Digitalizing Time Registration For Construction Workers

Development of a Mobile Application

Bachelor's Thesis
Degree Programme in Computer Applications
Spring 2025
Gitte Vandevenne

HAMK
Häme University
of Applied Sciences

Abstract

DP	Degree Programme in Computer Applications
Author	Gitte Vandevenne
Subject	Digitalizing Time Registration For Construction Workers
Supervisors	Tommi Lahti

Year 2025

The purpose of this thesis was to develop a mobile application for digitalizing time registration for construction workers, addressing the inefficiencies and inaccuracies associated with manual timekeeping systems.

The thesis sought to analyse the design principles and usability standards necessary for creating an intuitive user interface, as well as to explore the integration of Flutter and PostgreSQL for a scalable time registration system.

The research questions focused on the application of UI/UX design principles, the technical integration of mobile app development with a database, and the regulatory compliance challenges, particularly concerning GDPR and Checkinatwork requirements.

The methodology employed was practical, utilizing the Scrum framework for iterative development, which included designing, implementing, and testing the application.

The findings indicate that the application successfully facilitates time registration while ensuring compliance with relevant regulations. The analysis suggests that future enhancements could include payroll integration and further accessibility improvements.

It is recommended that the company consider user feedback for ongoing development and ensure adherence to accessibility standards to enhance usability for all users.

Keywords	Flutter – Mobile application – Checkinatwork
Pages	46 pages and appendices 1 page

Glossary

ERP	Enterprise Resource Planning
UI	User Interface
JIT	Just-In-Time
AOT	Ahead-Of-Time
SDK	Software Development Kit
GPU	Graphics Processing Unit
fps	frames per second
UX	User Experience
ISO	International Organization for Standardization
WCAG	Web Content Accessibility Guidelines
MVVM	Model-View-ViewModel
DBMS	Database Management System
ACID	Atomicity, Consistency, Isolation, Durability
BASE	Basically Available, Soft state, Eventual Consistency
MVCC	Multi Version Concurrency Control
QR	Quick Response
GDPR	General data protection Rules
EU	European Union

Table of Contents

Figures

Tables

Appendices

1 Introduction

Many construction companies rely on manual systems for administrative tasks such as timekeeping. This manual approach presents challenges in terms of accuracy and efficiency and complicates regulatory compliance.

In Belgium, anyone who performs work in immovable property at a workplace whose total cost equals or exceeds €500 000 (excluding VAT) must register with Checkinatwork. Through Checkinatwork, employers and contractors register their employees' and subcontractors' presence. The employees or independent subcontractors can also register themselves in the system (Checkinatwork, n.d.).

This thesis develops a mobile application designed to digitalise the timekeeping process for workers in the construction industry. This application will facilitate Checkinatwork and payroll administration and improve operational efficiency. The practical part of the thesis consists of designing, developing and implementing this application using Flutter.

This thesis is being conducted for a company called Deuse. It is a software engineering company specialising in developing customised digital tools for other companies. This particular application is for a dummy client in the construction industry.

The scope of the thesis includes building a robust and user-friendly application that integrates seamlessly with the client’s existing ERP (Enterprise resource planning) system. While the primary focus will be on creating the time registration functionality, future enhancements for payroll integration are acknowledged but fall outside the immediate scope.

The research questions of this thesis are:

1. What design principles and usability standards can be applied to create an intuitive and user-friendly interface for time registration in construction projects?
2. How can Flutter and PostgreSQL be integrated to develop a scalable and efficient time registration system for construction workers?
3. What are the security and regulatory compliance challenges in implementing a digital time registration system for construction companies, and how can they be addressed?

2 Theoretical Framework

The theoretical framework focuses on the knowledge required to develop the application. This chapter covers four main topics: mobile app development, database design, QR codes, and regulatory compliance. Each of these is important for developing this thesis. The following sections delve deeper into the theory behind these areas.

2.1 Mobile App Development

Several important parts of mobile app development will be explained in this subsection. The first aspect explained is the programming language Dart. The second one is the development framework, Flutter, and its advantages and disadvantages. The next part discussed is UI/UX design principles. The last aspect explained in this subsection is the MVVM design pattern.

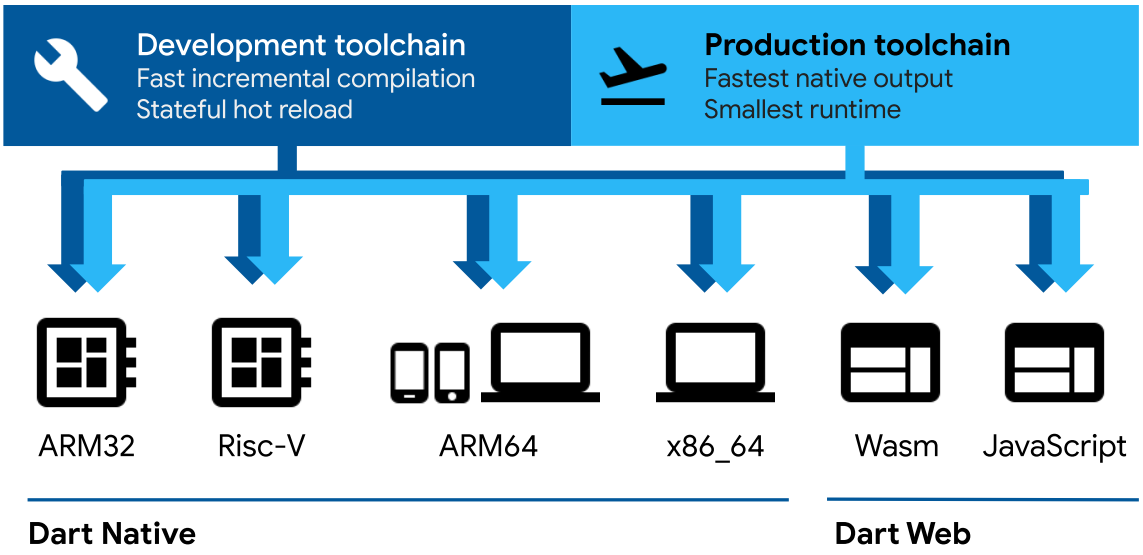
2.1.1 Dart

Dart (Dart Programming Language) is an open-source programming language introduced by Google in 2011. Designed with object-oriented principles, it is particularly well-suited for developing UIs. It supports dynamic typing, mixins for multiple inheritance, null safety to prevent runtime errors, and automatic memory management via garbage collection. Its ability to compile ahead of time into native code ensures high performance. Dart is primarily used for crafting user interfaces, eliminating the necessity of using an additional language for UI development (Dart Overview, n.d.).

Dart is designed for multi-platform development and optimised for building fast apps across web, mobile and desktop platforms. It includes a rich set of core libraries for web and mobile app development (Dart Programming Language, n.d.).

Dart’s compiler technology makes it possible to run code in different ways. This is shown in Figure 1. The code can be compiled to Dart native or Dart for Web. Dart native is for mobile and desktop apps. It provides just-in-time (JIT) compilation for hot-reloading during development and ahead-of-time (AOT) compilation for optimised native performance at runtime. Dart for Web compiles to JavaScript using dartdevc and dart2js. dartdevc is used for fast development with incremental compilation, and dart2js is used for optimised deployable JavaScript.

Figure 1. Dart platform compilation (Dart Overview, n.d.)



In software development, it is common to wait for long tasks to complete before proceeding. To allow other parts of the program to run while one task is still in progress, Dart offers support for asynchronous programming. This is managed through a construct called a Future, which acts as a placeholder for a value that will be available later. Futures can be managed using two methods.

The first method uses callbacks. This is shown in Figure 2. A callback function takes a parameter matching the type of the Future's expected result and is triggered automatically once the Future is complete.

Figure 2. Asynchronous programming: callback method (Lodrini, 2021)

```
return http.get(url, headers: headers).then((http.Response response) {
  // We assume that an empty response means that the request found no corresponding endpoint
  // Ideally, the server should be the one to handle this.
  final int statusCode = response.statusCode;
  String source = Utf8Decoder().convert(response.bodyBytes);
  final jsonData = response.body.isNotEmpty
      ? jsonDecode((source))
      : jsonDecode('{"message": "Route not found","status": false}');
  if (statusCode < 200 || statusCode >= 400) {
    return APIresponse(
        error: true,
        errorMessage: "${statusCode.toString()} ${jsonData["message"]}");
  } else {
    return APIresponse(data: jsonData["data"]);
  }
});
```

The second method uses the await and async keywords. This can be seen in Figure 3, which presents the same functionality as Figure 2 but written with async/await.

Figure 3. Asynchronous programming: async/await method (Lodrini, 2021)

```
http.Response response = await http.get(url, headers: headers)
// We assume that an empty response means that the request found no corresponding endpoint.
// Ideally, the server should be the one to handle this.

final int statusCode = response.statusCode;
String source = Utf8Decoder().convert(response.bodyBytes);
final jsonData = response.body.isNotEmpty
  ? jsonDecode((source))
  : jsonDecode('{"message": "Route not found","status": false}');

if (statusCode < 200 || statusCode >= 400) {
  return APIresponse(
    error: true,
    errorMessage: "${statusCode.toString()} ${jsonData["message"]}");
} else {
  return APIresponse(data: jsonData["data"]);
}
```

An async function executes synchronously up to the first await, at which point it pauses until the associated Future completes, then resumes execution. The advantage of using this method is its readability. This method is easier to follow when several asynchronous operations are performed within the same function.

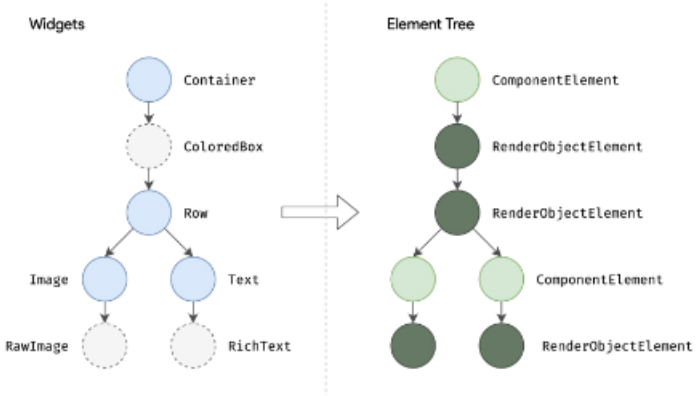
2.1.2 Flutter

In this project, the development framework Flutter (Flutter, n.d.) is used. This platform was chosen because Deuse always uses it for mobile application development. Flutter, introduced by Google in 2017, is an open-source mobile framework and software development kit (SDK) designed for building high-performance, cross-platform applications. Built on Dart, Flutter enables developers to create natively compiled apps for multiple platforms, including Android, iOS, chromeOS, desktop, and the web, using a single shared codebase.

