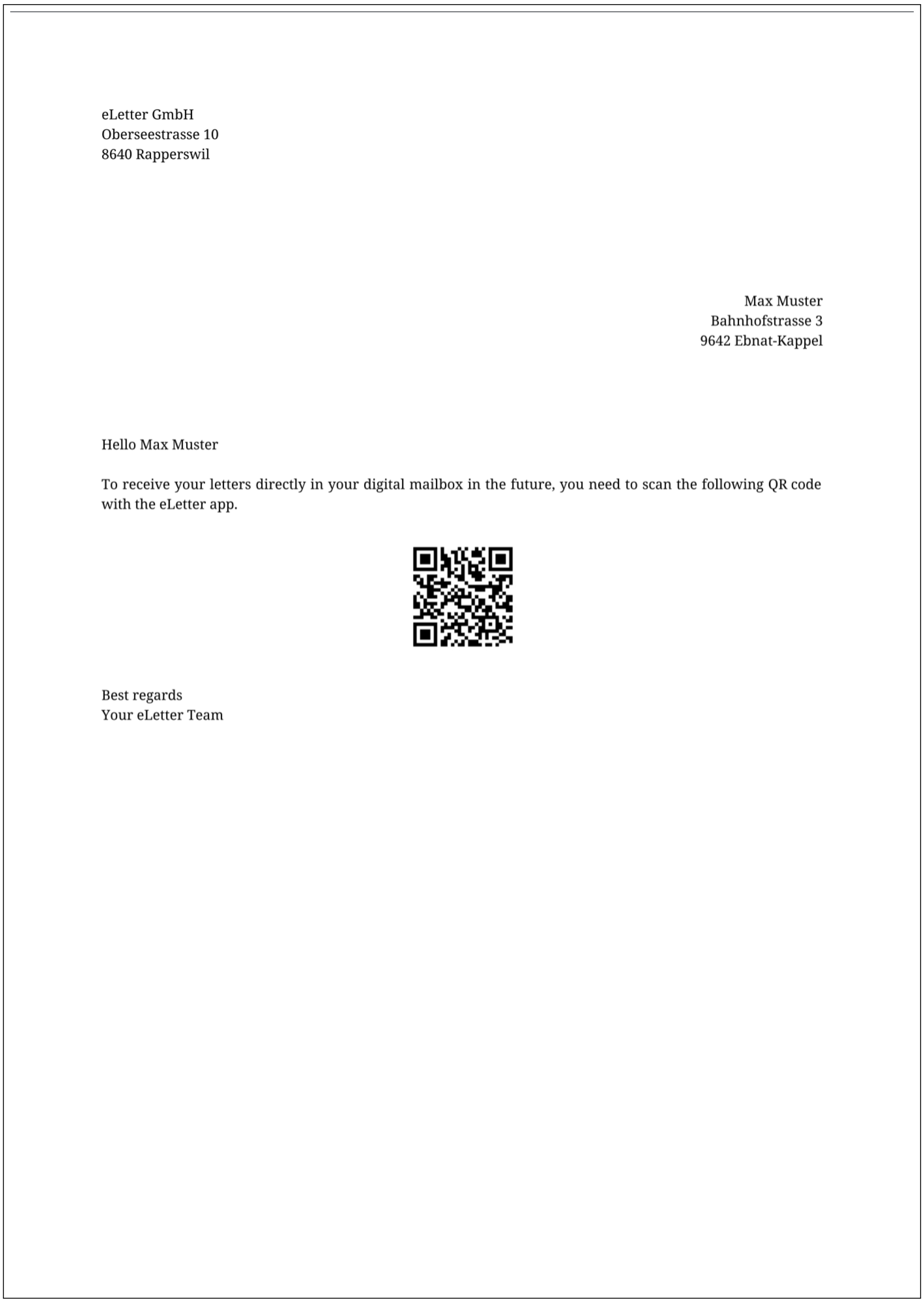
  Response: JSON

```
{
 "id": String,
 "date": Date,
 "downloadUrl": String (pre-signed URL),
 "senderName": String,
 "fileName": String,
 "firstPageURL": String (pre-signed URL),
 "recipientId": String
}
```

Listing 10.10: Response of the letters/detailsOf/ get request

10.2 Screenshots

Verification Letters



eLetter GmbH
Oberseestrasse 10
8640 Rapperswil

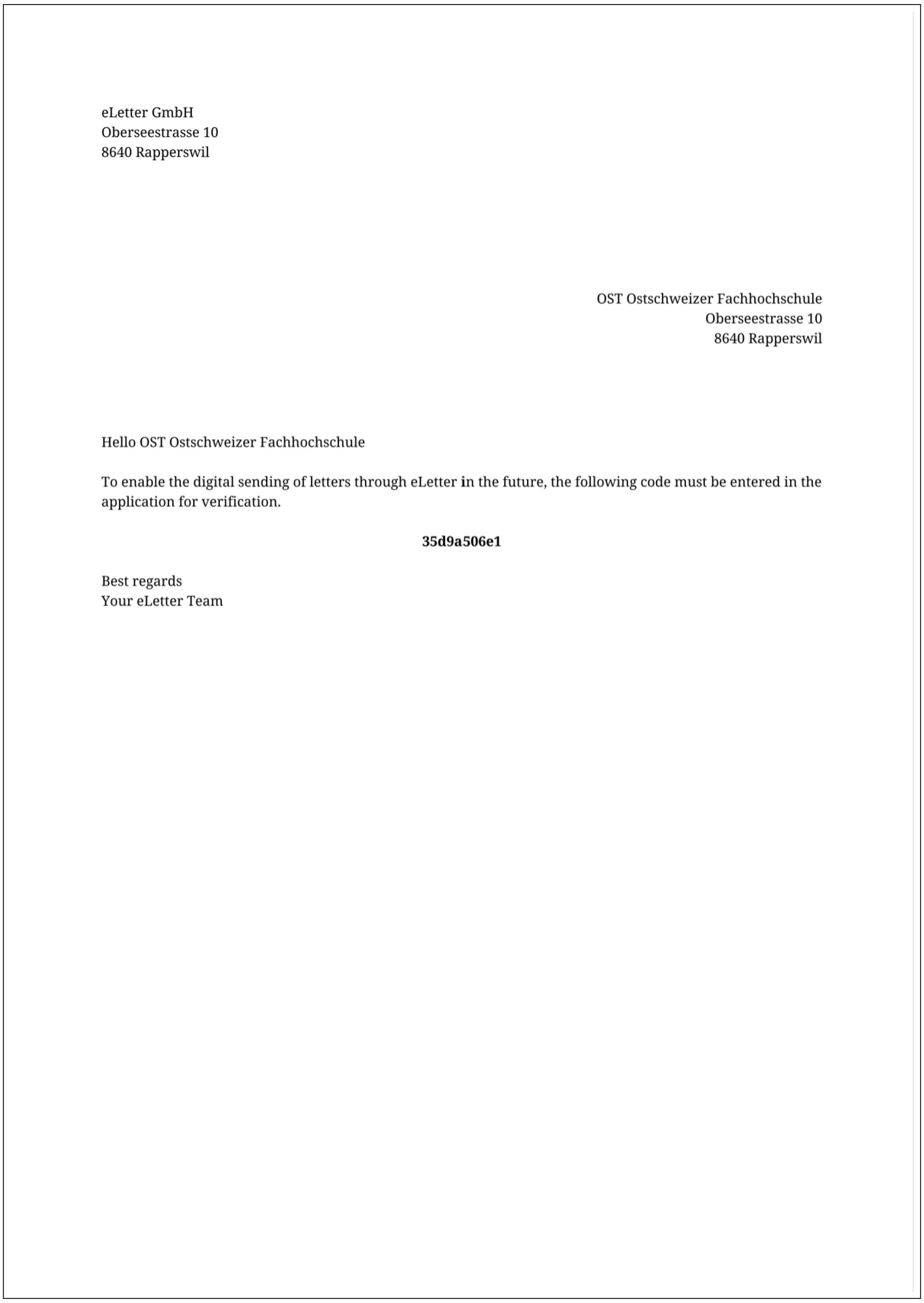
Max Muster
Bahnhofstrasse 3
9642 Ebnat-Kappel

Hello Max Muster

To receive your letters directly in your digital mailbox in the future, you need to scan the following QR code with the eLetter app.

Best regards
Your eLetter Team

Figure 10.1: Screenshot of the user verification letter



eLetter GmbH
Oberseestrasse 10
8640 Rapperswil

OST Ostschweizer Fachhochschule
Oberseestrasse 10
8640 Rapperswil

Hello OST Ostschweizer Fachhochschule

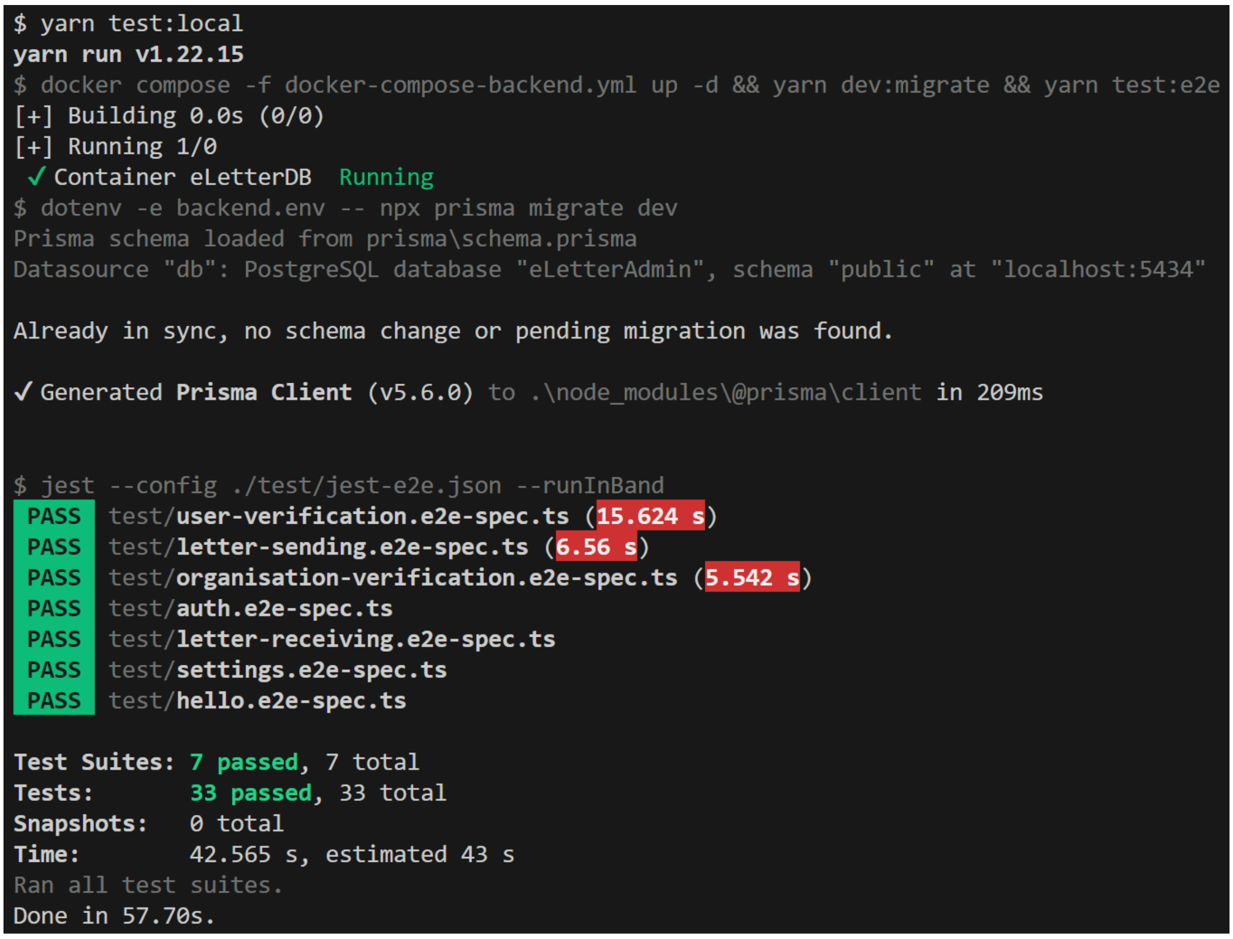
To enable the digital sending of letters through eLetter in the future, the following code must be entered in the application for verification.

