Jasmin Partanen

SmartAccess: Developing a Secure Access Control System Mobile Interface

Metropolia University of Applied Sciences

Bachelor of Engineering

Information technology

Bachelor's Thesis

30 Jan 2024

Tiivistelmä

Tekijä:	Jasmin Partanen
Otsikko:	Smart Access: Suojatun kulunvalvontajärjestelmän mobiilikäyttöliittymän kehittäminen
Sivumäärä:	32 sivua
Aika:	29.1.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Information Technology
Ammatillinen pääaine:	Mobile Solutions
Ohjaajat:	Joseph Hotchkiss, Projekti insinööri

Projektin tarkoituksena oli suunnitella ja kehittää turvallinen pääsynhallinnan käyttöliittymä, jonka avulla kerrostalojen asukkaat voivat myöntää väliaikaisen pääsyn rakennuksen tiloihin luomalla nelinumeroisia aikarajoitteisia pääsykoodeja. Tutkimuksen lähtökohtana oli havainto siitä, että perinteiset kerrostalojen pääsyjärjestelmät eivät kykene tarjoamaan asukkaille mahdollisuutta kontrolloida, kuka pääsee sisälle rakennukseen ja mihin aikaan.

Projekti toteutettiin vastaamaan turvallisuushuoliin, jotka liittyivät kasvavaan väestöön ja tarpeeseen modernisoida kerrostalojen nykyiset pääsyhallintajärjestelmät. Tutkimuksessa hyödynnettiin perusteellista kirjallisuuskatsausta, käyttäjäkyselyitä sekä työkaluja ja teknologioita, jotka mahdollistivat mobiilisovelluksen ja sen taustapalvelun suunnittelun. Projekti keskittyy pääsynhallintajärjestelmän mobiilisovellukseen sekä sen taustapalvelun kehittämiseen.

Tutkimuksen tuloksena valmistui pääsynhallinnan käyttöliittymän prototyyppi, jolla on merkittävä potentiaali tulevaisuuden pääsynhallintajärjestelmien kehityksessä. Tämä tutkimus luo perustan tulevalle kehitykselle ja tutkimukselle älykkäiden pääsynhallintajärjestelmien alalla. Tutkimuksen tuloksilla on käytännön vaikutuksia yrityksien, laitoksien sekä yksityishenkilöiden turvallisuuteen, ja samalla se tarjoaa käyttäjäystävällisen ja etähallittavan pääsynhallintajärjestelmän.

Yhteenvetona voidaan todeta, että tämä tutkimus osoittaa, että älykäs pääsynhallintajärjestelmä tarjoaa korkean tason turvallisuutta mutkattomasti tehden siitä ihanteellisen ratkaisun asuinalueille ja yrityksille.

Avainsanat: Pääsynhallinta, käyttöliittymä, älylukitus

Abstract

Author:	Jasmin Partanen
Title:	Smart Access: Developing a Secure Access Control System Mobile Interface
Number of Pages:	32 pages
Date:	29 January 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Professional Major:	Mobile Solutions
Supervisors:	Joseph Hotchkiss, Project Engineer

The goal of this project was to design and develop a secure access control user interface that allows apartment building residents or authorised personnel to grant temporary access to their buildings by generating four-digit access codes. The starting point of this study was the observation that traditional building access systems lack the ability to provide users the option to selectively allow guests to access the residential buildings.

The project was carried out to address the security concerns arising from the growing population and the need to modernise the building's access systems. The study involved a thorough literature review and data collection through surveys, as well as the use of various tools and technologies to design and develop the smart access control system, including a mobile application and its backend service.

The outcome of the study was a prototype of an access control user interface with significant potential to revolutionise building access control systems. This study provides a foundation for future development and research in the field of smart access control systems. The study's findings have practical implications for companies, institutions and individuals seeking to enhance security in their buildings while also providing a user-friendly and flexible access control system that is remotely manageable.

In conclusion, the smart access control system developed in this project provides a high level of security, convenience, and flexibility for its users, making it an ideal solution for residential complexes and companies alike.