Flutter uses a declarative approach, which builds the UI based on the app's current state. Flutter redraws the UI when the state changes and creates a new widget instance. Widgets are the building blocks of the user interface, providing the blueprint for how each part of the UI should appear and behave. These widgets are combined to form the widget tree. Each widget serves as a configuration used by Flutter to create an element, which is the visual representation displayed on the screen—these elements make up the element tree. When the application launches, the Flutter framework iterates through the widget tree, creating

and mounting the associated elements accordingly (Flutter Architectural Overview, n.d.). Figure 4 shows an example of the widget and element trees.

Figure 4. Widget tree and Element tree (Flutter Architectural Overview, n.d.)



There are two main kinds of widgets in Flutter: stateless widgets and stateful widgets. The key difference is how they manage and update their state. Stateless widgets are immutable and do not change once built, while stateful widgets can dynamically update their state and trigger UI changes (Flutter Documentation, n.d.).

A stateless widget does not store any internal state and relies entirely on the parameters passed to it. It remains unchanged unless explicitly rebuilt by its parent. These widgets are commonly used for displaying static content, such as labels, images, or decorative UI elements (Flutter Documentation, n.d.).

A stateful widget maintains a mutable state that can change over time. Unlike a stateless widget, which rebuilds only when its parent widget updates it, a stateful widget can manage its state internally. This is useful for interactive elements and any UI component that needs to respond to user interactions. A stateful widget consists of two classes: the widget class and the state class. The state class holds the widget's mutable data and updates the UI. This ensures that Flutter rebuilds only the affected parts of the UI when the state changes (Flutter Documentation, n.d.).

Every widget in the widget tree corresponds to an element in the element tree. Stateless widgets create stateless elements, which reference their associated widgets without maintaining any mutable state. Stateful widgets create stateful elements, which manage both the widget and its state object. The Flutter framework requires that each widget generate a corresponding element. Each of these elements keeps a reference to its

associated widget, resulting in the creation of the element tree as this process continues throughout the widget tree. Within this structure, a stateful element is linked to both its state object and the stateful widget it represents (Flutter Documentation, n.d.).

Using Flutter for application development has many advantages:

1. Cross-Platform Development: Cross-platform development typically requires writing and maintaining the same application in different languages and UI toolkits. With Flutter, there is a single codebase, and the compiler produces native code for various platforms (Flutter, n.d.).
2. High performance: Flutter applications are compiled to native ARM code and use the graphics processing unit (GPU). Rendering happens at 60 frames per second (fps) and 120fps for capable devices. Higher frame rates lead to smoother animations and transitions (Napoli, 2019).
3. Hot reload feature: Hot reload works by inserting updated source code directly into the running application. When changes are made, the system updates the altered classes by adjusting their modified fields or methods. Next, Flutter initiates a rebuild of the widget tree to reflect the updates in the user interface. This process eliminates the need to restart, redeploy, or fully rebuild the application, allowing the changes to appear on the screen almost instantly (Zammertti, 2019).
4. Rich set of widgets and tools: Flutter gives developers tools to build beautiful and professional applications, allowing for customisation in every aspect. The user interface can incorporate smooth animations, gesture recognition, and splash feedback effects. Flutter has a wide variety of widgets available (E.g. Layout widgets, interactive widgets, input widgets, …) (Flutter Documentation, n.d.).
5. Strong community and support: Flutter has detailed documentation, many plugins, and active forums for troubleshooting and collaboration. Google’s ongoing investment ensures regular updates, new features, and enhancements. This combination of strong community support and institutional backing makes Flutter a reliable and dynamic choice for developers worldwide.

Using Flutter also has disadvantages, such as a large app size, limited support for web and desktop and the adoption of Dart. The first disadvantage, large app size, can be a concern for some projects. Applications created using Flutter tend to have larger file sizes than native apps. The second disadvantage is limited support for web and desktop. While Flutter has made noticeable steps in web and desktop support, these platforms are still evolving and may not be as robust as mobile support. The last disadvantage is the adoption of Dart.

Even though Dart is a powerful, productive and accessible language, it is also not used as extensively as languages as JavaScript, which might present a learning curve for developers.

2.1.3 UI/UX design principles

When designing an application, usability, user-friendliness, and intuitiveness are important. The designer wants all users of the application to be able to use it efficiently without too many problems or losing too much time, making design principles essential. Following these principles will increase the capability of users to use the application.

Before explaining the design principles, the terms UI (user interface) and UX (user experience) should be described. The following definitions are based on the ISO (International Organization for Standardization). According to ISO, a UI is defined as “a set of all the components of an interactive system that provide information and controls for the user to accomplish specific tasks with the interactive system” (ISO, 2020, p. 3). ISO defines UX as “the user’s perceptions and responses that result from the use and/or anticipated use of a system, product or service” (ISO, 2018, p. 4).

Some important concepts to consider when creating a UI are a user-centred design, consistency, feedback, error prevention and recovery, flexibility and efficiency, and an aesthetic and minimalist design. A user-centred approach involves understanding the users’ needs, preferences, and behaviours. By prioritising the user experience, the solutions created feel natural, accessible, and enjoyable. Uniformity in design elements ensures that users can easily understand and interact with the interface without confusion (Dillon, 2023).

Users need clear and immediate responses to their actions. Feedback reassures users that their interactions are acknowledged and guides them on what to expect next. Every application should focus on preventing errors rather than fixing them. The design can reduce the chance of errors through validation, clear instructions, and confirmation prompts. When errors occur, the system should offer recovery options, such as undo or clear error messages explaining how to fix the issue (Nielsen, n.d.; Shneiderman et al., 2018, pp. 95–97).

UIs should cater to diverse users, from novices to expert users. Providing shortcuts allows experienced users to perform tasks more efficiently while maintaining a simple and approachable design for beginners. A practical design balances aesthetics with functionality. A clean and uncluttered interface highlights essential information, reducing cognitive load and helping users focus on what matters most. By eliminating unnecessary elements, designers create an elegant and intuitive experience (Nielsen, n.d.; Shneiderman et al., 2018, pp. 95–97).

Achieving an intuitive and practical interface requires following well-established design principles. These principles serve as guiding frameworks, helping designers create user-friendly and efficient experiences. Two of the most influential sets of guidelines in UI/UX design are Shneiderman's Eight Golden Rules and Nielsen's Usability Heuristics, which provide actionable insights for crafting interfaces that balance functionality, simplicity, and user satisfaction.

The ”Eight Golden Rules” of Ben Shneiderman are widely recognized guidelines for designing effective and user-friendly interactive systems. These principles are based on practical experience and serve as helpful recommendation for designers. The eight golden rules are (Shneiderman et al., 2018, pp. 95–97):

1. Strive for consistency. Similar actions should follow the same patterns in different situations. This includes using uniform terminology across prompts, menus, and help sections, as well as maintaining a consistent visual design throughout the application.
2. Seek universal usability. Interfaces should accommodate a wide range of users, accounting for variations in experience, age, ability, and technology access. Features like tooltips for beginners and shortcuts for advanced users can make interfaces more inclusive and effective.
3. Offer informative feedback. The system should respond to user actions with relevant feedback. While minor interactions may require only brief confirmation, more complex actions should prompt more detailed responses to inform the user of progress or success.
4. Design dialogs to yield closure. Group related actions together into meaningful sequences, and provide clear feedback at the end of each sequence to give users a sense of accomplishment and a signal that it's time to move on.
5. Prevent errors. The design should minimize the chances of mistakes, especially serious ones. When errors do occur, users should be given clear and simple steps to recover.

6. Permit easy reversal of actions. Wherever possible, users should be able to undo actions. This reduces anxiety and encourages exploration, as users feel safer trying unfamiliar options.
7. Keep users in control. Users should feel in command of the interface and be able to direct its operation. Unintended behaviours or unexpected outcomes can lead to frustration and should be avoided.
8. Reduce short-term memory load. Since short-term memory is limited, designers should avoid requiring users to remember information across different screens. Interfaces should present necessary information when and where it's needed.

The second set of UI/UX design guidelines is Nielsen's Usability Heuristics. They are broad rules of thumb and not specific usability guidelines. The ten usability heuristics are (Nielsen, 1994):

1. Visibility of system status
2. Match between the system and the real world
3. User control and freedom
4. Consistency and standards
5. Error prevention
6. Recognition rather than recall
7. Flexibility and efficiency of use
8. Aesthetic and minimalist design
9. Recognise, diagnose, and recover from errors
10. Help and documentation

The visibility of system status heuristic says that the system should always keep users informed about what is happening through clear and timely feedback. This helps users feel in control and reduces confusion or uncertainty.

The next heuristic is Match between the system and the real world. Interfaces should use language, concepts, and visuals that are familiar to users, following real-world conventions. This makes the system feel more intuitive and easier to understand.

The third heuristic is User control and freedom. Users should be able to undo or redo actions easily, especially when they make mistakes. Clear "emergency exits" allow users to backtrack without frustration.

The Consistency and standards heuristic says that designs should use familiar elements and follow platform or industry conventions to avoid confusing users. Inconsistent terms or layouts can create unnecessary learning curves.

The fifth heuristic is Error prevention. Preventing errors is better than showing error messages afterwards. The system should guide users and restrict actions to reduce the chance of making mistakes.

The next heuristic is Recognition rather than recall. Users should not have to remember information between different parts of the interface. Keeping options visible and accessible helps reduce cognitive load.

The seventh heuristic is Flexibility and efficiency of use. Interfaces should cater to both novice and expert users by offering shortcuts and customizable features. This allows experienced users to work faster without hindering beginners.

The Aesthetic and minimalist design heuristic says that interfaces should only display necessary and relevant information to avoid overwhelming the user. Clutter can distract users and make important information harder to find.

The next heuristic is to recognise, diagnose, and recover from errors. Error messages should be clear, use plain language, explain the problem, and suggest a solution. This helps users fix issues without feeling frustrated or helpless.

The last heuristic is Help and documentation. Even well-designed systems might need support resources. Good documentation should be easy to search, focused on user tasks, and provide clear instructions.

Next to the given guidelines, there are also important psychological principles to consider. These principles provide better insight into how users interact with interfaces, enabling designers to create more intuitive and compelling experiences. Concepts such as Gestalt principles, Hick’s Law and Fitts’s Law reveal how users process information, make decisions, and interact with elements, offering practical guidance for organising and optimising interface design.

The Gestalt principles highlight how humans perceive and organise visual elements as groups rather than as isolated parts. These principles include proximity, similarity, continuity, closure, and symmetry (Wertheimer, 2012, pp. 127–182).