35d9a506e1

Best regards
Your eLetter Team

Figure 10.2: Screenshot of the organisation verification letter

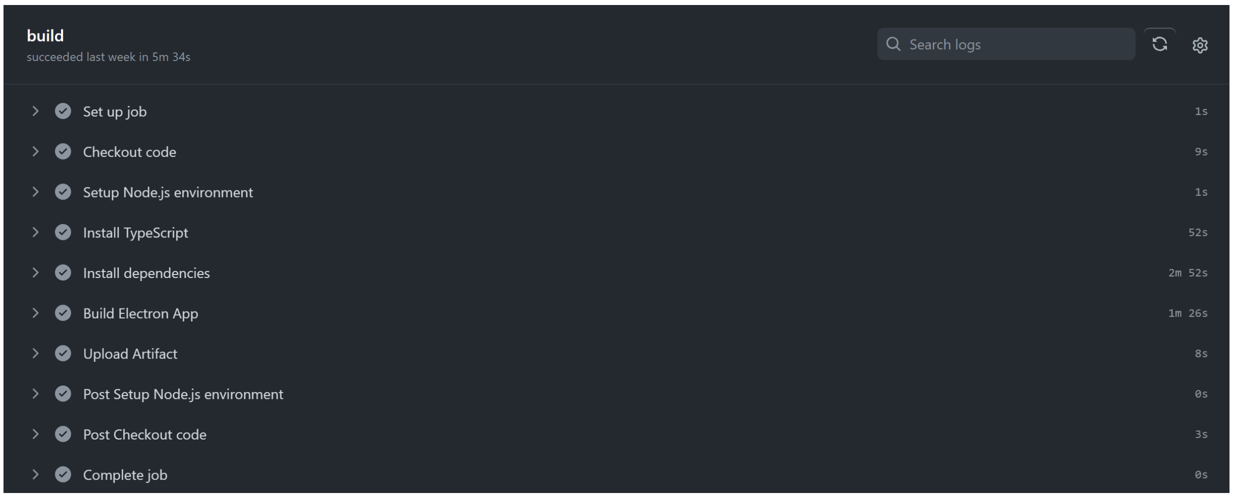
Backend test run



```
$ yarn test:local
yarn run v1.22.15
$ docker compose -f docker-compose-backend.yml up -d && yarn dev:migrate && yarn test:e2e
[+] Building 0.0s (0/0)
[+] Running 1/0
 ✓ Container eLetterDB  Running
$ dotenv -e backend.env -- npx prisma migrate dev
Prisma schema loaded from prisma\schema.prisma
Datasource "db": PostgreSQL database "eLetterAdmin", schema "public" at "localhost:5434"

Already in sync, no schema change or pending migration was found.

✓ Generated Prisma Client (v5.6.0) to .\node_modules\@prisma\client in 209ms


$ jest --config ./test/jest-e2e.json --runInBand
 PASS  test/user-verification.e2e-spec.ts (15.624 s)
 PASS  test/letter-sending.e2e-spec.ts (6.56 s)
 PASS  test/organisation-verification.e2e-spec.ts (5.542 s)
 PASS  test/auth.e2e-spec.ts
 PASS  test/letter-receiving.e2e-spec.ts
 PASS  test/settings.e2e-spec.ts
 PASS  test/hello.e2e-spec.ts

Test Suites: 7 passed, 7 total
Tests:       33 passed, 33 total
Snapshots:   0 total
Time:        42.565 s, estimated 43 s
Ran all test suites.
Done in 57.70s.
```

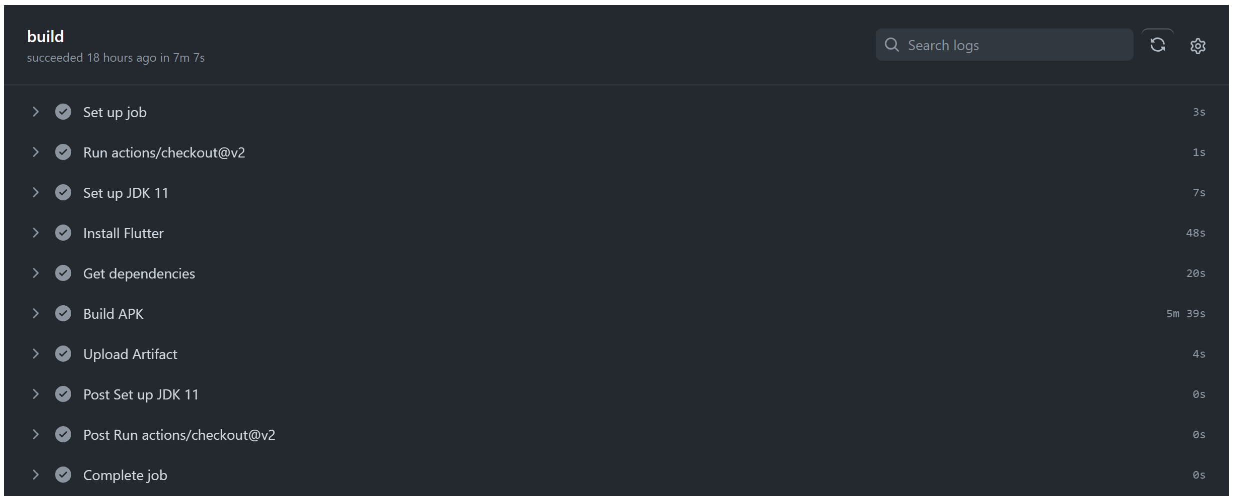
Figure 10.3: Backend Test Run

GitHub Actions runs



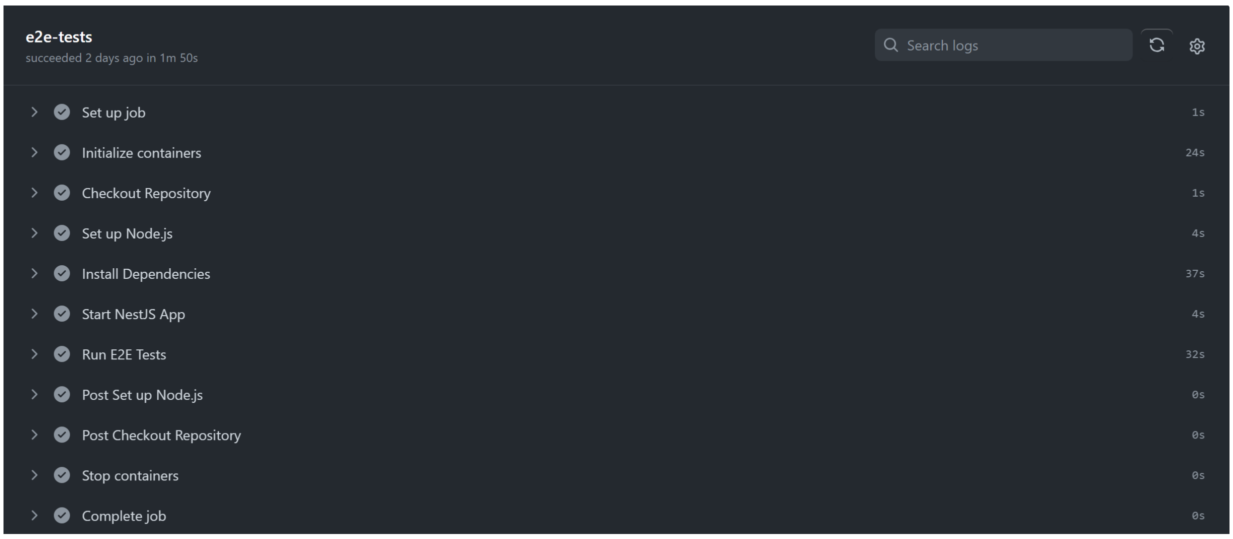
```
build
succeeded last week in 5m 34s

> ✓ Set up job                        1s
> ✓ Checkout code                     9s
> ✓ Setup Node.js environment         1s
> ✓ Install TypeScript                52s
> ✓ Install dependencies              2m 52s
> ✓ Build Electron App                1m 26s
> ✓ Upload Artifact                   8s
> ✓ Post Setup Node.js environment    0s
> ✓ Post Checkout code                3s
> ✓ Complete job                      0s
```

Figure 10.4: Desktop Application Build



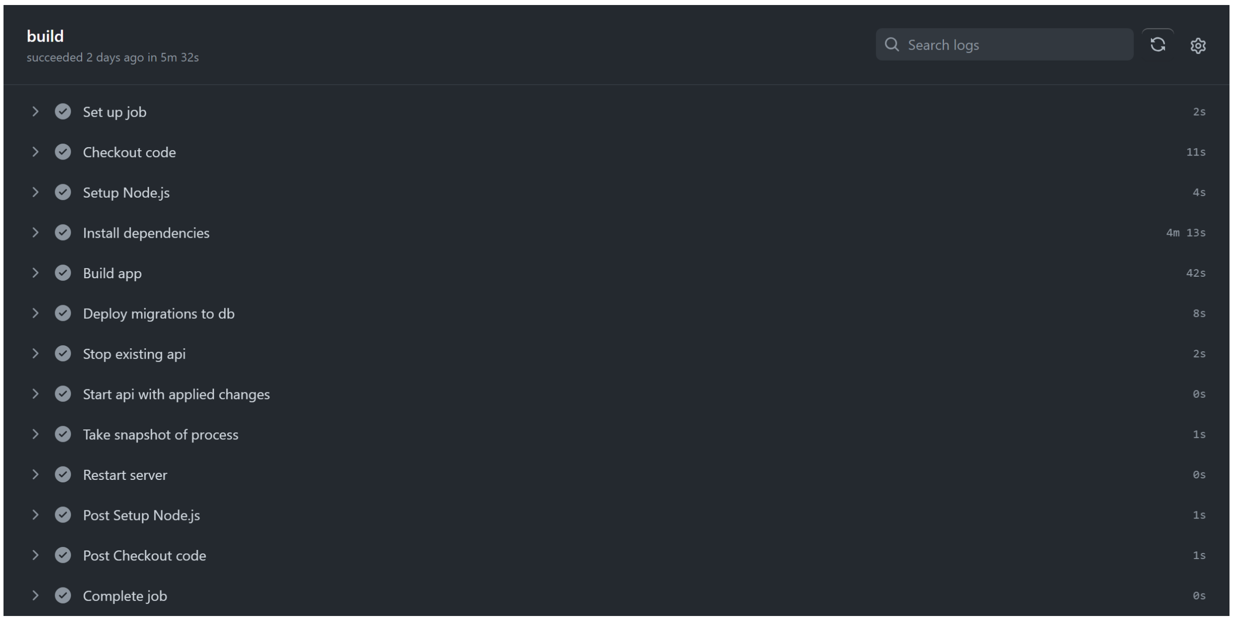
```
build
succeeded 18 hours ago in 7m 7s

Set up job                          3s
Run actions/checkout@v2             1s
Set up JDK 11                       7s
Install Flutter                    48s
Get dependencies                   20s
Build APK                       5m 39s
Upload Artifact                     4s
Post Set up JDK 11                  0s
Post Run actions/checkout@v2        0s
Complete job                        0s
```

Figure 10.5: Mobile Application Build



```
e2e-tests
succeeded 2 days ago in 1m 50s

Set up job                          1s
Initialize containers              24s
Checkout Repository                 1s
Set up Node.js                      4s
Install Dependencies               37s
Start NestJS App                    4s
Run E2E Tests                      32s
Post Set up Node.js                 0s
Post Checkout Repository            0s
Stop containers                     0s
Complete job                        0s
```