Keywords: access control, mobile, user interface, smart lock

Contents

List of Abbreviations

BLE:	Bluetooth Low Energy
IAM:	Identity and Access Management
TDD:	Test-Driven Development
API:	Application Programming Interface
UI:	User Interface
AWS:	Amazon Web Services
SNS:	Simple Notification Service
NoSQL:	Not only Structured Query Language
SQL:	Structured Query Language

1 Introduction

In an era dominated by digital advancements and heightened concerns for security, the evolution of access control systems becomes imperative. Traditional methods of physical access control are increasingly being augmented or replaced by sophisticated digital solutions, and the integration of mobile interfaces is at the forefront of this transformation. This final year project studies the innovative realm of access control systems with a focus on generating access codes through a mobile application.

The primary objective is to enhance existing access control methods by implementing a functional prototype of how the residents of apartment buildings could manage access codes themselves. In addition, the result will suggest how new access codes could be delivered to visitors securely.

To succeed with the objective, access control systems are studied. User survey analysis was also carried out to ensure that the suggested solution is unique. On top of this, the topics of authentication, digital identity and access management are discussed to understand what is needed to have a real-life working prototype.

2 Theoretical Background

The world's population reached 8 billion at the end of the year 2022, from which around 750 million people statistically live in Europe and 107 million people in Northern Europe. Finland's population is around 5.6 million. The population increase of 100 thousand people in the past 5 years is a sum of several factors such as international migration and increased birth rate. [1; 2.]

The growth itself increases the need to reanalyse how access control in residential buildings in major areas is implemented and if it is secure enough / if the buildings are secure enough. Providing the apartment buildings with regular door access codes is not the only solution to managing access as the residents of the buildings tend not to protect the access codes like passwords. Instead, the access codes are often shared freely to friends, family, and other third-party personnel (unknown people, e.g. fast-food couriers). Once a door lock is installed and access codes are set, it is difficult to control how often the access codes are revealed to outsiders. This alone creates an alarming security threat to the people living in buildings where the access code is exposed, and therefore a better way of using and sharing the door access codes need to be found. [3.]

2.1 Access Control Systems

Access control is a way to manage who can access resources or a place protected by an access control system. In general access control works as a security measure for buildings or companies and it can be divided into physical and logical access control. Physical access control is something that can physically be touched. It can be an access to work or home whereas logical access control is untouchable such as file systems or data. Physical and logical access control consist of several types of access control but the main six are:

- **Attribute-Based Access Control**
  Access is given based on persons' attributes rather than rights. The attributes can be anything and anyone. They can describe a person who is trying to access, for example by age or job title.
- **Discretionary Access Control**
  Access is given without a specific reason but with an authority of a person who owns the protected area.
- **Mandatory Access Control**
  A person is given a single authority depending on their status. In this the person might only obtain partial access to certain components.
- **Role-Based Access Control**
  Person obtains certain privileges or roles that entitle the person for access.
- **Rule-Based Access Control**
  Access is granted according to a set of criteria's such as location or time of the day.
- **Break-Glass Access Control**
  Access is granted to a person, who normally does not have an access, but will have it in certain situations, such as in emergencies.

Different types of access control can be implemented in various ways, but mostly they include at least software and, in some cases, a physical reader for authority verification purposes. Software products can be server-based, web-based, or cloud-based, depending on the company's needs. For instance, if the company has many locations in different countries, they will most likely have a server-based access control in some part of their system architecture.

In addition to the software-based access control, physical devices can be installed at the doors that need controlling. These so-called physical door access control readers are user interfaces for users to interact with and they can be divided into:

- **Keypads**
  They require a PIN or passcode to open but are not dependable on a physical key or a key card which can be easily stolen.
- **Swipe card reader**
  A reader authenticates people by reading a magnetic stripe on a key card or ID and gives access if the permission is granted.
- **Radio frequency identification**
  They use signal emitting data tags that can communicate with readers nearby. These data tags can typically be key cards or key fobs.
- **Biometric locks**
  Locks identify people by a personally unique physical trait, such as eyes, facial recognition, or a fingerprint.
- **Smart locks**
  Locks support mobile identification, key cards, key fobs and touchless activation.
  [4.]

Smart locks and mobile access control are an emerging trend, where the smart phone is used instead of physical keys and fobs to manage users' access rights and credentials to different places. Smart phones can communicate with the door readers through the Bluetooth and Near-Field Communication (NFC) technology. [5.]