The proximity principle says that when things are close together, viewers will associate them with one another. White space can boost visual hierarchy and information flow, contributing to easy-to-read and scan layouts (Wertheimer, 2012, pp. 127–182). The similarity principle says that if two things have the same shape, size, colour, or orientation, viewers will associate them with each other (Wertheimer, 2012, pp. 127–182).

The continuity principle says that once our eyes start tracking a particular visual path, they tend to maintain that direction until interrupted by another visual element. This motion creates a sense of flow or momentum as the eyes transition between objects. Elements arranged along a continuous line guide the eyes, making them perceived as a cohesive unit. Our eyes want to see simple closed forms instead of separate pieces (Wertheimer, 2012, pp. 127–182).

The closure principle says we can reduce the elements needed to convey visual information and reduce complexity by using closure. The last principle is symmetry. Our eyes perceive complex scenes in a way that makes them simpler by seeing symmetry (Wertheimer, 2012, pp. 127–182).

The second psychological principle is Hick's Law. This law states that "the amount of information extracted is proportional to the time taken to extract it" (Hick, 1952). This means that if there is more information to extract, it will take longer. In terms of UI/UX design, this law implies that decision-making time grows as the number and complexity of available options increases. To enhance usability, designers should aim to reduce the number of choices presented, divide complex tasks into manageable steps, and avoid overwhelming users. The designer has to be careful not to simplify too much to the point of abstraction.

The Third psychological principle is Fitts' Law. This law says that "the movement time to a target depends on the size of the target and the distance to the target" (Fitts, 1954). To shorten this movement time, bigger targets and shorter distances to a target are preferable. Enlarging the target area can be done using area cursors or expanding targets. The distance to the target can be reduced by object pointing, using the drag-and-pop technique or using different widgets like pie menus (Balakrishnan, 2004).

Another important concept in UI/UX design is accessibility. When creating an application, the designer considers the users' diversity. All users have different abilities, and some users may have specific disabilities. The designer should create an interface that is accessible to all users. The WCAG (Web Content Accessibility Guidelines) documents explain how to make web content more accessible to people with disabilities (Kosova-Alija & Manninen, 2023).

The WCAG guidelines are based on four principles of accessibility: perceivable, operable, understandable, and robust. The perceivable principle ensures that users can detect and interpret the content presented. The operable principle focuses on making all interface elements functional and navigable by users. According to the understandable principle, both the information and the interface should be easy to comprehend. Lastly, the robust principle emphasises the importance of ensuring that content remains accessible across a wide range of current and future technologies. Every accessibility principle has a set of guidelines (WCAG 2 Overview, n.d.). Table 1 presents a summary of these guidelines, copied from the WCAG Quick Reference resource.

Table 1. WCAG summary (How to Meet WCAG (Quick Reference), n.d.)

Perceivable	**Operable**	**Understandable**	**Robust**
Provide text alternatives for non-text content.	Make all functionality available from a keyboard.	Make text readable and understandable.	Maximize compatibility with current and future user tools.
Provide captions and other alternatives for multimedia.	Give users enough time to read and use content.	Make content appear and operate in predictable ways.	
Create content that can be presented in different ways, including by assistive technologies, without losing meaning.	Do not use content that causes seizures or physical reactions.	Help users avoid and correct mistakes.	
Make it easier for users to see and hear content.	Help users navigate and find content		
	Make it easier to use inputs other than keyboard.		

The guidelines have three levels of conformance. These levels say how well the guideline is met. The levels are A, AA, and AAA. Level AAA is the strictest level of conformance. For level A conformance, all level A success criteria should be satisfied. All level A and AA success criteria should be satisfied for level AA conformance. All level A, AA, and AAA success criteria should be satisfied for level AAA conformance. The designer's goal should be to achieve the highest level of conformance to the WCAG guidelines.

The UI/UX design can be evaluated using different methods. The first one is Heuristic evaluation. This method assesses the user interface based on established usability principles. The method looks at conformation with principles like the Nielsen usability heuristics, the eight golden rules, and the WCAG guidelines. This method helps identify potential issues before real users interact with the system.

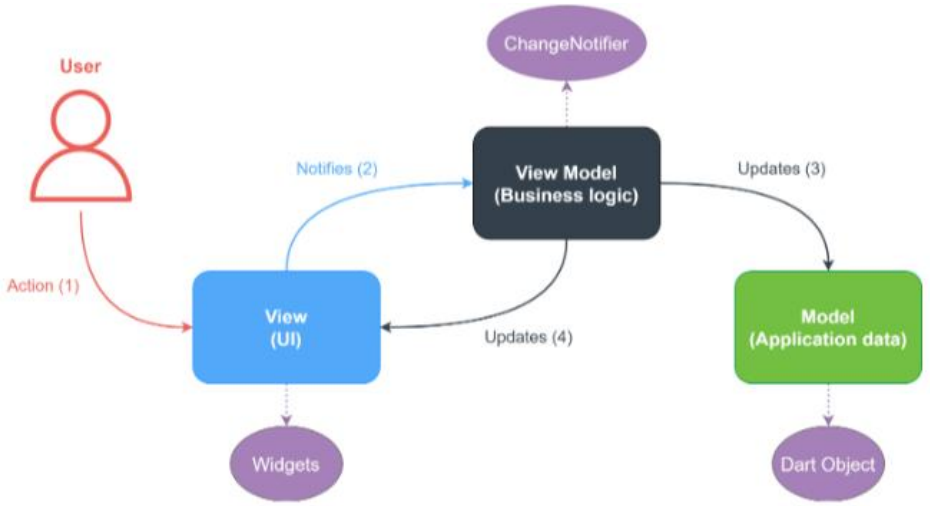
The second evaluation method is user testing. In this method, real users interact with the application to evaluate its functionality, usability, and overall experience. This method helps designers understand how actual users navigate and perceive the interface.

2.1.4 MVVM Pattern

The Model-View-ViewModel (MVVM) architectural pattern is used to implement this application. It is important to maintain a clear distinction between the UI and the logic that controls it when designing and creating applications. When the lines between the two get blurred, it results in challenges in code comprehension, making updates, or testing the application.

The MVVM pattern is a well-established design approach in software engineering that promotes the separation between an application's logic and presentation layer (The Model-View-ViewModel Pattern, 2017). Figure 5 illustrates how the core elements—model, view, and view model—interact within this structure.

Figure 5. MVVM pattern (Lodrini, 2021)



It is important to note that the MVVM pattern has many variations, and its implementation can vary for each application. For this thesis, this is how each component interacts:

- The View: This component defines the UI's structure, layout and appearance. In Flutter, the view consists solely of widgets, and a widget tree can represent its structure. In the implementation, one view was created per page, constructed from widgets that may be reused.
- The ViewModel: This component contains properties and commands that the view can bind to. It reacts to interactions from the view by updating the model and notifying the view to refresh accordingly. While the ViewModel defines the functionality provided by the UI, the view handles how that functionality is visually represented. This implementation is achieved using the ChangeNotifier. Each view is associated with its own ChangeNotifier instance, which it observes for updates.
- The Model: These non-visual classes are responsible for representing and managing the application's core data. In this implementation, most models correspond to objects that would be saved in the database when this is added. A ViewModel usually contains several Models and variables to manage the behaviour of the view.

Implementing the MVVM pattern offers more than just an organisational structure for files. It brings other benefits, especially regarding managing states and optimising the application.

2.2 Database Design

Databases are the backbone of any application, enabling efficient storage, retrieval, and data management. Database design involves selecting a database management system (DBMS) to handle the application's specifics. A well-designed database is crucial in software development as it ensures data integrity, facilitates scalability and maintenance, and simplifies the enforcement of data security measures.

There are two kinds of databases: relational and non-relational databases. Relational databases organise data into tables with defined schemas and are built around the ACID (atomicity, consistency, isolation, durability) principles. Examples of relational databases are MySQL or PostgreSQL. These databases are ideal for applications with relations between the data models.

Non-relational databases (NoSQL) manage data in more flexible formats like documents, key-value pairs, or graphs, and do not rely on fixed schemas. They prioritise scalability and performance, following BASE (Basically Available, Soft state, Eventual Consistency) principles. MongoDB and DynamoDB are commonly used NoSQL databases, well-suited for handling large volumes of unstructured or semi-structured data.

The database design and implementation are out of the initial scope of this thesis. In future work, a database could be added. In this case, the relational PostgreSQL database would be used. PostgreSQL, a powerful and open-source DBMS, is employed in this project due to its advanced features and strong community support.

2.2.1 PostgreSQL

The object-relational database management system PostgreSQL is known for its standard compliance and extensibility. It serves as a secure and efficient database server, managing workloads ranging from small single-machine programs to large applications with numerous concurrent users.

The key features of PostgreSQL are ACID compliance, scalability, extensibility, advanced data types, full-text search, and MVCC. PostgreSQL follows the ACID principles and ensures reliable transactions and data integrity. It supports large datasets and concurrent users efficiently. Another key feature is extensibility. PostgreSQL allows users to define custom data types, operators, and functions. It supports advanced data types like JSON, arrays, and hstore for diverse data storage needs. PostgreSQL also provides powerful search capabilities within text data. The last key feature is Multi-Version Concurrency Control (MVCC). PostgreSQL facilitates high levels of concurrent processing (About PostgreSQL, n.d.).

Different databases should be compared to decide on which database to choose. The first database PostgreSQL is compared to is MySQL. Both of these databases are open-source RDBMS, but PostgreSQL offers advanced features like full-text search and custom data types, whereas MySQL is often praised for its speed in read-heavy operations (About PostgreSQL, n.d.; MySQL, n.d.).

The second database PostgreSQL is compared to is SQLite. This is a lightweight, file-based database that is suitable for embedded applications. It lacks the advanced features and scalability of PostgreSQL (About PostgreSQL, n.d.; SQLite, n.d.).

The last database PostgreSQL is compared to is Firebase. This is a NoSQL, cloud-based database that offers real-time data synchronisation, ideal for applications requiring real-time updates, but it may not provide the relational data integrity features of PostgreSQL.

The advantages of PostgreSQL can be split into five categories. The first category is performance advantages. PostgreSQL efficiently manages extensive data volumes with optimised storage mechanisms. It offers various indexing techniques, including B-tree, Hash, and GIN, to enhance query performance, and it includes a sophisticated query planner and optimiser to execute queries efficiently (About PostgreSQL, n.d.).

The second category is support for complex operations. PostgreSQL is capable of handling intricate queries involving multiple tables and subqueries, and it supports advanced data types like JSON, arrays, and custom data types, allowing for flexible and efficient data modelling (About PostgreSQL, n.d.).

The third category is community. PostgreSQL is freely available with a permissive license, encouraging widespread use and collaboration. It is backed by an active community contributing to continuous improvements, extensive documentation, and many third-party tools and extensions (About PostgreSQL, n.d.).