Figure 10.6: Backend Test run GitHub Actions



```
build
succeeded 2 days ago in 5m 32s

> Set up job                        2s
> Checkout code                     11s
> Setup Node.js                     4s
> Install dependencies              4m 13s
> Build app                         42s
> Deploy migrations to db           8s
> Stop existing api                 2s
> Start api with applied changes    0s
> Take snapshot of process          1s
> Restart server                    0s
> Post Setup Node.js                1s
> Post Checkout code                1s
> Complete job                      0s
```

Figure 10.7: Backend Build

Desktop Application

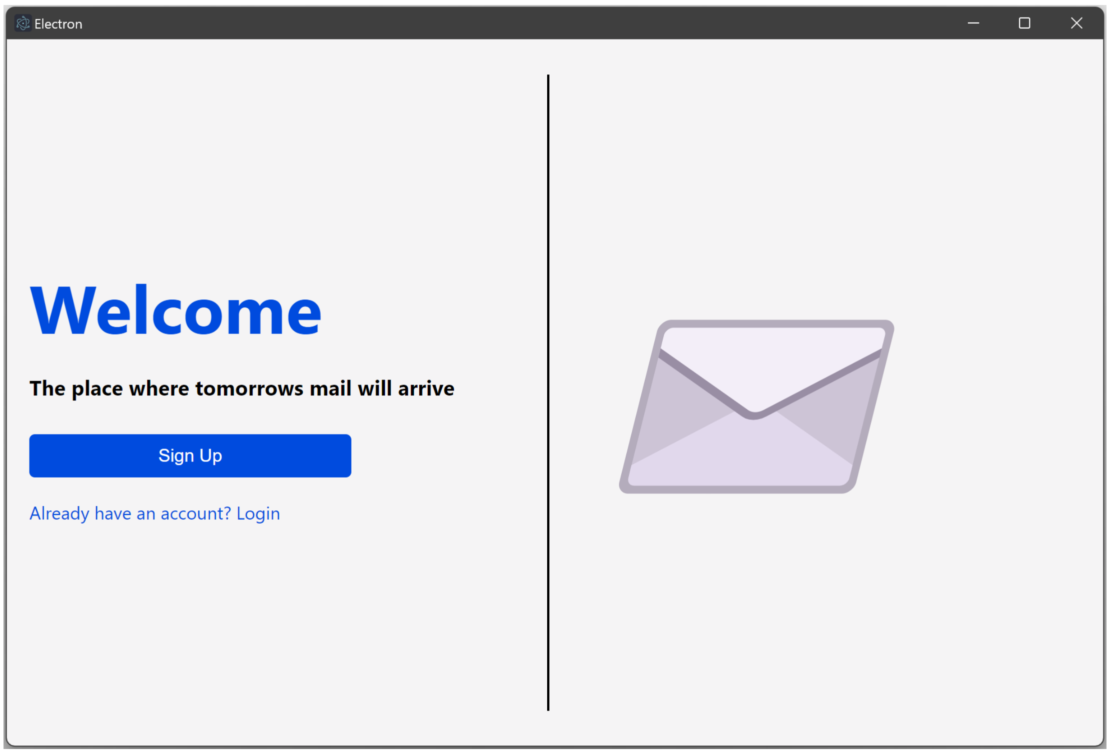
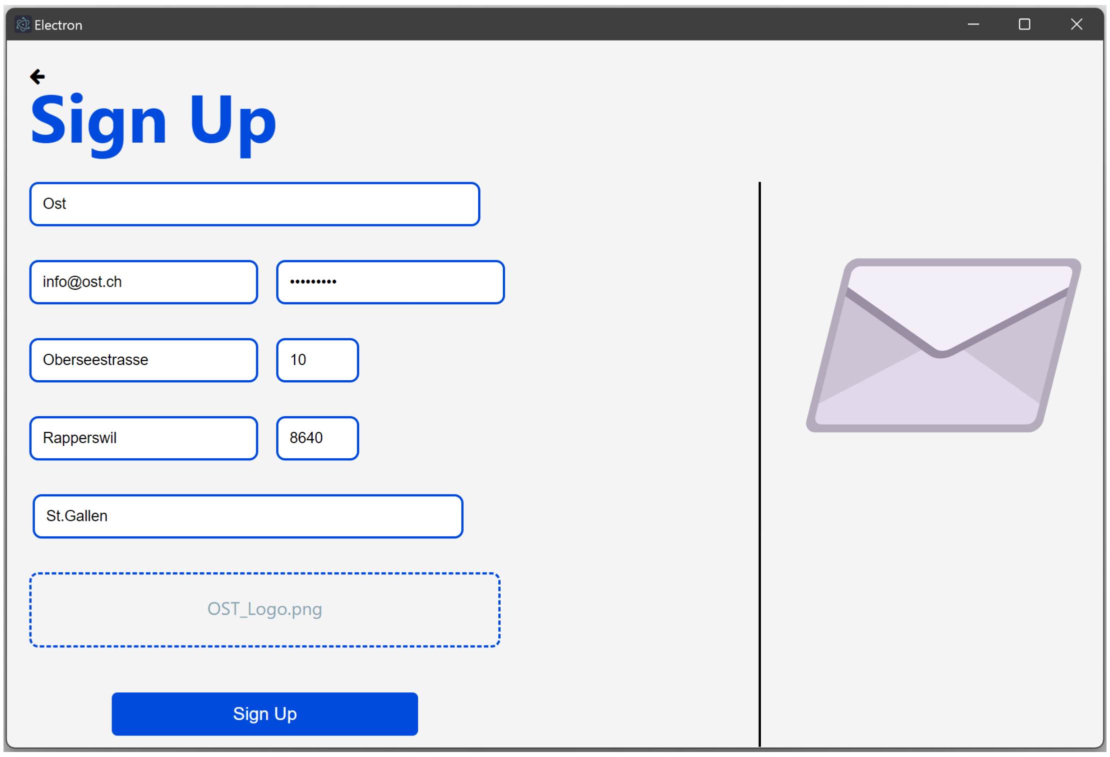

Electron
Welcome
The place where tomorrows mail will arrive
Sign Up
Already have an account? Login