Bluetooth is a short-range wireless technology that operates in two major frequencies varying from 2400 - 2483.5 MHz and 2402 - 2480 MHz. Over time Bluetooth has evolved dramatically from its initially released version 1.0 in 1999 to the latest version 5.3 released in 2021. Throughout the release history, the objective has been centred around enhancing speed and security. There was a significant milestone with Bluetooth version 4, as low-power Bluetooth called Bluetooth Low Energy (BLE) was introduced. [6.]

Currently, Bluetooth can connect devices with pairing or non-pairing centric mode. The difference between these two modes is that if the connection is allowed in pairing mode (versions below 4), the devices need to be linked before

being able to communicate whereas in non-pairing mode (versions above 4) this is not required (see Figure 1).

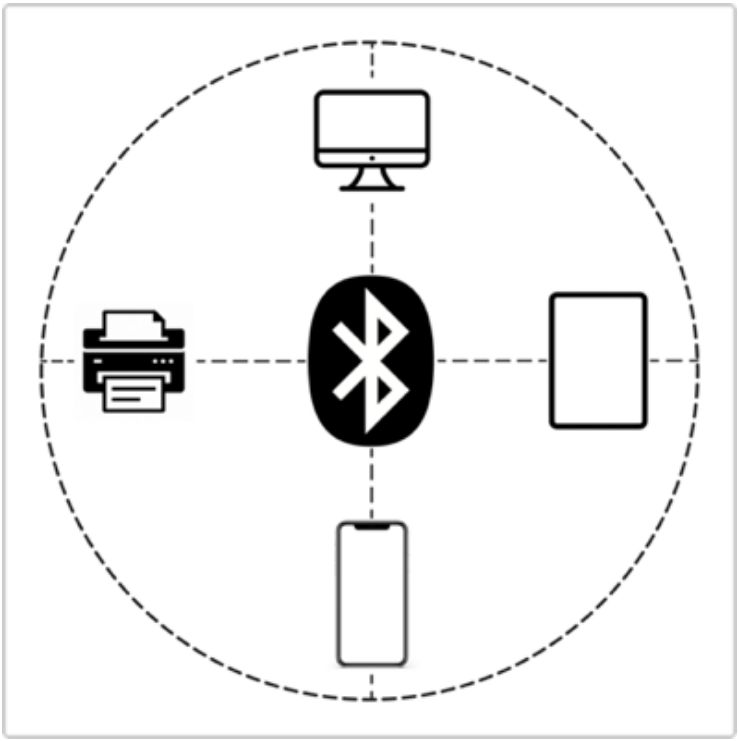

Figure 1 Communication with Bluetooth. BLE connects devices together allowing them to communicate between each other without the need for pairing. [6]

After BLE was introduced, a wider selection of devices has been able to use Bluetooth for security, wearables, and portable systems due to improved energy efficiency and performance. As seen in Figure 1, all the devices in the same Bluetooth radar have the possibility to transmit data with each other while consuming less battery. [6]

The NFC technology is an evolution from a similar technology called Radio Frequency Identification, which uses electromagnetic waves to capture and read transmitted data.

The functionality of NFC consists of readers which are integrated into devices such as smartphones, tablets or dedicated NFC readers and tags which contain an embedded microchip and antenna, also found in smartphones and other wearable devices. (see Figure 2).

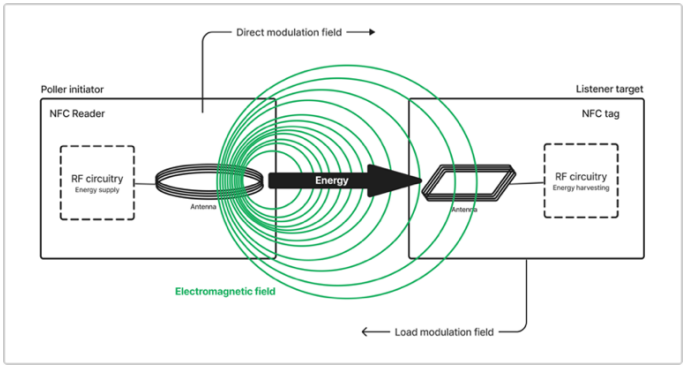

Figure 2 Communication with NFC. Near field technology uses electromagnetic waves to handle transmitted data. [7]

As in Figure 2, the tag holds credential information which can be used to unlock a door when brought close to a compatible reader. The maximum distance for the electromagnetic field to work is around 4 cm. [7.]