The fourth category is security. PostgreSQL supports various authentication methods, including MD5, GSSAPI, and SSPI. It also offers SSL support for encrypting client-server communications and provides granular control over database permissions, enhancing security. The last category is compatibility with cloud services. Many cloud services offer managed PostgreSQL databases. Examples are AWS RDS, Google Cloud SQL, and Microsoft Azure (About PostgreSQL, n.d.).

PostgreSQL is thus chosen for the project because of its rich feature set, performance capabilities, extensibility, and strong community support. These characteristics make it a compelling choice for applications that require a robust and scalable relational database system.

2.2.2 Integration with Mobile Apps

Designing an effective database schema is crucial for mobile applications to ensure efficient data storage, retrieval, and integrity. Some key principles of database schema design are normalisation, indexing, defining relationships, query optimisation, and data partitioning (Database Design in DBMS, 2024).

Normalisation is organising the data, and it minimises redundancy and dependency. This process involves breaking down large tables into smaller, related ones and defining links between them. Indexes are created on frequently queried columns to enhance data retrieval speed. Proper indexing can significantly improve query performance. Establishing clear relationships between tables maintains data integrity and facilitates complex queries. Efficient queries minimise resource consumption and reduce latency. Data partitioning refers to the process of splitting extensive tables into smaller, more manageable segments to enhance system performance and simplify maintenance (Churcher, 2007).

Ensuring security in PostgreSQL involves implementing robust authentication, protecting against vulnerabilities, and adhering to data protection regulations. The first step in security is authentication and authorisation. Some examples of how this can be done are OAuth, JWT, and API keys. OAuth is a widely adopted open standard designed for secure access delegation, often utilised in token-based authentication systems. JSON Web Tokens (JWTs) are a compact and secure way to transmit information between different parties. API keys, on the other hand, are basic authentication tokens that allow access to APIs by validating client requests (Ben Natan, 2005, pp. 38–39).

Another important concept in securing the database is protecting against SQL injection and data breaches. This can be done by using parameterised queries to prevent SQL injection attacks and by sanitising and validating all user inputs to ensure they conform to expected formats (Ben Natan, 2005, pp. 149–168).

Encrypting data is also an essential step in ensuring security. Data encryption has to be done at rest and in transit. At rest, sensitive data is stored in the database using encryption algorithms. In transit, SSL/TSL is used to encrypt data transmitted between the client and server (Ben Natan, 2005, pp. 297–326).

The last security consideration is adhering to data protection regulations. For GDPR compliance, only the data necessary for the application’s functionality can be collected, explicit consent from users should be obtained before collecting or processing their data, and personal data should be anonymised to protect user identities (General Data Protection Regulation (GDPR), 2024).

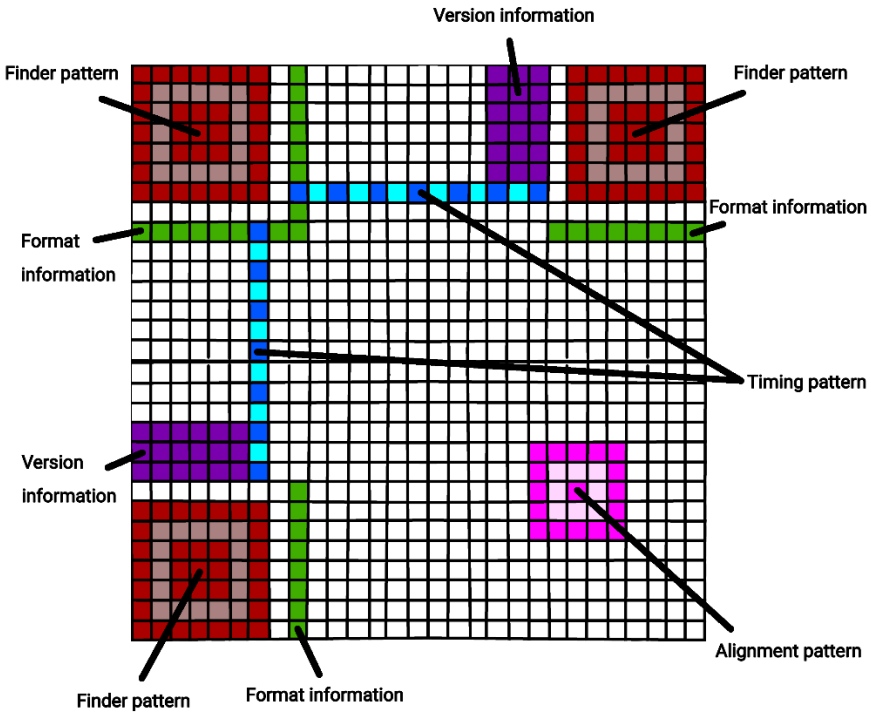
2.3 QR Codes

QR codes (Quick Response codes) are a type of two-dimensional matrix barcode that can store data efficiently and be scanned at high speed. Unlike traditional barcodes, which store limited numerical data, QR codes can encode alphanumeric characters and binary data. This allows them to store URLs, contact details, and other types of information.

2.3.1 Structure of a QR code

QR codes have a well-defined structure consisting of various patterns and modules arranged in a square grid. The key components include finder patterns, alignment patterns, timing patterns, format and version information, and data and error correction areas (Tiwari, 2016). Figure 6 shows these components inside a QR code symbol.

Figure 6. Structure of a QR Code symbol



The finder patterns are large square patterns located at three corners of the QR code. They consist of an outer dark square, an inner light square, and a solid dark square in the centre. These patterns are specifically designed to be distinct and unlikely to appear elsewhere in the code. They enable scanners to quickly locate and accurately align the QR code for proper decoding.

Alignment patterns are smaller square structures composed of a dark outer square, a light inner square, and a solid dark centre. QR codes of version 2 and above use these patterns to correct distortions.

Timing patterns are made up of a sequence of alternating dark and light modules. There are two timing patterns, a horizontal and a vertical one. These patterns help determine the density of the symbol, locate module positions, and extract version information.

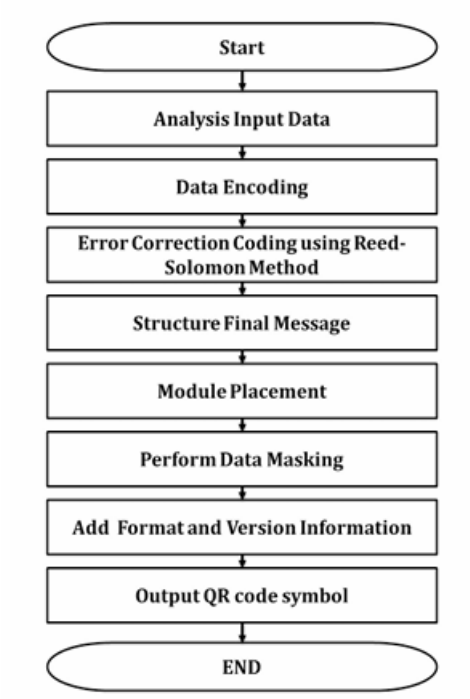
The format and version information are data required for the scanner to understand error correction levels and encoding formats. The data and error correction areas are sections where the encoded information and error correction data are stored.

2.3.2 QR code encoding

QR codes support different encoding modes to store various types of data. Numeric mode to store digits (0-9) allows up to 7089 characters. Alphanumeric mode stores letters, numbers, and a few symbols and supports up to 4296 characters. Byte mode stores binary data and allows up to 2953 bytes. The amount of data a QR code can store depends on its version. There are 40 versions, with version 1 containing a 21x21 grid and version 40 containing a 177x177 grid.

Figure 7 shows the steps in encoding a QR code. Firstly, data analysis is performed to determine the best mode for the data. Next, the data is encoded. This results in a string of bits that is split up into data codewords. Each data codeword is 8 bits long. Next, an error correction step is performed. This means that error correction codewords are generated. QR scanners read both the data codewords and the error correction codewords and can determine that they read the data correctly by comparing the two.

Figure 7. QR code encoding (Tiwari, 2016)



The next step is to organise the data and error correction codewords in the correct order. Once properly ordered, the data bits are inserted into the QR code matrix. To ensure accurate scanning, a data masking technique is applied, which modifies the code based on one of the predefined mask patterns outlined in the QR code specification. The last step is to add format and version information into specific regions of the QR code. Format information indicates the error correction level and the mask pattern used, while version information represents the dimensions of the QR matrix (Tiwari, 2016).

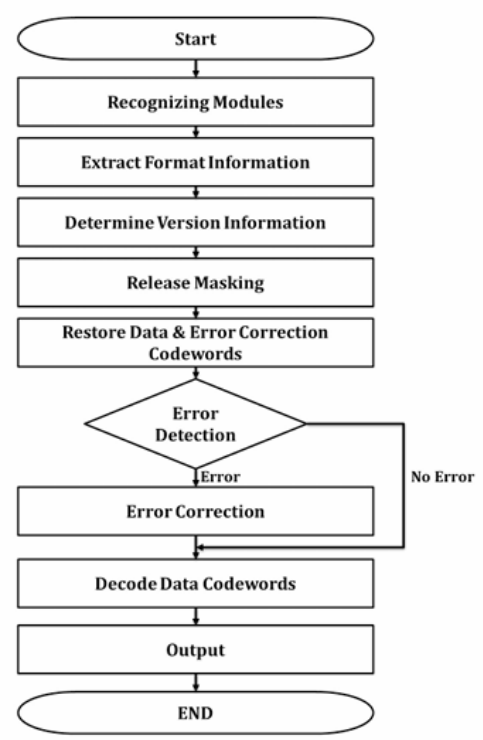
The error correction codewords are created using Reed-Solomon error correction. There are four levels of error correction. The choice of error correction level depends on environmental conditions and application requirements (Tiwari, 2016). The levels are:

- Level L (Low): recovers up to 7% of damaged data
- Level M (Medium): recovers up to 15% of damaged data
- Level Q (Quartile): recovers up to 25% of damaged data
- Level H (High): recovers up to 30% of damaged data

2.3.3 QR code decoding

Decoding data from the QR code is the reverse of the encoding procedure. Figure 8 shows an overview of the process.

Figure 8. QR code decoding (Tiwari, 2016)



The first step in the decoding process is recognising the modules. The scanner captures an image of the QR code and distinguishes the dark and light modules, interpreting them as a sequence of binary digits (“0”s and “1”s). Next, the format information is decoded, the associated mask pattern is removed, and error correction is applied to the format data. If applicable, the version information is then retrieved. The next step is releasing the masking. This is done by XORing the encoded bit pattern and the corresponding mask pattern, which is identified using the format information. The data and error correction codewords can be restored by reading the symbol characters. If any errors are detected, they can be corrected using the error correction codewords. The last step is decoding the data codewords. They are divided into segments based on mode indicators and character count values. These segments are decoded according to their respective encoding modes, and the final result is the reconstructed text (Tiwari, 2016).