Figure 10.8: Desktop application welcome page



Ost

info@ost.ch

•••••••••

Oberseestrasse

10

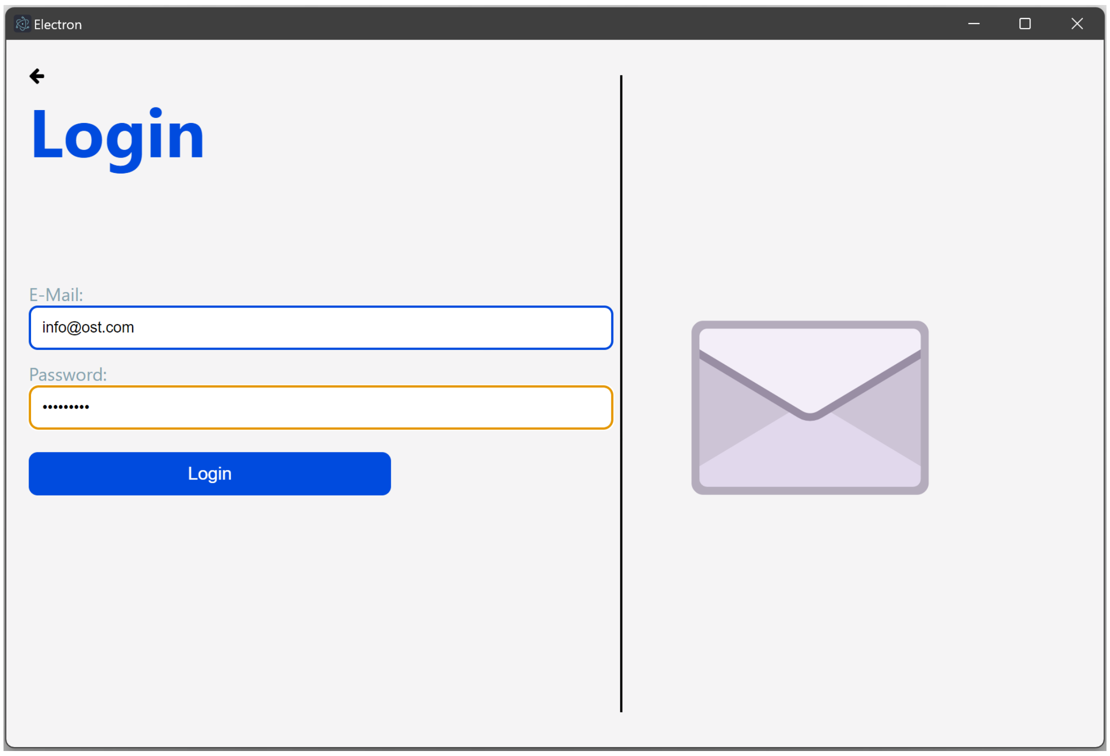
Rapperswil

8640

St.Gallen

OST_Logo.png

Figure 10.9: Desktop application sign up page



E-Mail: info@ost.com

Password: •••••••••

Figure 10.10: Desktop application sign in page

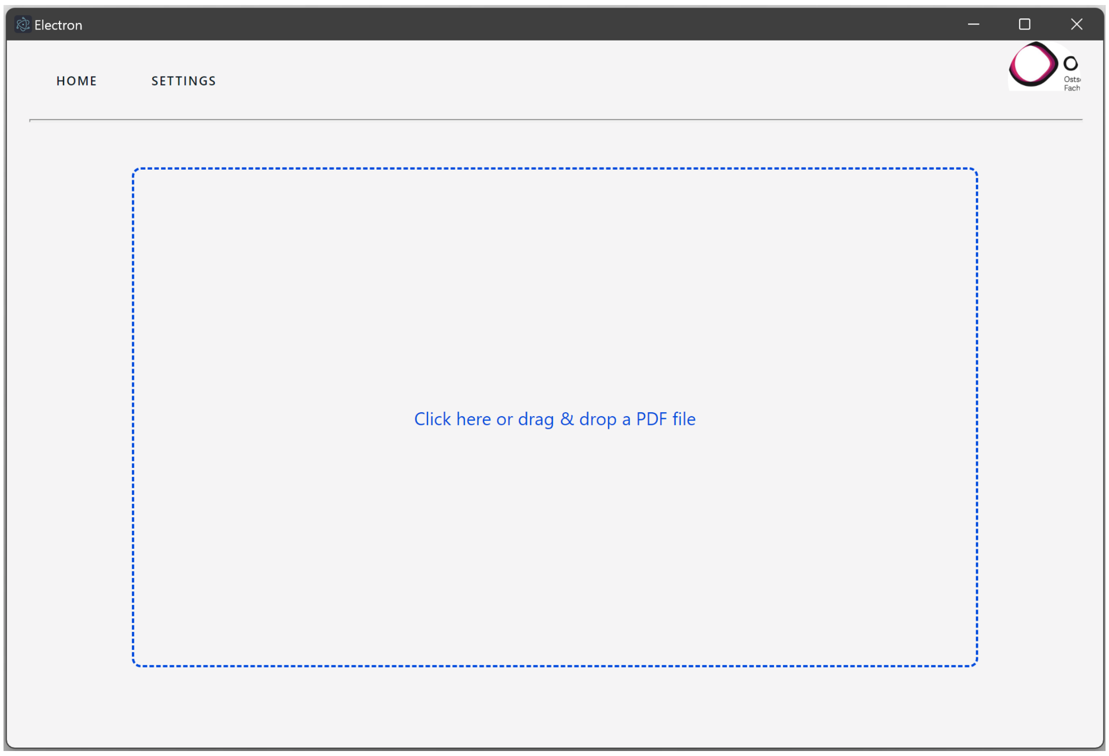


Figure 10.11: Desktop application home page

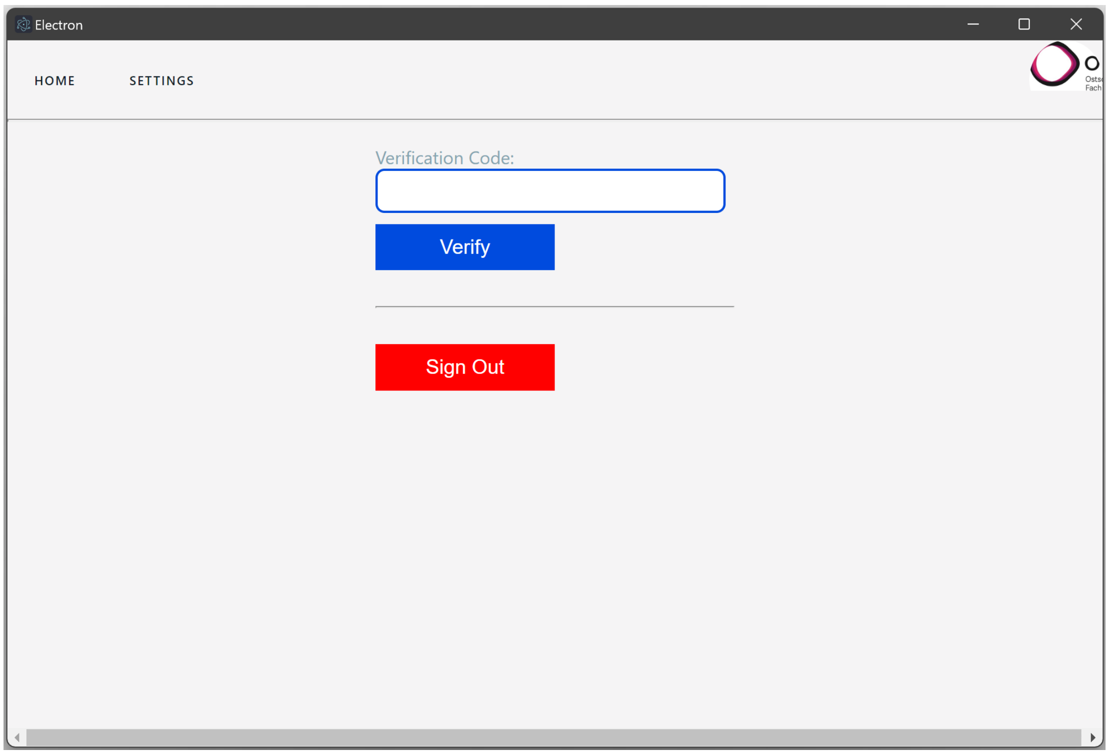
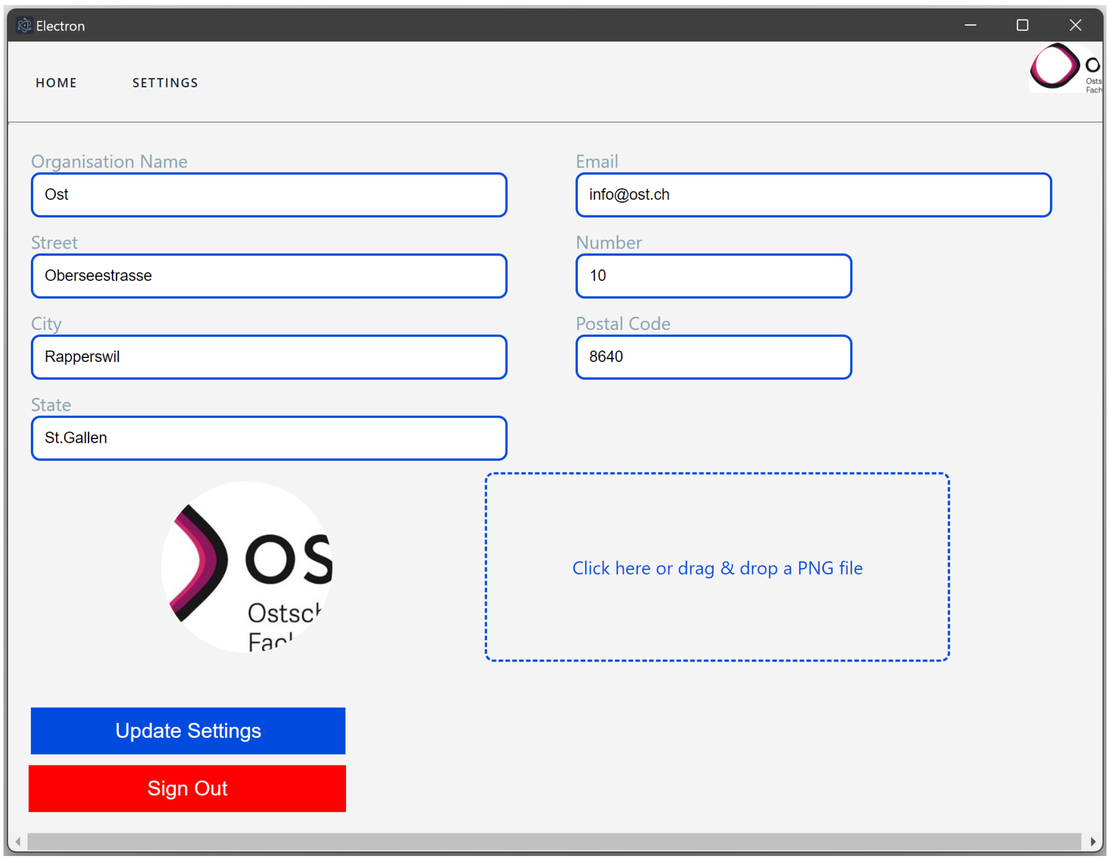


Figure 10.12: Desktop application verification page



Electron

Organisation Name: Ost	Email: info@ost.ch
Street: Oberseestrasse	Number: 10
City: Rapperswil	Postal Code: 8640
State: St.Gallen	

Figure 10.13: Desktop application settings page

Mobile Application

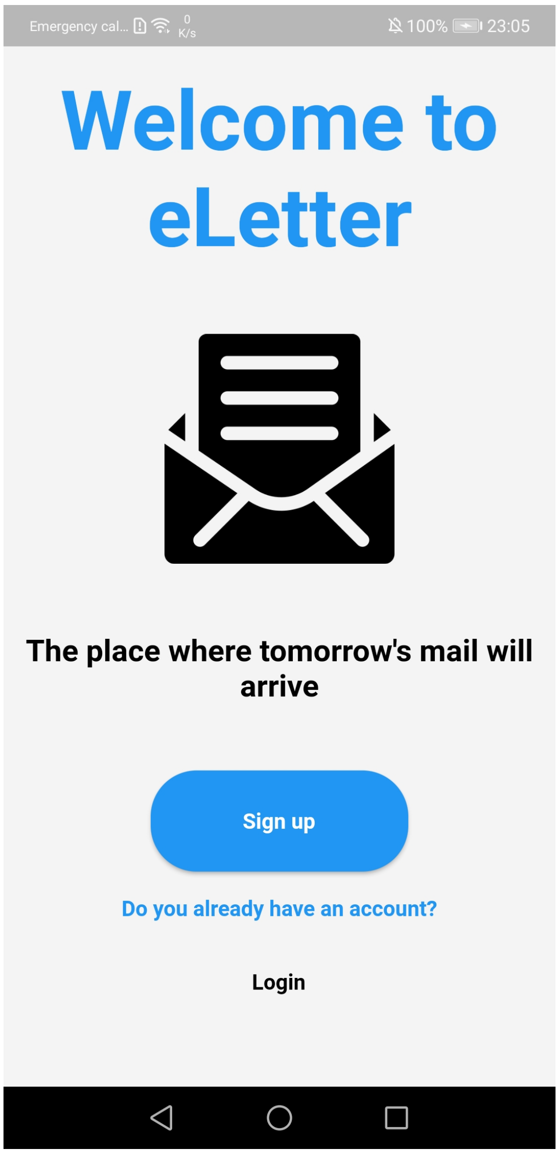
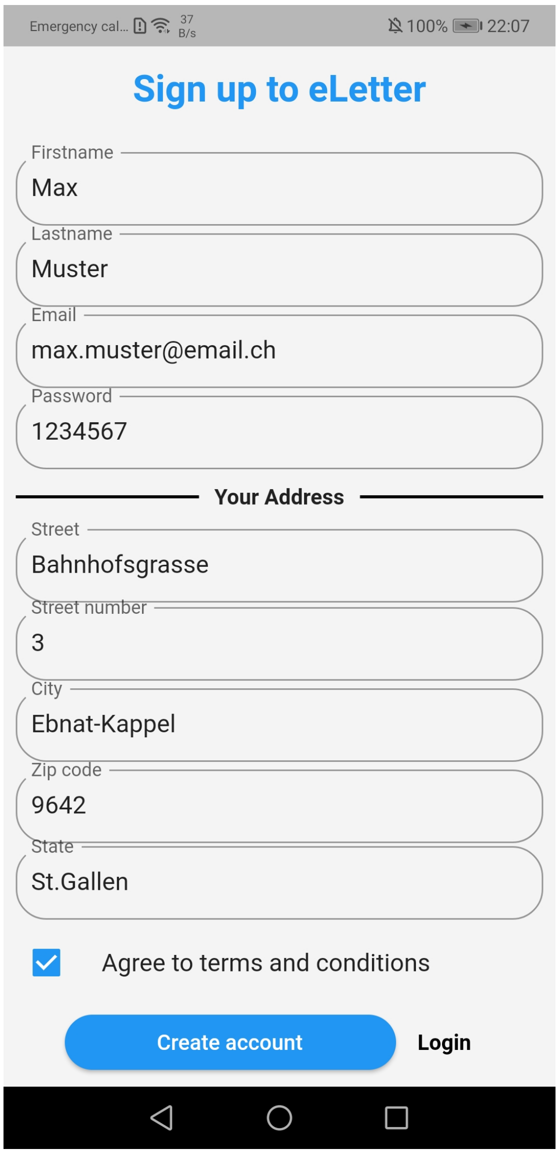

Emergency cal...
100% 23:05
Welcome to eLetter
The place where tomorrow's mail will arrive
Sign up
Do you already have an account?
Login

Figure 10.14: Mobile application welcome page



Sign up to eLetter

Firstname: Max

Lastname: Muster

Email: max.muster@email.ch

Password: 1234567

Your Address

Street: Bahnhofsgrasse

Street number: 3

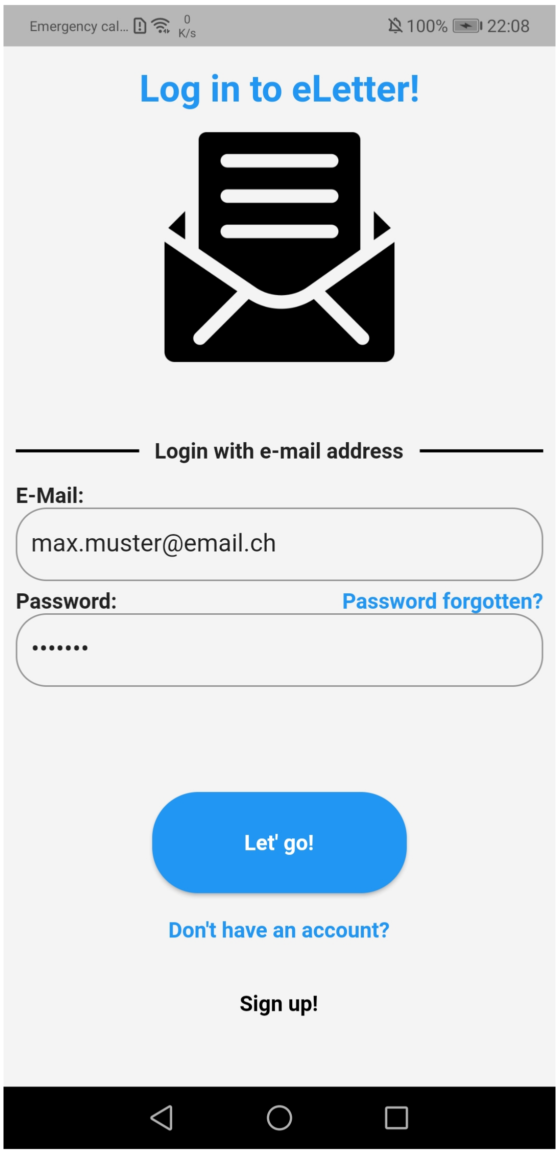
City: Ebnat-Kappel

Zip code: 9642

State: St.Gallen

[✓] Agree to terms and conditions

Figure 10.15: Mobile application sign up page


Emergency cal... 0 K/s 100% 22:08

Log in to eLetter!