2.2 Authentication, Digital Identity, and Access Management

Authentication is one of the access control mechanisms used to define whether individuals or objects have permission to do or access a specific thing. Other processes of access control mechanism include:

- **Identification**
  Identification tells who the user is.
- **Authorization**
  Authentication tells if the user has permission to access or not as well as helps to confirm the user's real identity.
- **Accounting**
  Authorization defines the limitations on the user rights, such as what details one can see, and accounting keeps record of the number of entries, actions, and timestamps.

In the authentication process, the user is required to provide proof of who they are by using authentication or so-called credentials. This means that if the user is claiming to be someone, they need to provide evidence accordingly. The most typical and commonly used factors are:

- **Something generally known:**
  The user provides information only they know. It can be a PIN code or password.
- **Something possessed:**
  The user provides an item they have, such as a smart card.
- **Something considered as a physical trait:**
  The user has a physical trait that can be used as part of the authentication, such as a fingerprint.
  [8.]

Different authentication and identification methods have increased in mobile development as the regular username and password are not the only way to know if the user is legitimate. In mobile environments different types of identity solutions, such as Stripe and Signicat are used to make stronger authentications. These identity solutions provide online identity APIs to confirm the identity of users from attributes such as name, address, and date of birth to prevent them from being fraudulent. [9.]

On top of this, different identity and access management (IAM) platforms are used to store user information and to provide seamless application experiences to users. IAM platforms offer password vaults for storing and maintaining access to applications, creation, and management of user identities in applications and databases as well as enforcing security for secure access to applications. IAM solutions, such as Okta, provide the Single Sign-On feature which can be utilised to give the user a wider spectrum of login options, such as passwordless or PIN code login. [10.]

3 Tools and Methods

The project was completed following the Agile methodologies, such as Scrum. The project board and roadmap were managed in the Jira project management tool and documents were stored in a Google Drive folder. The project was divided into six parts in the following manner:

- **Planning**
  Planning included creating a vision of the project and planning the scope of the project. In addition, the user survey questionnaire was compiled and analysed.
- **Design**
  The design phase included the whole system design, from user interface (UI) design to the database schema and the Application Programming Interface (API) architecture.
- **Implementation Cloud**
  The Cloud implementation consisted of everything related to the backend and Amazon Web Services (AWS). These included creating mock data in the DynamoDB and configuring AWS Lambdas and API Gateway as well as Simple Notification Service (SNS).
- **Implementation Mobile**
  Mobile implementation included everything related to the frontend, such as Test-Driven Development (TDD) on the mobile application.
- **Testing**
  The testing phase included writing unit testing, overall system testing, increasing the test coverage and handling errors in both frontend and backend. In addition, code was simplified.
- **Review**
  The review phase of the project included preparing the project for submission.

3.1 User Survey Analysis

User survey analysis was used in this project to support the objectives and arguments for the proposed need of access control system modernization.

Residents of different apartment buildings around Finland were asked to fill out an online questionnaire, where the goal was to analyse answers to the following three questions: How many times door codes, if any, are changed in apartment buildings; how safe residents feel in comparison to the current access control solutions; and what the factors are to increase safety. A scale of 1 (feeling extremely bad) to 5 (feeling extremely good) was used to analyse the answers, with the combination of open-choice and multiple-choice answers. The survey reached 71 residents out of which 32.4% were identified as men and 67.6% as women.

Out of all the answers, 67.6% of the residents informed that the code in their building has never changed or the building does not have one. (see Figure 3).