2.4 Regulatory Compliance

Regulatory compliance is important in application development, especially in industries that handle sensitive user data. Following regulations ensures legal compliance, builds user trust and mitigates the risks of penalties or reputational damage. Checkinatwork requirements and the General Data Protection Regulation (GDPR) are two regulations relevant to this project.

2.4.1 Checkinatwork Requirements

Checkinatwork is a registration system in Belgium that monitors individuals working in immovable property. Everyone who works on a site with a total cost equal to or higher than €500 000 should be registered. This registration system is to counter undeclared work and to ensure that all employees are insured (Checkinatwork, n.d.; Checkinatwork, 2025).

The responsibility of registering falls on the employers, contractors, and self-employed individuals. The main contractor plays a crucial role in providing a registration system for subcontractors and ensuring compliance throughout the subcontracting chain. Daily registrations must occur before an individual begins working (Checkinatwork, n.d.; Checkinatwork, 2025).

Specific information must be provided to complete the registration process, including the worker’s personal data, contact information, and workplace location. Failure to comply with Checkinatwork obligations can lead to administrative and criminal penalties, reinforcing the importance of adherence to the system’s requirements (Checkinatwork, n.d.; Checkinatwork, 2025).

Automation of attendance recording further enhances efficiency and accuracy, reducing administrative burdens while ensuring adherence to regulations. By integrating these practices, companies can manage workforce monitoring and compliance within the Belgian regulatory framework.

2.4.2 GDPR and Data Security

The General Data Protection Regulation (GDPR) is a European Union (EU) law to improve data protection and privacy for individuals within the EU. It applies to all organisations that handle personal data. When designing user interfaces, the GDPR principles must be considered to ensure legal compliance and to maintain users' trust.

GDPR is built on several main principles that govern the collection, protection, and storage of personal data. These principles include Lawfulness, Fairness, and transparency; purpose limitation; data minimisation; accuracy; storage limitation; and integrity and confidentiality (General Data Protection Regulation (GDPR), 2024).

The first principle, Lawfulness, Fairness and Transparency, requires organisations to process personal data in a lawful manner and to be transparent about their data practices. The second principle, purpose limitation, says that personal data should be collected for clear, legitimate purposes and should not be used for other incompatible purposes. The third principle, data minimisation, states that only the personal data necessary for the specified purpose should be gathered and processed. The next principle, Accuracy, mandates that organizations take steps to ensure personal data is accurate and current. The principle of Storage Limitation ensures that personal data is not kept longer than necessary for its intended use. Lastly, Integrity and Confidentiality emphasise the importance of securing personal data to prevent unauthorized access, loss, or breaches (General Data Protection Regulation (GDPR), 2024).

The Checkinatwork application must incorporate GDPR principles to protect user data and enhance trust. Privacy by design and default ensures that data protection measures are integrated from the start, limiting data collection and securing processing. Clear and accessible privacy policies should be provided so users can easily understand how their data is handled, with policies presented in a user-friendly manner. Explicit consent mechanisms, such as opt-in checkboxes and granular user control over data sharing, should be incorporated into the design. Secure authentication, including strong encryption and multi-factor authentication, helps protect user data from unauthorised access. Furthermore, users should have straightforward access to delete or transfer their data upon request, ensuring compliance with the right to erasure and data portability.

3 Methods and Materials

The primary objective of this thesis is to design and develop a mobile application to digitalise timekeeping for construction workers, making it a development project. The project uses a structured Scrum methodology to ensure an iterative and efficient development process. The scrum methodology and the tools used for the implementation of this project are discussed in this chapter.

3.1 Development methodology: Scrum

The Scrum methodology is a widely used agile framework for iterative and incremental software development. For this project, weekly Scrum meetings with Deuse are held to review progress, discuss obstacles, and set goals for the next sprint. Each sprint lasts one week and focuses on delivering functional components of the application. The reason this methodology was chosen is because Deuse uses Scrum in the company for the development of its projects.

In this project, there are four major sprints. The first sprint is the design of the application. In this sprint, a possible design for the application should be made. The design should take into consideration the UI/UX design principles covered in section 2.1.3. The second sprint is focused on the architecture of the application. In this sprint, the architecture should be set up. This means considering the needed variables and methods and creating a main overview. The application’s login page should also be developed in the second sprint. The third sprint includes creating the application's calendar and shift pages. The last sprint focuses on the QR code part of the application. The QR code and the QR-code scanning pages of the applications will be developed.

The Scrum methodology was chosen for three major reasons. The first one is its flexibility and adaptability. The iterative nature of Scrum allows adjustments based on stakeholder feedback. The second reason is its clear progress tracking. The weekly sprints provide transparency in the project's progress. The last major reason is its enhancement of collaboration. Regular communication with the company ensures alignment of goals and lowers the barrier to asking questions.

A daily diary was also kept next to the Scrum methodology. This personal log is maintained to document daily progress, challenges, and technical decisions, which aids in reporting.

3.2 Tools and Technologies

The following tools and technologies were used for the development of the application. Some of these technologies are explained in more detail in Chapter 2.

- Android studio: Android studio was used as a development environment. This is the official Integrated Development Environment (IDE) for Android app development. It provides a comprehensive set of tools and features to help developers create, test, and debug applications efficiently.
- Dart: Dart was used as the programming language for this project. The programming language is already covered in section 2.1.1.
- Flutter: The Flutter framework is used to develop the application. In section 0, a detailed description of Flutter, including advantages and disadvantages, is given.
- GitHub: For source code management, the version control system GitHub is used. All code changes are committed, ensuring that no code or progress gets lost.
- Google Accessibility Scanner: To assess the accessibility of the application, the Google Accessibility Scanner is used to evaluate compliance with WCAG.

3.3 Ethical considerations and use of AI

This thesis deals with ethical considerations like data responsibility and information security. The use of AI is also an ethical consideration in the developmental process.

The first ethical consideration that has to be made is data responsibility. Because the application deals with the personal data of employees, it is very important to comply with GDPR regulations mentioned in section 2.4.2. Adding a secure authentication mechanism to the application helps in protecting the sensitive data and making sure only authorised people can get to the data.

The second ethical consideration to be made is information security. Even if personal data is handled responsibly, it should not be accessed by unauthorized people. Encryption will be used for data storage and transmission to ensure this does not happen. User roles and permissions will restrict unauthorised access to certain parts of the application.

In this thesis, AI was used to create the topic form and the thesis plan. After this, AI was not used anymore in writing the thesis.

4 Implementation of the Application

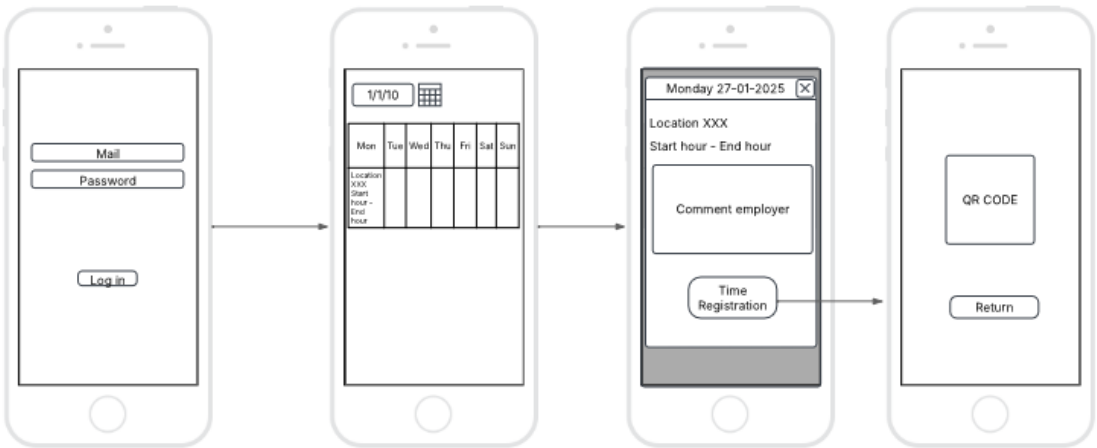
This chapter is on the implementation of the application. It explains the requirements and overall working of the solution, as well as the design and implementation steps. The implementation was split into different sprints, as mentioned in section 3.1. The implementation subsections each describe one sprint.

4.1 Requirements

The mobile application must provide user authentication, time registration, QR code generation and QR code scanning. The application can be used by two types of users: the employee and the employer. Both of these types of users get a login screen when opening the app. They should authenticate using a valid email and password combination. The next screen the users get depends on their user type.

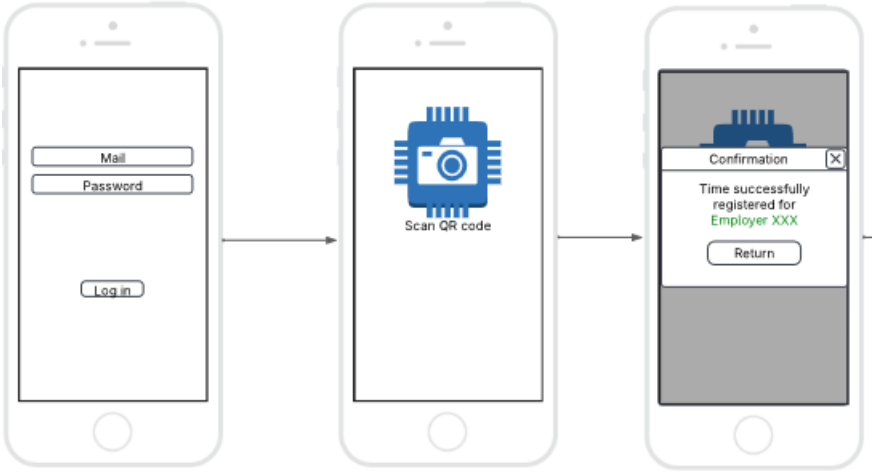
The employee gets a screen with a calendar. In this calendar, all the employee's shifts for this week are shown. When the employee clicks a specific day, he gets a more detailed screen about this shift where there can be a comment from his employer. At the bottom of this screen, there will be a button for time registration. When clicking this button, a QR code is generated. Figure 9 shows wireframes created by Deuse to explain the overall working of the employee side of the application. In the application, there is also a month calendar page. This page can be accessed from the week calendar view and will show a full month with indicators if the employee has a shift to be worked on this day.

Figure 9. Employee application wireframes



The employer has a different screen when logging in. Figure 10 shows the wireframes created by Deuse about the employer side of the application.