Login with e-mail address

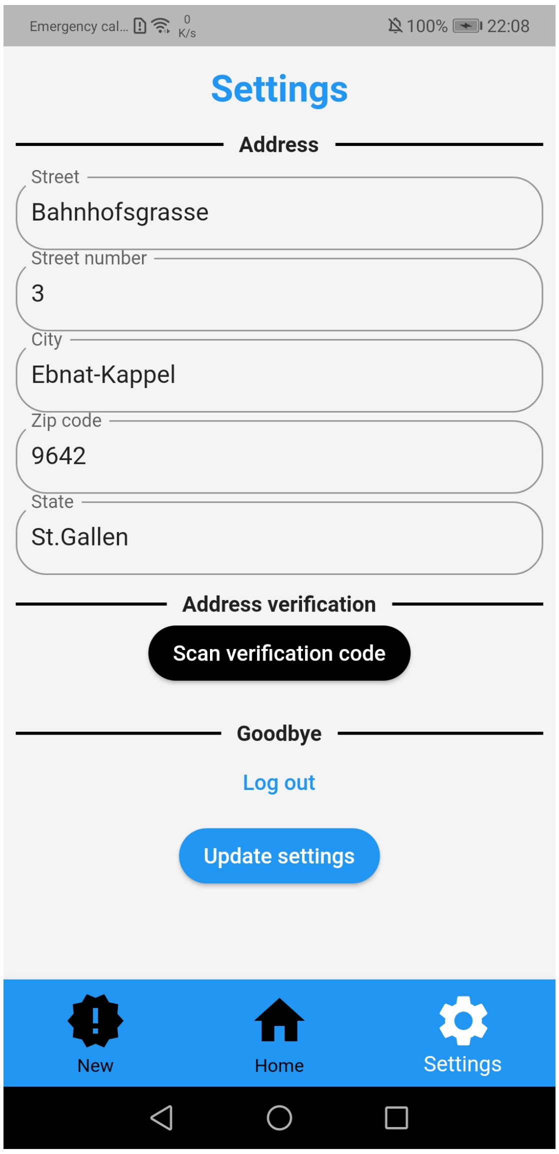
E-Mail:

max.muster@email.ch

Password: Password forgotten?

•••••••

Figure 10.16: Mobile application login page


Emergency cal... 0 K/s 100% 22:08

Settings

Address

Street: Bahnhofsgrasse

Street number: 3

City: Ebnat-Kappel

Zip code: 9642

State: St.Gallen

Address verification

Figure 10.17: Mobile application settings page

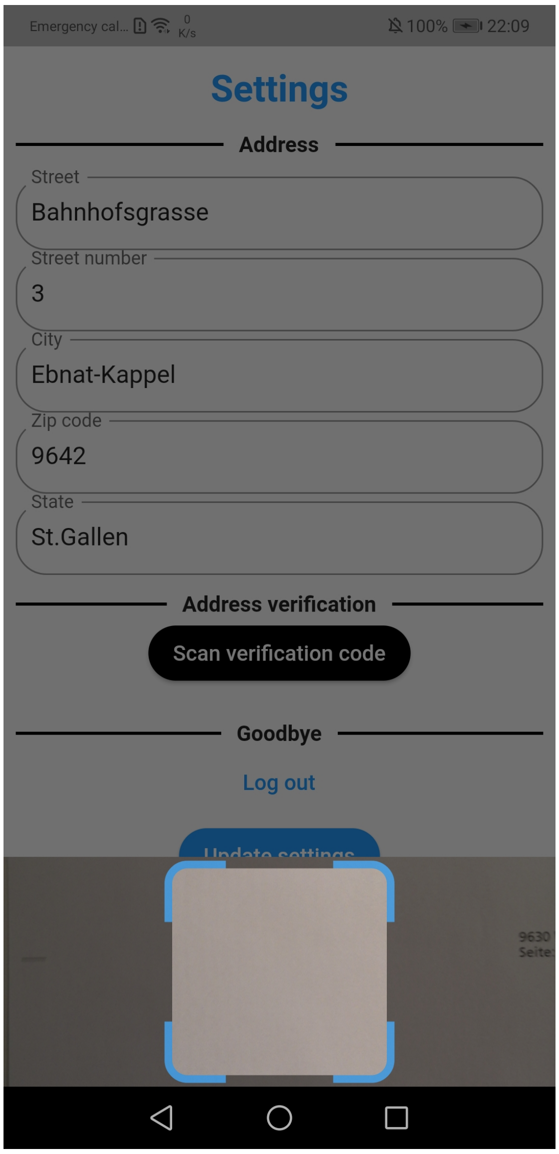


Figure 10.18: Mobile application QR code scanning

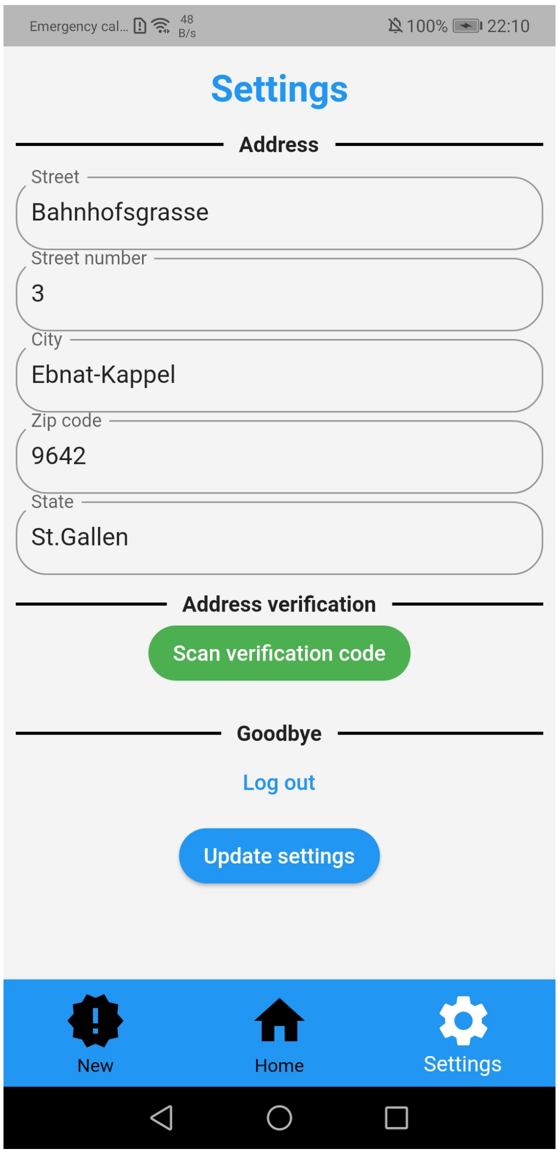


Figure 10.19: Mobile application settings page

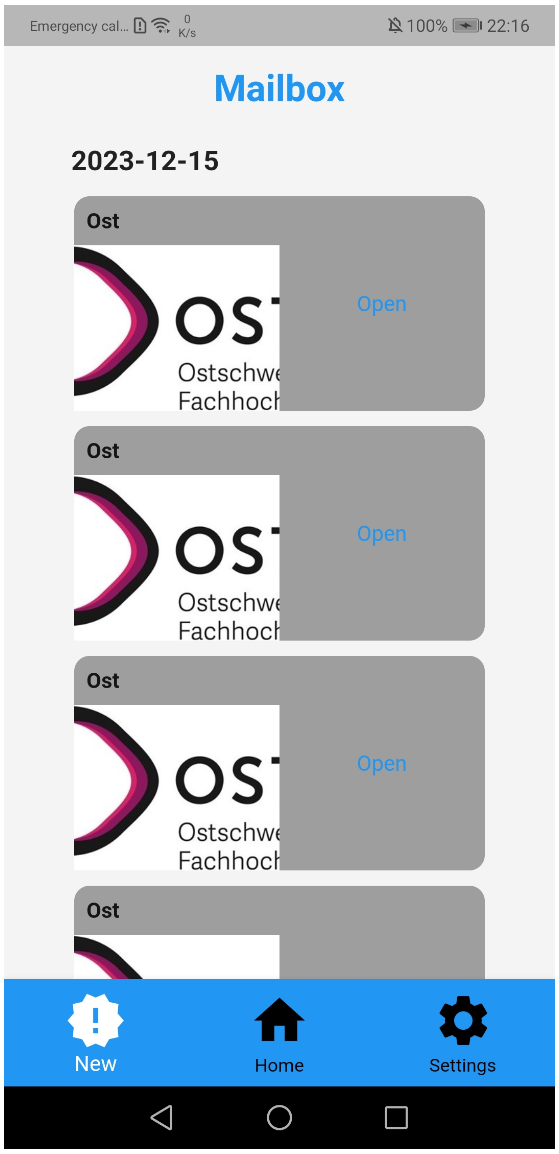


Figure 10.20: Mobile application unread letters

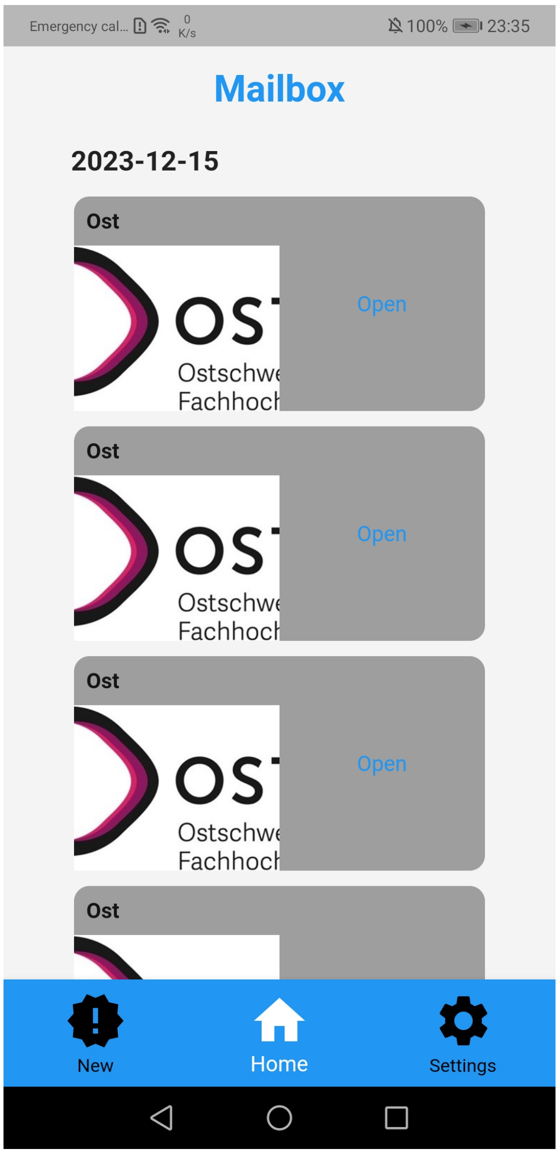


Figure 10.21: Mobile application all letters

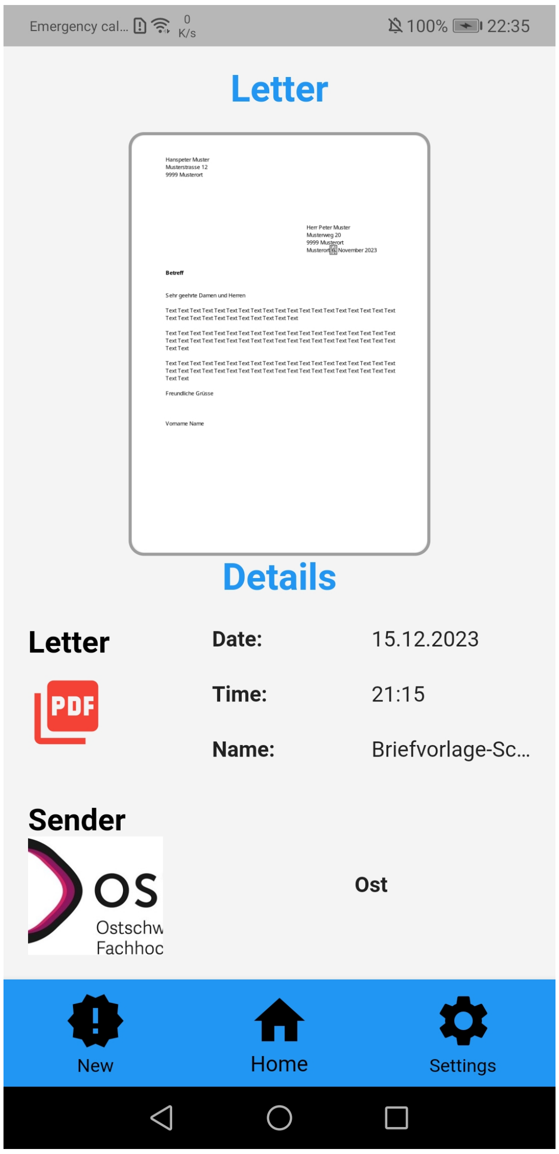
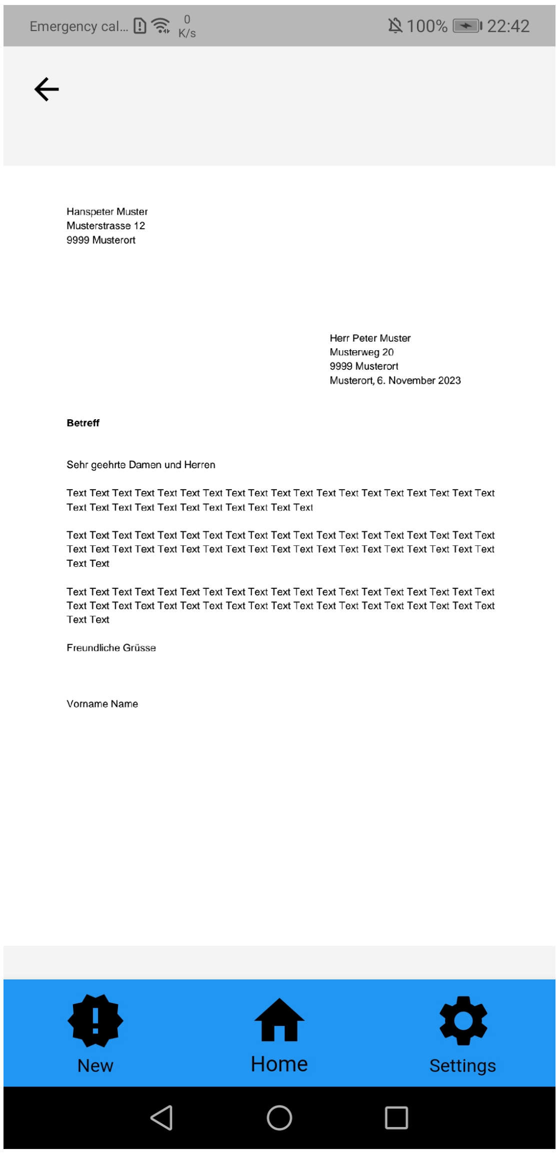


Figure 10.22: Mobile application letter details



Hanspeter Muster
Musterstrasse 12
9999 Musterort

Herr Peter Muster
Musterweg 20
9999 Musterort
Musterort, 6. November 2023

Betreff

Sehr geehrte Damen und Herren

Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text

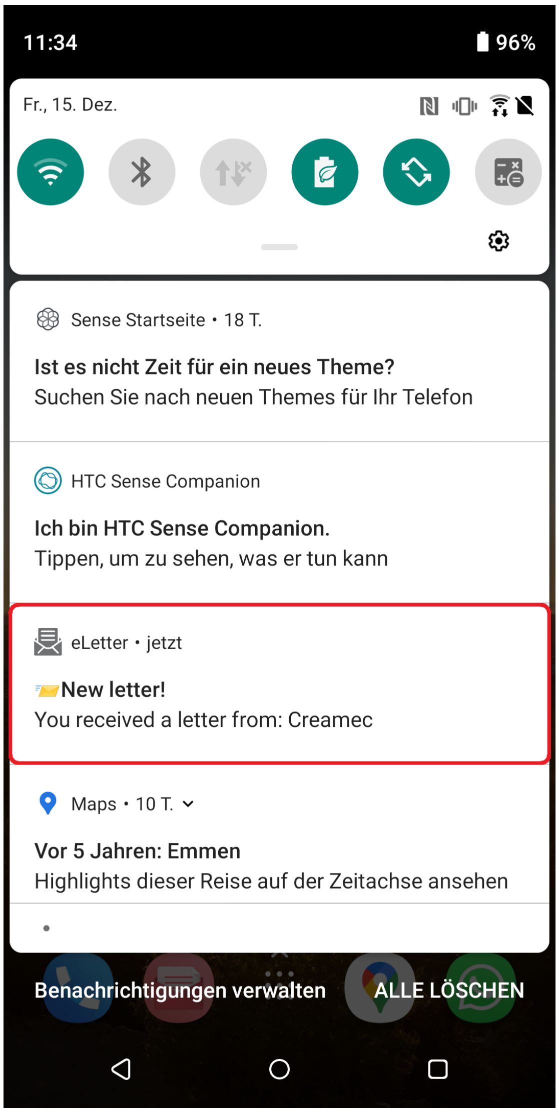
Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text

Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text

Freundliche Grüsse

Vorname Name

Figure 10.23: Mobile application pdf view



11:34 96%

Fr., 15. Dez.

Sense Startseite • 18 T.

Ist es nicht Zeit für ein neues Theme?
Suchen Sie nach neuen Themes für Ihr Telefon

HTC Sense Companion

Ich bin HTC Sense Companion.
Tippen, um zu sehen, was er tun kann

eLetter • jetzt

New letter!
You received a letter from: Creamec

Maps • 10 T.

Vor 5 Jahren: Emmen
Highlights dieser Reise auf der Zeitachse ansehen

Figure 10.24: Mobile application push-notification

10.3 Task definition

Aufgabenstellung Semesterarbeit "Digitaler Briefversand"

1. Beteiligte Personen

- Studierende: Marc Kissling, Andrew Willi, Gian-Luca Vogel
- Industriepartner: AdaptIT GmbH, Michael Güntensperger
- Betreuer: Frank Koch

2. Problembeschrieb

Wichtige Unterlagen werden meist per Mail oder Brief versendet. Die Brief-Variante kostet Zeit und Geld. Beim Mailversand hingegen ist von Nachteil, dass wichtige Informationen leicht in der Masse untergehen. Mit der zu entwickelnden Lösung soll sowohl das Senden wie auch Empfangen vereinfacht werden. Ver- sendet wird durch den Upload eines PDF's im Brief-Format in einer Desktop-Applikation. Im Anschluss wird das Dokument sofort an die zu entwickelnde App des Empfängers ausgeliefert. Falls der Empfänger die App noch nicht im Einsatz hat, wird das Dokument physisch versendet und Informationen zum App-Download sowie zur Account-Erstellung beigelegt. Empfänger wie auch Versender werden an- hand ihrer Wohnadresse verifiziert.

3. Aufgabenstellung

In dieser Arbeit soll nur ein Teil der umfangreichen Applikation entwickelt werden. Dafür angedacht sind:

- Desktop-App für den Versand der digitalen Briefe (bzgl Empfang siehe optionale Anforderungen)
- Einfache Mobile-App zum Empfangen von digitalen Briefen
- Backend für Verwaltung von Accounts und Dokumenten
- Registration als neuer Empfänger, inklusive Validierung der Adresse
- Möglichkeit zum Upload von Briefen (z.B. nach Amazon S3)

Gerne werden die Interessen und Ideen der Studierenden berücksichtigt.

Technische Umgebung

Für die Umsetzung wird mit Web-Technologien gearbeitet.

- Frontend-Desktop: React Native / Electron
- Frontend-App: Flutter / React Native / ionic
- Backend: Node.js
- Datenbank: MySQL / PostgreSQL
- Object Storage: AWS S3 / Spaces on Digital Ocean
- Hoster: DigitalOcean (und Hostpoint)

Funktionale Anforderungen

- Eingeschränkter Zugriff auf eigene Dokumente (Mobile-App)
- Registration als neuer Empfänger
- Adressverifikation mittels z.B. QR-Code per Post
- Registration als Firma / Versender
- Upload von Dokumenten für Versand
- Deployment in die Cloud

Optionale Anforderungen

- Automatische Adresserkennung im PDF
- Kategorisierung der Dokumente in Mobile-App
- Versandart in Desktop-App auswählen
- Kostenanzeige beim Versand via Desktop-App
- Dashboard zur Verwaltung von Mitarbeitenden für Briefversand inkl. Berechtigungen
- Firmenbranding für Desktop-App
- Webapplikation für Empfänger

Nicht-Funktionale Anforderungen

- Das Entwicklerteam implementiert die Features gemäss der abgesprochenen Priorität mit dem Kunden
- Das Backend sollte 1000 Requests pro Minute handeln können
- Jede Seite sollte nicht länger als 200ms für das Laden benötigen
- Die Desktop-Applikation soll auf Windows laufen
- Die Applikation soll einfach installiert / ausgeführt werden können
- Drei von vier Test-user sollten das UI (Kategorien: layout, responsiveness, colour, content) der Applikation mit einem handy mit einer Note von mindestens 8 von 10 bewerten, wobei 10 das beste ist.
- Die Datenbank soll bis zu 10’000 Dokumente / PDF’s managen können.
- Errors sollen keine Systemfehler erzeugen, aber eine Error Nachricht Zeigen und das System auf den vorherigen Zustand zurücksetzen.
- Jeder Error soll im System geloggt werden
- Jede Kommunikation zwischen Fron- und Backend soll mit einem SSL-Zertifikat verschlüsselt werden.
- Daten welche in Eingabefelder abgefüllt werden, sollen zuerst validiert werden, bevor diese durch das System verarbeitet werden. SQL Injection test der Eingabefelder sollte keine Verletzlichkeiten zeigen.

- User-Passwörter werden nicht in plain-text in der Datenbank gespeichert.
- Businesslogik im Backend soll modular aufgebaut werden, so dass sie erweitert werden kann.
- Die Backend-API soll durch API-testing Tools getestet werden.
- Datenbank, Backend und Frontend sollen auf unterschiedlichen Instanzen deployed werden.

4. Zur Durchführung

Mit dem Betreuer finden Besprechungen gemäss Absprache statt. Die Besprechungen sind von den Stu- dierenden mit einer Traktandenliste vorzubereiten und die Ergebnisse in einem Protokoll zu dokumentie- ren, das dem Betreuer per E-Mail zugestellt wird. Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen gemäss Projektplan sind einzelne Arbeitsresultate in vorläufigen Versionen abzugeben.