Figure 10. Employer application wireframes



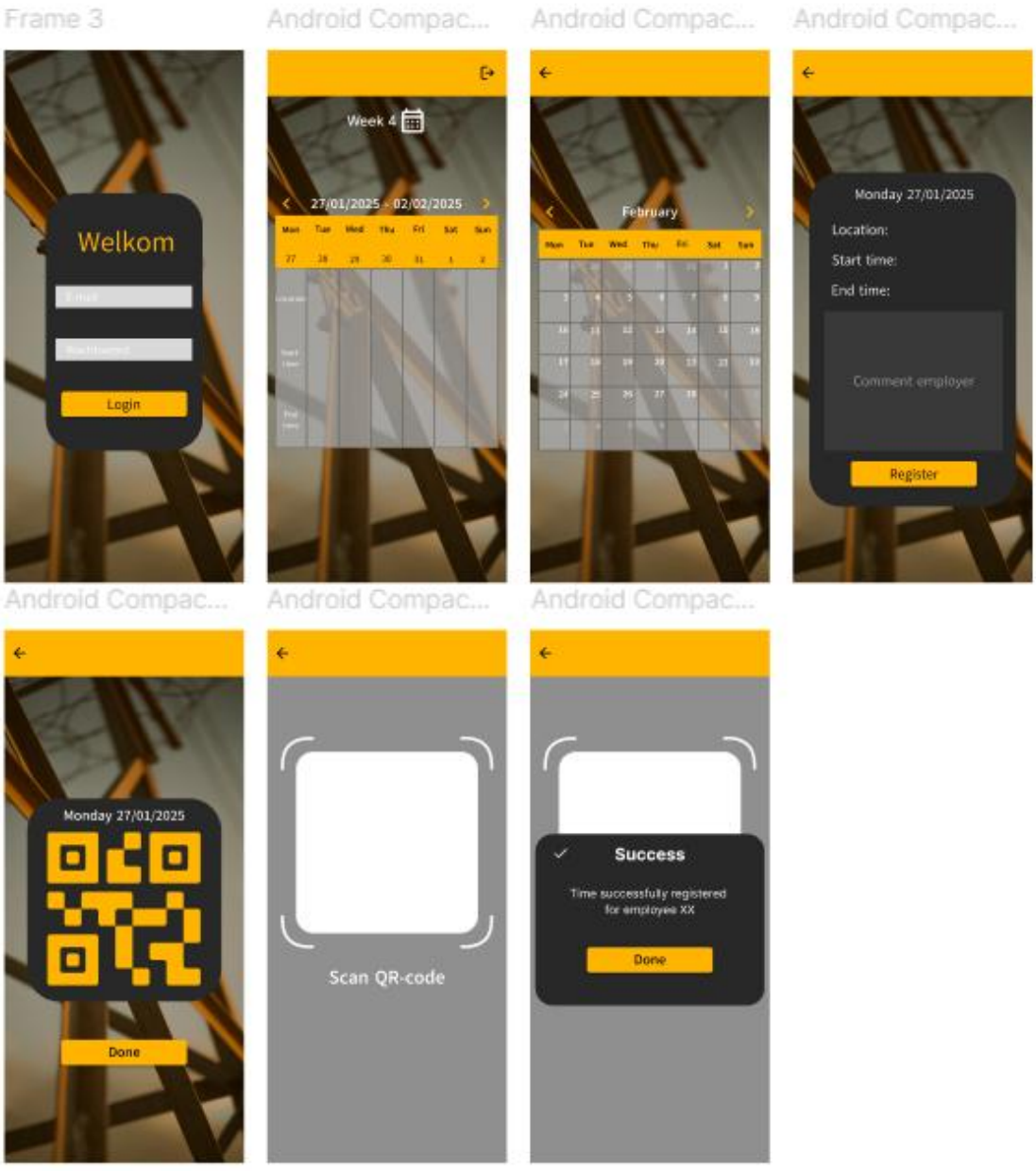
The employer gets a screen to scan a QR code. With this screen, he scans the QR code generated by the employee's application. After scanning a QR code, the employer gets a success message that the employee's registration was successful. In a later stage of the application, this data would be saved in a database and could be used by the company's ERP system.

4.2 Design

The first sprint of the project was the design phase. In this phase, a design for the entire application should be made. There are six important pages in the application. These are the login page, the week calendar view, the month calendar view, the shift detail view, the QR code view and the QR code scanning view. Each of these pages underwent three design iterations, and the final selection was based on the company's preference.

In this section, the three design iterations are shown, and a short explanation of some design choices is provided. The first design is shown in Figure 11. In this design, a background image is used. This image shows a metal construction that can be easily connected to construction. The primary colour of this design is yellow. This colour was chosen because of the colour of construction hats.

Figure 11. Design 1

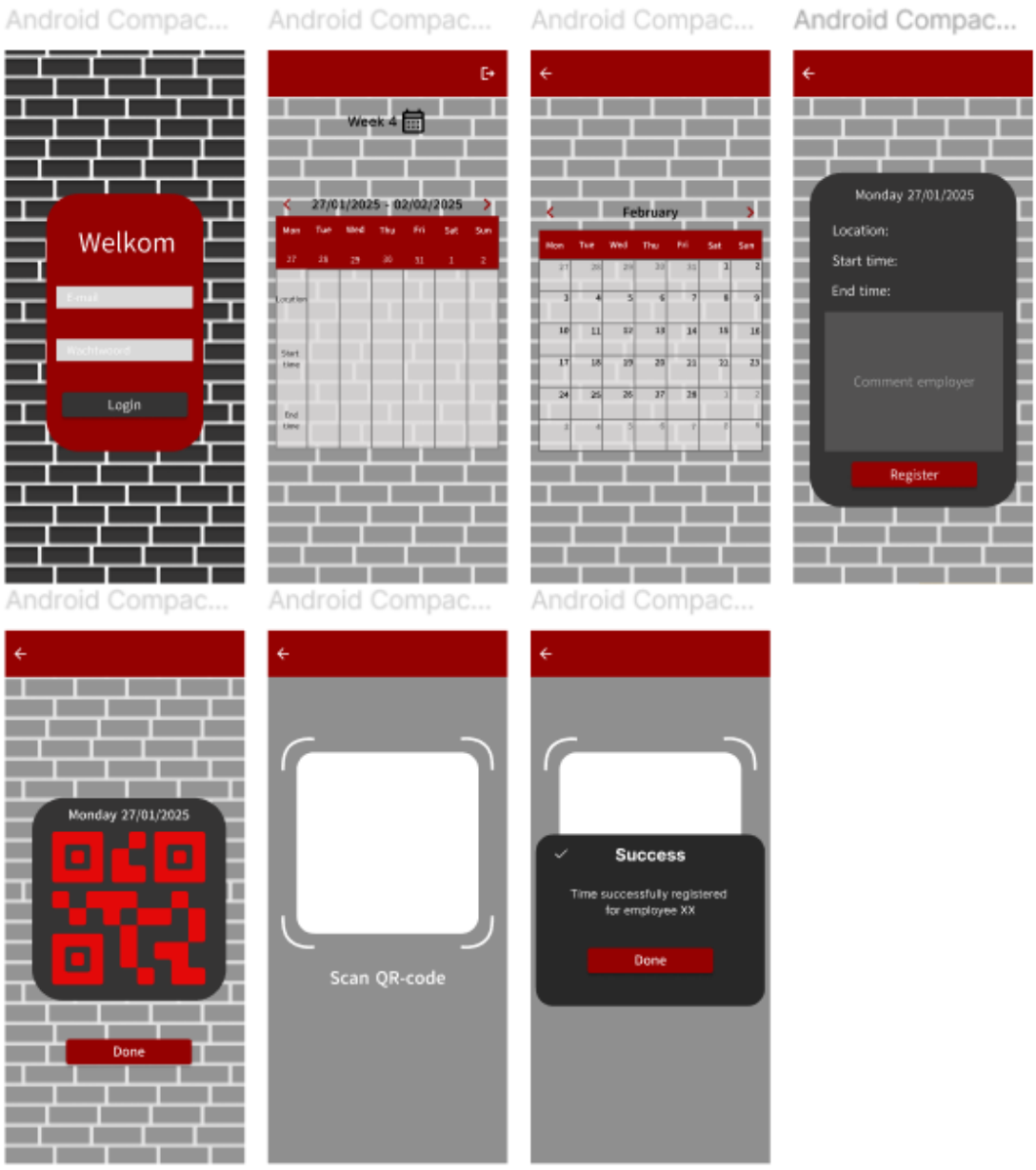


The second design is shown in Figure 12. In this design, instead of a background image, flowing shapes are used as a background. The main colour of this design is also yellow, resembling the colour of construction hats.

Figure 12. Design 2

The last design is shown in Figure 13. In this design, the background is made to resemble bricks. The primary colour of this design is dark red to resemble bricks. The main elements, like the calendar, still resemble the other designs. This design, especially the login page, gives a cluttered feeling because of the pattern on the background.

Figure 13. Design 3



After showing the three designs to the company, design 1 was chosen. This design was chosen because it was liked the most since there is not a big difference between the designs implementation-wise.

4.3 Application architecture and login

The first real implementation sprint focused on the overall architecture design and the development of the login screen.

4.3.1 Application architecture

During this sprint, the primary task was to create the foundation of the application's architecture. The application followed the MVVM architectural pattern that was explained in section 2.1.4 During the architecture design phase, all the model classes needed were created in the models.dart file. This application has three classes: a User class, a Shift class, and a CheckIn class. The attributes of these classes can be seen in Table 2.

Table 2. Model classes of the application

User	- Int **id** - String **email** - String **password** - UserType **userType**
Shift	- Int **id** - Int **userID** - DateTime **date** - String **location** - DateTime **startTime** - DateTime **endTime** - String? **employerComment**
CheckIn	- Int **id** - Int **shiftID** - Int? **employerID** - DateTime? **checkInTime** - Bool **checkedIn**

A User has an ID, an email, a password, and a userType. This type is an enum with as possible values Employee or Employer. All of the attributes of this class are required to create a new User.

For the userType an enum is chosen because this makes the application more scalable than using a Boolean. If another userType should be added in a future version of the application it is easy because an enum is used.

A shift has an ID, a userID, a date, a location, a start- and endTime, and possibly an employer comment. The userID in the shift is the ID of the user who will have to work that specific shift. When creating a new shift, all attributes except for the employer comment are required. This comment can be added by the employer when creating the shift, but it does not have to exist.

A Checkin has an ID, a shiftID, a boolean to show if there has been checked in, and possibly an employerID and a check in time. The shiftID is the id of the shift that is connected to this check-in. A check-in is created for a shift the first time an employee opens this shift. At the moment of creation, the Boolean checkedIn is false because there has not been checked in yet. There also is no employer or checkInTime yet; this information is added when the QR code is scanned, and there is officially checked in.

After creating the model classes, the views and viewModels were created. A viewModel is created for every view. In the architecture design phase, nothing was added to the views yet. In the viewModels, variables and functions were considered that could be useful for implementing the views in a later phase.