5. Dokumentation und Abgabe

Siehe Leitfaden Abschnitt 5.5 "Umfang und Form der Abgabe".

6. Termine

Siehe veröffentlichte «Termine SA HS23».

7. Bewertung

Siehe veröffentlichter «Leitfaden BA SA» Abschnitt 6 "Bewertung", insbesondere 6.4.
Rapperswil, den 28.08.23

Frank Koch

10.4 Testing Desktop Application

10.4.1 UX Testing

Flow:

1. **Welcome Page:**
   Please open and view the Welcome Page.
2. **Sign up:**
   Create a user account by signing up. Follow the registration process.
3. **Login:**
   Log in using the credentials you just created.
4. **Upload:**
   Now that you are logged in, upload a file.
5. **Address input:**
   Now put in the address.
6. **Send:**
   Now send the letter.
7. **Settings:**
   Now locate where you can change the picture, but do not change it.
8. **Finish:**
   Congratulations you did it! Thank you for participating.

Test Person 1

User Description:
The test participant is a 24-year-old female with a strong affinity for design and a normal affinity for technology. She uses mainly Apple products.

She is an intersting candidate for evaluating an application's usability and appeal to a broader demographic. Because she is not very tech-savvy, she can provide valuable insights into how well an application caters to users of younger generation.

Negative Feedback:

- The user expressed that the color scheme of the application is not very appealing and the design is outdated. It does not have the flair, which the modern applications have.
- In her opinion the icon for uploading and sending document(s) is not clear. She suggested to use a house icon. And overall that the icons are a little bit big.
- She was very surprised how fast the whole process was.

Positive Feedback:

- On the other side she said it is very easy to do the tasks.
- The upload process is very fast.

Conclusion:
In summary, the design is not very appealing but the process is very easy.

10.4.2 Usablity Test

Only the scenario and tasks will be shown to the test users. We will ask all other questions.
Background Questions:

- Can you briefly describe your experience with similar applications or online services?
- Have you used applications with similar features before?

Scenario:
Imagine you own a small start-up business that frequently needs to send and receive important documents such as contracts and legal papers. Currently, you rely on traditional methods such as mail and email, each of which has its own set of challenges. You are now introduced to a new solution that aims to streamline both the sending and receiving of important documents.

Tasks:

1. **Sign up:**
   Create a user account by signing up. Follow the registration process.
2. **Login:**
   Log in using the credentials you just created.
3. **Settings:**
   Verify your account.
4. **Upload:**
   Now that you are logged in, upload a file.
5. **Address input:**
   Now put in the address of the receiver, which is visible on the uploaded file.
6. **Send:**
   Now send the letter.
7. **Settings:**
   Change your logo.

Closing questions (in order of importance):

1. On a scale of 1 to 10, how satisfied are you with the overall layout of the application?
2. What did you think of the layout and design of the different pages/forms?
3. Did you find the application responsive to your actions?
4. Thoughts on the colour scheme used throughout the application?
5. Were the instructions and content presented in a clear and understandable manner?

Result of Usability test:

Participant 1:

1. **Test date:** 01.12.2023
2. **Background:** 24-year-old female student with some experience of similar applications. She has already done our UX testing.

Ratings:

1. **General satisfaction rating (scale 1-10):** 7
2. **Layout and design:** The design of the different pages and forms is good. She liked the navigation bar at the top, which is different from the mock-up.
3. **Responsiveness rating:** She did not give specific feedback on responsiveness. But she did try different email formats and postcode formats, which gave her the helpful error messages.
4. **Colour scheme thoughts:** The overall colour scheme was good, but she thought the red colour of the unsubscribe button was too aggressive.
5. **Clarity of instructions and content:** In the login task, she did not fully understand the purpose of the upload functionality because there was no description of it. After seeing the test file she realised what it was.

Conclusion: It has not many features, but the upload functionality should be better.

Participant 2:

1. **Test date:** 12.12.2023
2. **Background:** 23 year old male student with experience of similar applications.

Ratings:

1. **General satisfaction rating (scale 1-10):** 8
2. **Layout and design:** The design is very simple, but nothing more. For what it is, it is a solid application.
3. **Responsiveness Rating:** Tried different streets and postcodes. He was happy with the error messages.
4. **Colour scheme thoughts:** It is good.
5. **Clarity of instructions and content:** In the registration task, he completely ignored the purpose of the upload functionality. After the login button did not work, he asked what was wrong.

Conclusion: Simple and clear application, it does what it should.

10.5 Testing Mobile Application

10.5.1 UX-Testing

Flow:

1. **Welcome Page:**
   Please open and view the Welcome Page.
2. **Sign-up:**
   Create a user account by signing up. Follow the registration process.
3. **Login:**
   Log in using the login credentials you just created.
4. **Home:**
   Navigate to the Home screen of the app.
5. **Open a Document (PDF from Ostschweizer Fachhochschule):**
   Search for and open the specified document.
6. **Find Document Name and Sending Time:**
   Locate the name and sending time of the opened document from the detail view.
7. **View the Document Itself:**
   Open and view the document.
8. **Return to Detail View:**
   Navigate back from the document view to the detail view.
9. **Show All New Documents (via bottom menu navigation):** Use the bottom menu navigation to find and display all new documents.
10. **Increase the House Number in Settings:**
    Go to the app's settings and increase the house number by one.

Test Person 1

User Description:
The test participant is a 60-year-old male with a strong affinity for technology, particularly in the field of information technology (IT). He brings with him several decades of experience and a keen interest in exploring and evaluating new digital solutions. His age and background make him an interesting candidate for assessing the usability and accessibility of software, as he represents an older demographic with a deep understanding of technology.

Despite his age, this user maintains an active engagement with modern digital tools and applications. He has a wealth of experience in using various software applications and can offer valuable insights into how well an application caters to users of his generation.+
Negative Feedback:

- The user expressed dissatisfaction with the app's color scheme, describing it as monotonous and lacking variety.
- He found the app's design somewhat outdated, suggesting that it could benefit from a more contemporary look.
- During his interaction with the app, the user mistakenly confused the "Scan Code" button in the settings with the "Update Settings" button, highlighting potential issues with button labels and clarity.
- The user also noted that, in his opinion, the application appeared to have a relatively limited amount of content for a standalone application.

Positive Feedback:

- On a positive note, the user commended the app's ability to provide a clear and comprehensive overview of its features, making it easy to navigate and understand.
- He was able to complete all tasks assigned during the testing session efficiently and without encountering significant obstacles.
- The user appreciated the app's clean and uncluttered interface, noting that it wasn't overloaded with information.
- The inclusion of large buttons in the application was a particularly welcome feature for the user, as it made the interface more accessible, even for individuals with less precise motor skills.

Conclusion:
In summary, this test participant's IT background, combined with his age, makes him a valuable resource for evaluating an application's usability and appeal to a broader demographic. His feedback, both positive and negative, offers valuable insights for improving the app's design and functionality.

10.5.2 Usablity Test

To evaluate the user experience and functionality of the mobile application through a series of tasks that encompass starting the app, registration, login, address validation, and information retrieval.

Test Environment:
Device: Test-Device (Android)
Application Version: preinstalled version on the device
Internet Connection: Wi-Fi/3G/4G/5G

Participant Profile:
Age: 18-80
Tech-Savviness: Moderate to High

Flow:

1. **Start the App**
   Objective: Assess the app's loading time and initial responsiveness.
   Steps: Locate the application icon and open the app.
   Expected Outcome: The application should load within [2 seconds] and display the welcome screen.

2. **Navigate to the Registration**
   Objective: Evaluate the intuitiveness of the navigation to the registration page.
   Steps: From the home screen, locate and select the option to register.
   Expected Outcome: The registration page should be easily accessible from the home screen.

3. **Register on the Platform**
   Objective: Test the registration process for ease and efficiency.
   Steps: Complete the registration form with the required details and submit.
   Expected Outcome: Registration should be successful without errors, and the user should be directed to a confirmation screen.

4. **Log Into the App**
   Objective: Assess the login process and its functionality.
   Steps: Log in using the newly created credentials.
   Expected Outcome: The login should be smooth, and the user should be directed to the main dashboard.

5. **Increment House Number in Settings**
   Objective: Test the functionality of incrementing the house number by one in the settings.
   Steps: Go to the address modification section in the Settings. Select the current house number, increase it by one and update the settings.
   Expected Outcome: The application should successfully increment the house number by one and display the updated address in the settings.

6. **Validate the Address with the QR-Code**
   Objective: Test the QR code scanning feature for address validation.
   Steps: Navigate to the address validation section in the Settings and use the application to scan a provided QR code.
   Expected Outcome: The application should successfully scan the QR code and validate the address.

7. **Log Out**
   Objective: Ensure the log-out process is user-friendly and secure.
   Steps: Navigate to the settings or account menu and select the option to log out.
   Expected Outcome: The application should successfully log the user out, redirecting them to the login or welcome screen, ensuring that no personal data is accessible until the next login.

8. **Log Into the application again with prepared credentials**
   *Objective:*Evaluate the login procedure for an existing account.
   Steps: Log in using the prepared credentials. *Credentials:* E-Mail: anna.zuellig@gmail.com, Password: Anna01!
   Expected Outcome: The login should be smooth, and the user should be directed to the main dashboard with the prepared letters.

9. **Write Down Who is the Sender of the Letter**
   Objective: Evaluate the app's ability to retrieve specific historical data.
   Steps: Locate and open the correspondence history, find the letter dated 07.12.2023, and identify the sender.
   Expected Outcome: The application should display the history accurately, and the user should easily find the sender's name.

10. **Write Down at what Time the Letter was Sent**
    Objective: Test the app's functionality in providing detailed information about past activities.
    Steps: In the correspondence history, locate the letter dated 08.12.2023 and note the time it was sent.
    Expected Outcome: The time the letter was sent should be clearly displayed and accurate.

11. **Write Down the Last Sentence from the PDF**
    Objective: Assess the app's ability to handle document viewing and information extraction.
    Steps: Find the PDF sent by Ost-Rapperswil, open it, and write down the last sentence.
    Expected Outcome: The PDF should open seamlessly within the app, and the user should be able to easily read and extract the required information.

12. **Open Push Notification**
    Objective: Evaluate the effectiveness and responsiveness of push notifications.
    Steps: Wait for a push notification to arrive and open it by tapping on the notification. *Expected Outcome:* The application should respond immediately to the App.

Test Person 1

User Description:
The test participant is a 60-year-old male with a strong affinity for technology, particularly in the field of information technology (IT). He brings with him several decades of experience and a keen interest in exploring and evaluating new digital solutions. His age and background make him an interesting candidate for assessing the usability and accessibility of software, as he represents an older demographic with a deep understanding of technology.

Despite his age, this user maintains an active engagement with modern digital tools and applications. He has a wealth of experience in using various software applications and can offer valuable insights into how well an application caters to users of his generation.

It was the same test person who had already tested the UX prototype.

1. **Overall Experience**
   The overall experience includes thematic areas such as meta content, images and relevance, which are all sub-themes of the content. But the overall experience also includes balance and alignment, contrast and proportionality as well as scaling in relation to the layout.
   1[ ] 2[ ] 3[ ] 4[ ] 5[ ] 6[ ] 7[ ] 8[ ] 9[**X**] 10[ ] (1 - Very Poor, 10 - Excellent)

2. **Usability**
   This includes balance and alignment, contrast and proportionality as well as scaling in relation to the layout. Usability also includes aspects such as touchscreen friendliness, fast loading time and the interactivity of the elements in terms of responsiveness.
   1[ ] 2[ ] 3[ ] 4[ ] 5[ ] 6[ ] 7[ ] 8[**X**] 9[ ] 10[ ] (1 - Very Poor, 10 - Excellent)

3. **Dynamic-Content**
   Responsiveness also includes aspects such as fast loading times, touchscreen friendliness and interactivity. Criteria such as meta content, images and relevance, which fall under the content category, are also part of content which, like responsiveness, is a sub-category of dynamic content.
   1[ ] 2[ ] 3[ ] 4[ ] 5[ ] 6[ ] 7[ ] 8[ ] 9[**X**] 10[ ] (1 - Very Poor, 10 - Excellent)

4. **Look and feel**
   The look and feel should evaluate the color harmony as well as the contrast and the interplay of background and texture.
   1[ ] 2[ ] 3[ ] 4[ ] 5[ ] 6[ ] 7[**X**] 8[ ] 9[ ] 10[ ] (1 - Very Poor, 10 - Excellent)

5. **Features and Functionality**
   Are there any features or functionalities that you particularly liked or found useful?
   - *Good overview page before opening the letters*
   - *Intuitive display after the address has been verified (button highlighted in green).*

   Are there any features or functionalities that you think need improvement?
   - *Web solution for customers with advanced age would still be desirable due to the screen sizes of mobile devices.*
   - *Print function would also be desirable.*

6. **Problems or Issues**
   Did you encounter any problems or issues while using our product/service?