The application has two providers. Providers are classes that need to be available from everywhere in the application. In this case, there is a UserProvider and a ServiceProvider. The UserProvider has one attribute: the current user. When logging in, this user is set and can then be used throughout the whole application. When logging out, this user is set to null again. The ServiceProvider creates all services needed in the views and then makes them available throughout the whole application. It uses the singleton design pattern to make sure that all services are only created once in the application.

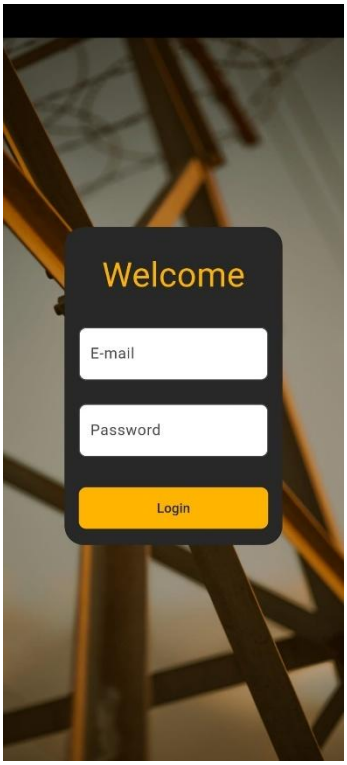
4.3.2 Login page implementation

The second part of this sprint was implementing the login view. The login screen serves as the entry point for users, ensuring that only authorised individuals can access the application. The implementation of the login page involves a LoginViewModel that manages the authentication and state, a LoginView for all UI components, and an AuthenticationService that serves as a database. When a database is added in future work, the AuthenticationService will serve as a connection between the application and the database for authentication.

The LoginViewModel class handles the logic related to authentication. It stores credentials, interacts with the AuthenticationService for user verification and updates the UserProvider with the authenticated user after logging in.

The LoginView UI provides a simple interface for logging in. It includes fields for email and password input, along with a login button that triggers authentication. Upon successful login, users are redirected based on their UserType. Employees navigate to the week calendar, while employers proceed to the QR scanning page. In case the authentication is unsuccessful, an error message is displayed. Figure 14 shows the login screen in the finished application.

Figure 14. Implemented Login screen



The AuthenticationService handles the authentication logic, verifying the credentials against a predefined set of users. If the database is implemented in future work, the credentials are verified against all users saved in the database.

4.3.3 Challenges and Solutions

A problem encountered during this phase was incompatibility between Flutter versions. Deuse provided a base project, but it was unusable due to a difference in the Flutter version. The needed directories and files were copied from the given base project to a new Flutter project to resolve this problem.

4.4 Calendar and shifts

The next implementation sprint included implementing the pages that show a calendar with shifts to be worked and a detailed page about a shift. There are two types of calendar pages. There is a page that shows a weekly view and another one that shows a monthly overview.

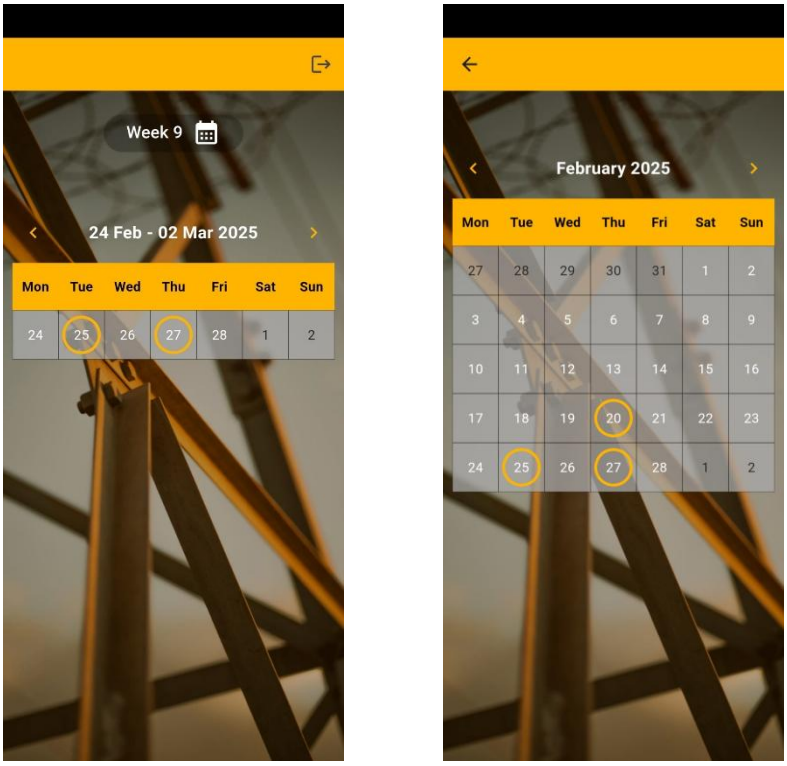
4.4.1 Calendar pages implementation

The calendar pages provide a way for employees to track their scheduled shifts. The MonthCalendarView displays shifts for an entire month, while the WeekCalendarView focuses on a single week. Users can switch between these views and select specific days to view detailed shift information.

The MonthCalendarViewModel and WeekCalendarViewModel handle the state management for these views. They manage user interactions, such as selecting a date and retrieving shifts. The TableCalendar package is used to create an interactive calendar component for the calendar view (Table_calendar 3.2.0, 2025).

When a user selects a date, the shifts assigned for that day are retrieved using the CalendarService, which maintains a record of shifts. When a database is implemented in the application, the CalendarService retrieves these shifts from the database and makes them available for the calendar view models. Figure 15 shows the week and month calendar screens in the finished application.

Figure 15. Implemented week calendar and month calendar screens



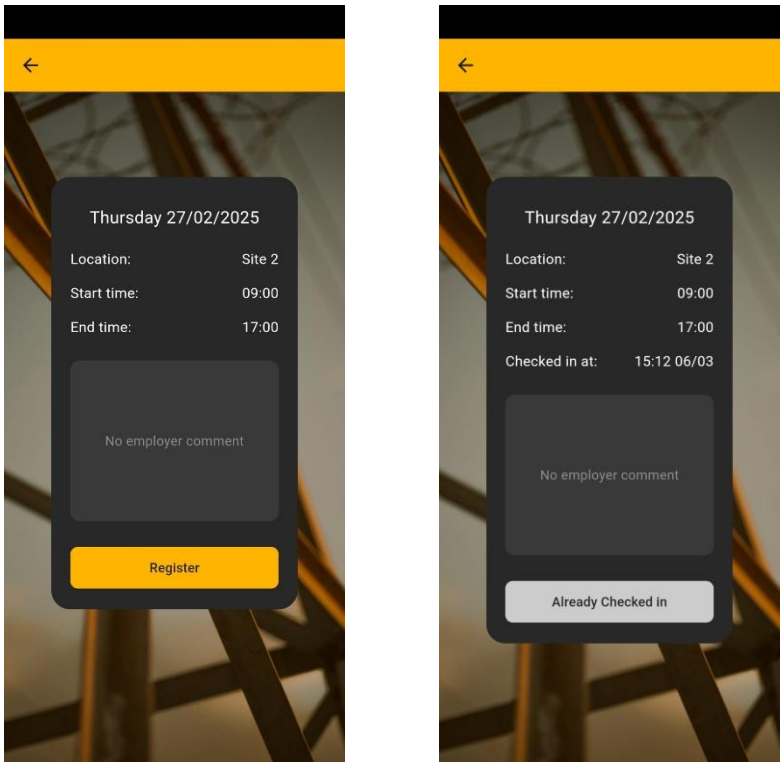
Each shift is displayed within the calendar by drawing a coloured circle around the date. This allows users to see their upcoming work schedule easily.

4.4.2 Shift page implementation

The shiftView presents detailed information about a specific shift, including the location, start and end times, and employer comments. Users can check in for their shift using the dedicated button that interacts with the CheckInService.

The ShiftViewModel handles logic such as retrieving shift details, determining whether a user has already checked in for this shift, and displaying employer comments. Figure 16 shows the implemented shift screen. The second screen in the figure shows how the screen looks when the shift has already been checked in.

Figure 16. Implemented shift screen



If a user has already checked in, the check-in time is displayed, and the button is disabled. Otherwise, the employee can initiate a check-in process by clicking the register button and navigating to the QR code that has to be scanned by the employer.

4.4.3 Challenges and Solutions

A challenge encountered during the implementation of this sprint was the issue of employees having multiple shifts on one day. In the initial design, the shift information would be inside the calendar. This could lead to a cluttered UI when there are multiple shifts in one day. This was resolved by displaying a circle in the calendar when there are one or more shifts and then displaying the detailed shift information under the calendar.

4.5 QR codes

The last implementation sprint was the QR code part of the application. For the employee part of the application, this meant the page where the QR code is shown. For the employer, this meant the page used to scan these QR codes.

4.5.1 QR page implementation

The QrView generates and displays the QR code for an employee's shift check-in. It uses the 'PrettyQrView' package to create an aesthetic QR code that contains the check-in ID (Pretty_qr_code 3.3.0, 2024). The UI also shows the date of the shift the employee is checking in to and a button for navigation back to the shift detail page. Figure 17 shows the implemented QR code screen.