   The keyboard covered one of the fields when registering the address.

7. **Recommendations**
   Based on your experience, what improvements or additional features would you recommend for our product/service?

   It would be more intuitive for me personally if the buttons for login and registration would swap places.

8. **Additional Comments**
   Please provide any other comments or suggestions you may have.

   With regard to the benefit for the user of the app, I doubt whether it offers the company or the user any significant benefit.

Negative Feedback:

- *swap login and registration buttons*

Positive Feedback:

- *Tidy and clean look*
- *Fast response time within the application*

Conclusion:
On the whole, the application was intuitive to use and you quickly recognized where, what and how you could reach. It would be nice if there was a web version, but this has nothing to do with the application directly. I wish the team continued success in their studies and fun at work in the coming years.

Test Person 2

User Description:
The third participant is a 23-year-old woman living in a medium-sized city, sharing her living space in a shared housing. She has a pronounced passion for travel, often exploring new destinations. Her urban living situation and communal lifestyle provide her with diverse interpersonal interactions and experiences, enriching her understanding of various cultures and lifestyles. This background, combined with her love for travel, gives her a broad perspective on life, potentially influencing her approach to personal and professional challenges.

1. **Overall Experience**
   The overall experience includes thematic areas such as meta content, images and relevance, which are all sub-themes of the content. But the overall experience also includes balance and alignment, contrast and proportionality as well as scaling in relation to the layout.
   1[ ] 2[ ] 3[ ] 4[ ] 5[ ] 6[ ] 7[ ] 8[ ] 9[**X**] 10[ ] (1 - Very Poor, 10 - Excellent)

2. **Usability**
   This includes balance and alignment, contrast and proportionality as well as scaling in relation to the layout. Usability also includes aspects such as touchscreen friendliness, fast loading time and the interactivity of the elements in terms of responsiveness.
   1[ ] 2[ ] 3[ ] 4[ ] 5[ ] 6[ ] 7[ ] 8[ ] 9[ ] 10[**X**] (1 - Very Poor, 10 - Excellent)

3. **Dynamic-Content**
   Responsiveness also includes aspects such as fast loading times, touchscreen friendliness and interactivity. Criteria such as meta content, images and relevance, which fall under the content category, are also part of content which, like responsiveness, is a sub-category of dynamic content.
   1[ ] 2[ ] 3[ ] 4[ ] 5[ ] 6[ ] 7[ ] 8[ ] 9[ ] 10[**X**] (1 - Very Poor, 10 - Excellent)

4. **Look and feel**
   The look and feel should evaluate the color harmony as well as the contrast and the interplay of background and texture.
   1[ ] 2[ ] 3[ ] 4[ ] 5[ ] 6[ ] 7[ ] 8[ ] 9[**X**] 10[ ] (1 - Very Poor, 10 - Excellent)

5. **Features and Functionality**
   Are there any features or functionalities that you particularly liked or found useful?
   - *I thought it was good that the new documents were displayed separately. This way you always have an overview of which documents you have already viewed and which you haven't yet.*

   Are there any features or functionalities that you think need improvement?
   - *Signup*
   - *Login*

6. **Problems or Issues**
   Did you encounter any problems or issues while using our product/service?

   No significant

7. **Recommendations**
   Based on your experience, what improvements or additional features would you recommend for our product/service?

   It might be a good idea to be able to share the documents with other people who also have this app.

8. **Additional Comments**
   Please provide any other comments or suggestions you may have.

   I like the idea. But I have not found a way to group the documents. This would generally make it easier for me to add my own kind of sorting.

Negative Feedback:

- *No Navigate, feedback.*

Positive Feedback:

- *Realy fast App.*

Conclusion:
It was fun to use the application and I can well imagine that this will become the new way to manage documents. I can also imagine using it in the business sector, but there would have to be a few more functions added. I hope that the idea will catch on in the future and that I won't have to scan in all my documents like I have to do so far.

Test Person 3

User Description:
The second individual is a 27-year-old man working in sales, known for his dynamic lifestyle. He resides in a mountainous region, embracing solitude during the weekdays. His home environment reflects a blend of tranquility and ruggedness, shaping his personal and professional demeanor. Despite his isolated living situation, he maintains an active social life, traveling every weekend to visit his girlfriend in a different city. This routine highlights his adaptability and dedication to maintaining personal relationships, qualities that likely translate into his professional life as a salesperson.

1. **Overall Experience**
   The overall experience includes thematic areas such as meta content, images and relevance, which are all sub-themes of the content. But the overall experience also includes balance and alignment, contrast and proportionality as well as scaling in relation to the layout.
   1[ ] 2[ ] 3[ ] 4[ ] 5[ ] 6[ ] 7[ ] 8[ ] 9[**X**] 10[ ]   (1 - Very Poor, 10 - Excellent)

2. **Usability**
   This includes balance and alignment, contrast and proportionality as well as scaling in relation to the layout. Usability also includes aspects such as touchscreen friendliness, fast loading time and the interactivity of the elements in terms of responsiveness.
   1[ ] 2[ ] 3[ ] 4[ ] 5[ ] 6[ ] 7[ ] 8[ ] 9[**X**] 10[ ]   (1 - Very Poor, 10 - Excellent)

3. **Dynamic-Content**
   Responsiveness also includes aspects such as fast loading times, touchscreen friendliness and interactivity. Criteria such as meta content, images and relevance, which fall under the content category, are also part of content which, like responsiveness, is a sub-category of dynamic content.
   1[ ] 2[ ] 3[ ] 4[ ] 5[ ] 6[ ] 7[ ] 8[ ] 9[ ] 10[**X**]   (1 - Very Poor, 10 - Excellent)

4. **Look and feel**
   The look and feel should evaluate the color harmony as well as the contrast and the interplay of background and texture.
   1[ ] 2[ ] 3[ ] 4[ ] 5[ ] 6[ ] 7[ ] 8[**X**] 9[ ] 10[ ]   (1 - Very Poor, 10 - Excellent)

5. **Features and Functionality**
   Are there any features or functionalities that you particularly liked or found useful?
   - *Opening the PDF without having to leave the app.*
   - *The display of when and at what time the document was delivered by whom was good and clearly presented.*

   Are there any features or functionalities that you think need improvement?
   - *The preview of the PDF looked a bit strange during one test*

6. **Problems or Issues**
   Did you encounter any problems or issues while using our product/service?

   No, i faced no problems

7. **Recommendations**
   Based on your experience, what improvements or additional features would you recommend for our product/service?

   Swipe functionality to change the view

8. **Additional Comments**
   Please provide any other comments or suggestions you may have.

   - - -

Negative Feedback:

- *unattractive icon*

Positive Feedback:

- *fast reaction time*
- *good overview of the letters*
- *nice scroll-functionality*

Conclusion:
Finally an application in the administration environment that has a very fast response time.

Test Person 4

User Description:
The test participant is a 53-year-old woman employed in the marketing sector, residing in a rural area. As a mother of two adult children, she brings a unique perspective to her work, balancing professional responsibilities with the insights gained from raising a family. Her rural living environment and experience as a parent significantly influence her approach to marketing, especially in understanding and connecting with diverse consumer demographics. Her insights are particularly valuable in campaigns targeting family-oriented audiences or those living outside urban centers.

1. **Overall Experience**
   The overall experience includes thematic areas such as meta content, images and relevance, which are all sub-themes of the content. But the overall experience also includes balance and alignment, contrast and proportionality as well as scaling in relation to the layout.
   1[ ] 2[ ] 3[ ] 4[ ] 5[ ] 6[ ] 7[ ] 8[ ] 9[**X**] 10[ ] (1 - Very Poor, 10 - Excellent)

2. **Usability**
   This includes balance and alignment, contrast and proportionality as well as scaling in relation to the layout. Usability also includes aspects such as touchscreen friendliness, fast loading time and the interactivity of the elements in terms of responsiveness.
   1[ ] 2[ ] 3[ ] 4[ ] 5[ ] 6[ ] 7[ ] 8[**X**] 9[ ] 10[ ] (1 - Very Poor, 10 - Excellent)

3. **Dynamic-Content**
   Responsiveness also includes aspects such as fast loading times, touchscreen friendliness and interactivity. Criteria such as meta content, images and relevance, which fall under the content category, are also part of content which, like responsiveness, is a sub-category of dynamic content.
   1[ ] 2[ ] 3[ ] 4[ ] 5[ ] 6[ ] 7[ ] 8[ ] 9[**X**] 10[ ] (1 - Very Poor, 10 - Excellent)

4. **Look and feel**
   The look and feel should evaluate the color harmony as well as the contrast and the interplay of background and texture.
   1[ ] 2[ ] 3[ ] 4[ ] 5[ ] 6[ ] 7[ ] 8[**X**] 9[ ] 10[ ] (1 - Very Poor, 10 - Excellent)

5. **Features and Functionality**
   Are there any features or functionalities that you particularly liked or found useful?
   - *All of them*

   Are there any features or functionalities that you think need improvement?
   - *The registration page*

6. **Problems or Issues**
   Did you encounter any problems or issues while using our product/service?

   When registering, I could not see some fields while I was filling them in. And the button is a bit narrow at the bottom of the screen.

7. **Recommendations**
   Based on your experience, what improvements or additional features would you recommend for our product/service?

   - *A download function would be good*
   - *An archiving function that saves the documents on the PC*

8. **Additional Comments**
   Please provide any other comments or suggestions you may have.

   It's a nice application and it works well, but I don't quite understand what the added value is compared to e-mail. But the application contains everything I need for the use case.

Negative Feedback:

- *missing download function*

Positive Feedback:

- *good overview*
- *nice idea*
- *good implementation fo filter new documents*

Conclusion:
Finally an application in the administration environment that has a very fast response time.

User-Test Evaluation

10.6 Testing Backend

10.6.1 Test code snippet

The following code snippet demonstrates the implementation of an integration test. The test is divided into three parts. First, it initializes the application with mocked services in the beforeAll statement. Then, the necessary data for the test is created in the database. After that, the actual test is executed. Finally, after all tests in the file are executed, the database is cleared in the afterAll statement.

```
describe('Letter sending tests (E2E)', () => {
 let app: INestApplication;
 let prisma: PrismaClient;
 let user: User;
 let organisation: Organisation;

 beforeAll(async () => {
  const moduleFixture: TestingModule = await Test.createTestingModule({
    imports: [AppModule],
  })
    .overrideProvider(PostalService)
    .useValue(postalService)
    .compile();

  app = moduleFixture.createNestApplication();
  await app.init();
  prisma = new PrismaClient();
  user = await addUserToDb(prisma, userData);
  organisation = await addOrganisationToDb(prisma, organisationData);
  await prisma.$disconnect();
 });

 afterAll(async () => {
  await clearDb(prisma);
  await app.close()
 });

 it('It should not be able to send letter because organisation is not
    verified', async() => {
  ...
 });
  ...
});
```

Listing 10.11: Test code sniped