Figure 17. Implemented QR screen

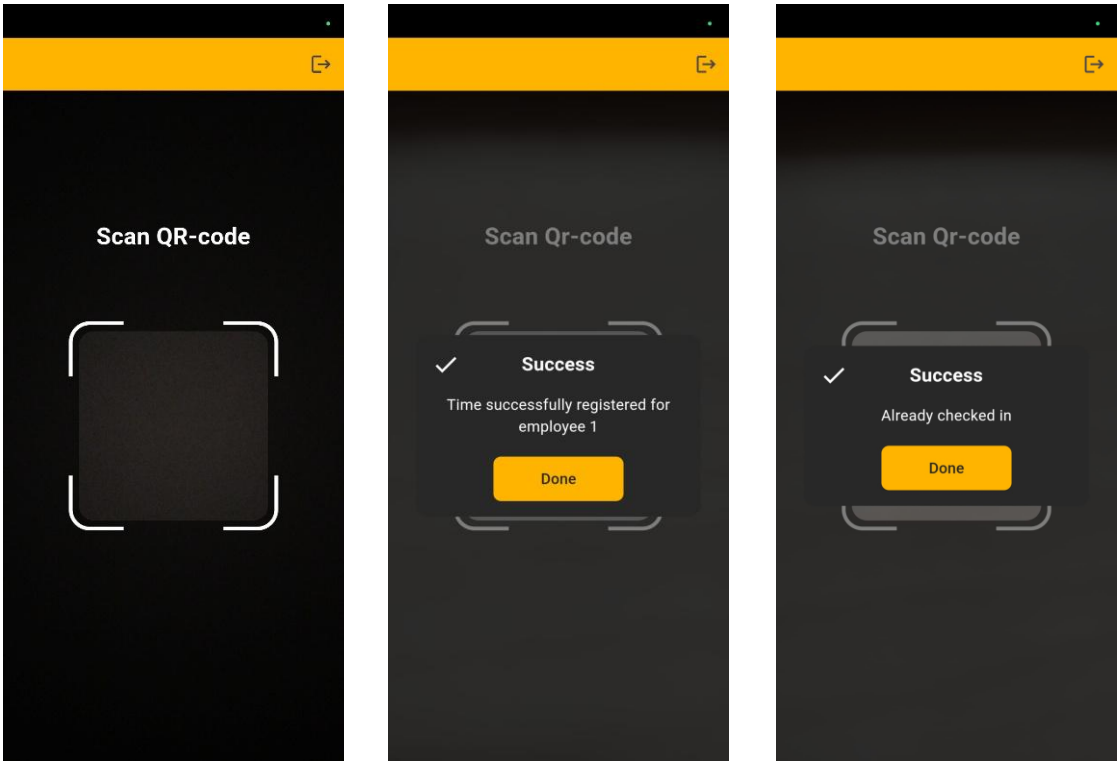
The QrViewModel manages the logic. It retrieves the relevant check-in instance and associated shift details. This ensures that the QR code accurately represents the employee's shift check-in.

4.5.2 Scan QR page implementation

The ScanQrView allows employers to scan QR codes presented by employees. When an employee's QR code is scanned in, he is checked in for that shift. The scanning feature is implemented using the MobileScanner package, which detects QR codes and processes the check-in (Mobile_scanner 6.0.6, 2025).

The ScanQrViewModel handles the QR code processing logic. It extracts the check-in ID and verifies the check-in status. If valid, the check-in time is recorded, and a confirmation message is displayed. The view model also ensures that previously checked-in employees are identified to prevent duplicate check-ins. Figure 18 shows the QR scanning page of the employer.

Figure 18. Implemented QR scanning page



The second image in the figure shows the message the employer is shown after scanning a QR code. The Third image in the figure shows the message the employer sees after scanning an already checked-in QR code.

4.5.3 Challenges and Solutions

A problem encountered in this sprint was the possibility of scanning the QR code multiple times. The check-in time would be altered every time the QR code was scanned, possibly leading to incorrect information being saved. A verification mechanism was added to the ScanQrViewModel to solve this problem. This mechanism checks if the shift has already been checked in. If it is, the UI will display a message that there has already been checked in and disable the scanning function.

5 Results

The result of this thesis is a functioning application without a database. When a database is added, the application could be used by a construction company to check in their employees when they arrive on shift.

This application could also be integrated with the ERP system of the construction company. Integrating the application's database into the company's ERP system could help with payroll administration. For the application data to be useful for integration into the ERP system, a checkout functionality should be added to the application.

Deuse will not use the finished application for a real client. This thesis was a dummy project for the author to learn how to work with Flutter and to learn the flow of creating an application from start to finish.

The Accessibility Scan to evaluate WCAG compliance identified several accessibility issues, including missing labels for screen readers, small tap targets, and duplicate item descriptions. These issues impact the application's conformance level. Based on the findings, the current version of the application does not fully meet WCAG Level A conformance. Improvements such as ensuring all interactive elements have proper labels and increasing tap target sizes should be made to achieve compliance.

Although accessibility was not the primary focus of this thesis, addressing these issues in future development would make the application more inclusive and usable for individuals with disabilities.

6 Conclusion

This thesis developed a mobile application for the digitalisation of time registration for construction workers. This application was developed using Flutter. The research focused on three main aspects: usability and design principles, technical implementation with Flutter and PostgreSQL, and security and regulatory compliance challenges.

Firstly, it was examined how design principles and usability standards can enhance the user experience in a time registration system. The application ensures an intuitive interface that minimises user errors and maximises efficiency by applying UI/UX best practices, such as Shneiderman's Eight Golden Rules and Nielsen's Usability Heuristics. The design choices made in this thesis, including a clear layout and accessibility considerations, contribute to a seamless user experience for construction workers and employers.

Secondly, the technical implementation of the application was done, focussing on the integration of Flutter. Flutter's cross-platform capabilities allowed for a single codebase to support both Android and iOS, reducing development time and maintenance efforts. PostgreSQL was identified as a robust and scalable database solution that can be used in future work. The MVVM architectural pattern further improved the maintainability and scalability of the application.

Finally, the thesis analysed the security and regulatory compliance challenges of digital time registration. Compliance with Belgium's Checkinatwork regulations was the most important part of this thesis. This ensured that the application met legal requirements for workforce monitoring. When the database is designed for future work, it is important to meet GDPR requirements, ensuring secure data handling, user authentication, and encrypted storage of personal information.

In conclusion, this project successfully developed a functional mobile application that streamlines time registration for construction companies. The combination of strong UI/UX design, a scalable technical architecture, and adherence to security and legal standards ensures that the application is practical and compliant.

Future work involves integrating a real-time database, implementing payroll automation, conducting user testing with construction workers to refine the system further, and addressing accessibility issues.

References

About PostgreSQL. (n.d.). *PostgreSQL*. https://www.postgresql.org/about/

Balakrishnan, R. (2004). *‘“Beating”’ Fitts’ law: Virtual enhancements for pointing facilitation*.

Ben Natan, R. (2005). *Implementing Database security and Auditing*. Elsevier Science & Technology. https://ebookcentral-proquest-com.ezproxy.hamk.fi/lib/hamk-ebooks/reader.action?docID=286701&ppg=8#

Checkinatwork. (n.d.). *Sociale Zekerheid.* https://www.socialsecurity.be/site_nl/employer/applics/checkinatwork/index.htm

Checkinatwork. (2025). *Working in Belgium.* https://www.workinginbelgium.be/en/checkinatwork.html

Churcher, C. (2007). *Beginning Database Design*. Apress. https://learning.oreilly.com/library/view/beginning-database-design/9781590597699/

Dart overview. (n.d.). *Dart.* https://dart.dev/overview#web-platform

Dart programming language. (n.d.). *Dart.* https://dart.dev/

Database Design in DBMS. (2024, September 24). *Geeksforgeeks.* https://www.geeksforgeeks.org/database-design-in-dbms/

Dillon, A. (2023). *Understanding users: Designing Experience through layers of meaning*. Routledge.

Fitts, P. (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, *47*(6). https://www2.psychology.uiowa.edu/faculty/mordkoff/infoproc/pdfs/Fitts%201954.pdf

Flutter. (n.d.). *Flutter.* https://flutter.dev/

Flutter architectural overview. (n.d.). *Flutter.* https://docs.flutter.dev/resources/architectural-overview

Flutter documentation. (n.d.). *Flutter Docs.* https://docs.flutter.dev/

General Data Protection Regulation (GDPR). (2024, April 22). *Principles relating to processing of personal data*. https://gdpr-info.eu/art-5-gdpr/

Hick, W. E. (1952). On the rate of gain of information. *Quarterly Journal of Experimental Psychology*, *4*(1), 11–26.

How to Meet WCAG (Quick Reference). (n.d.). *W3C*. https://www.w3.org/WAI/WCAG22/quickref/

International Organization for Standardization. (2018). *Ergonomics of human-system interaction—Part 11: Usability: Definitions and concepts* (ISO Standard No. 924111:2018).

International Organization for Standardization. (2020). *Ergonomics of human-system interaction—Part 110: Interaction principles* (ISO Standard No. 9241110:2020).

Kosova-Alija, M., & Manninen, T. (2023). *The Importance of Web Accessibility for Users, Developers, and Businesses*. https://unlimited.hamk.fi/yrittajyys-ja-liiketoiminta/the-importance-of-web-accessibility-for-users-developers-and-businesses/

Lodrini, G. (2021). *ATHLETin: Web module for the management of athletes' training calendar and medical appointments*. Liège university.

Mobile_scanner 6.0.6. (2025, February 4). *Pub.Dev*. https://pub.dev/packages/mobile_scanner

MySQL. (n.d.). *MySQL.* https://www.mysql.com/

Napoli, M. (2019). *Beginning Flutter*. Wrox. https://learning.oreilly.com/library/view/beginning-flutter/9781119550822/f07.xhtml

Nielsen, J. (n.d.). *Ten Usability Heuristics* (Semantic Scholar.). https://pdfs.semanticscholar.org/5f03/b251093aee730ab9772db2e1a8a7eb8522cb.pdf

Nielsen, J. (1994). *10 Usability heuristics for user interface design*. Nielsen Norman Group. https://www.nngroup.com/articles/ten-usability-heuristics/

Pretty_qr_code 3.3.0. (2024, March 10). *Pub.Dev*. https://pub.dev/packages/pretty_qr_code

Shneiderman, Plaisant, Cohen, Jacobs, & Elmqvist. (2018). In *Designing the User Interface: Vol. Sixt edition* (pp. 95–97). Pearson.

SQLite. (n.d.). *SQLite.* https://www.sqlite.org/index.html

Table_calendar 3.2.0. (2025, January 8). *Pub.Dev.* https://pub.dev/packages/table_calendar

The Model-View-ViewModel Pattern. (2017, July 8). *Microsoft Learn.* https://learn.microsoft.com/en-us/previous-versions/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

Tiwari, S. (2016). *An Introduction To QR Code Technology*.

WCAG 2 Overview. (n.d.). *W3C.* https://www.w3.org/WAI/standards-guidelines/wcag/

Wertheimer, M. (2012). *On Perceived Motion and Figural Organization*. MIT Press. https://ebookcentral-proquest-com.ezproxy.hamk.fi/lib/hamk-ebooks/reader.action?docID=3339476&ppg=198#

Zammertti, F. (2019). *Practical Flutter*. Apress. https://learning.oreilly.com/library/view/practical-flutter-improve/9781484249727/html/Cover.xhtml

Appendix 1: Data management plan

Description of thesis research data

The thesis research data exists of notes from scrum meetings, self-written code and possibly code examples from the internet.

Management and storage of the research data

The notes from the scrum meetings with the client will be stored in the thesis author's daily diary on the author's own password-protected computer. The code written by the author will be stored on a private GitHub repository. In addition to the thesis author, the thesis supervisor and the company may also handle the data.

Processing of personal data and sensitive data

No personal data is collected in the thesis research.

Ownership of research data

The thesis data is owned by the author, since it is code written by the author.

Further use of research data after the completion of the thesis

The research data will not be reused. The thesis author will securely store the data for one year from the date of thesis approval to ensure the results can be verified if necessary, and then the data will be securely destroyed.