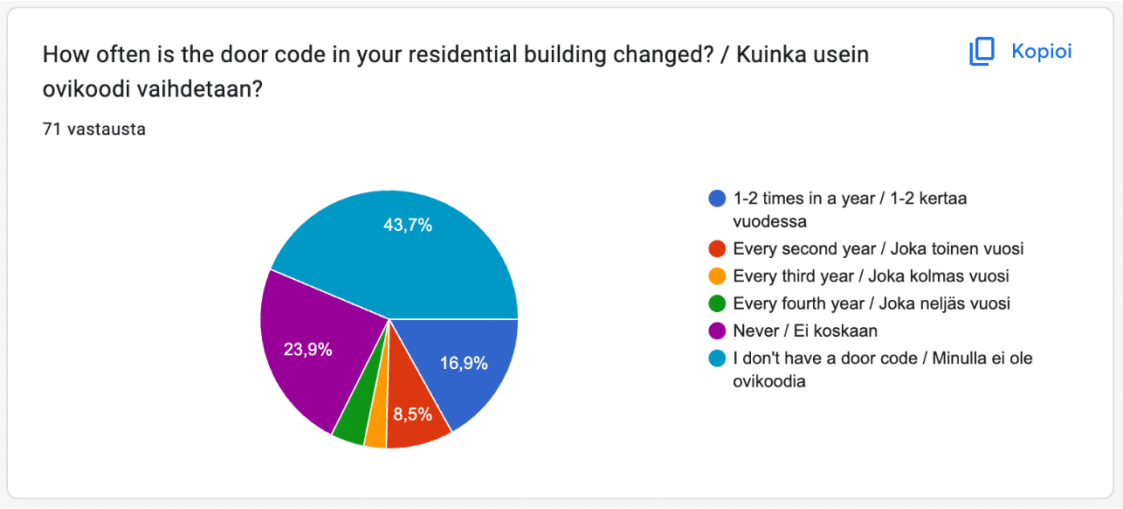

Figure 3 Frequency of changing door codes in residential buildings. Over half of the informants reported that the code is never changed, or the building does not have one.

While 16.9% of the interviewees claimed, as Figure 3 illustrates, that the code is changed frequently 1-2 times a year, most of the answers indicate a high

possibility of having the code memorized by the wrong people as the change frequency is long.

According to the survey, most of the people had lived in their current building for 1-2 years (40.8%), 33.8% less than a year and 25.4% over three years.

The answers show that most interviewees felt comfortable in their building. However, the current access control solutions leave 46.4% of the residents feeling between not safe at all to moderately safe. (see Figure 4)

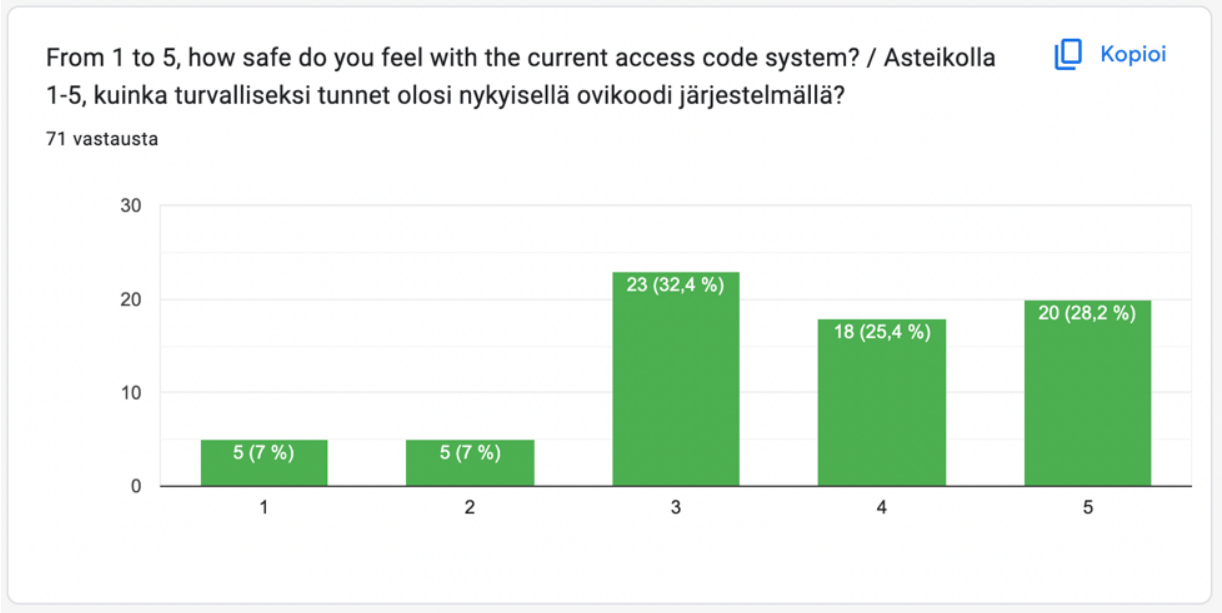

Figure 4 Distribution of answers on how safe people feel with current access code systems. 46.4% of the interviewees admitted feeling not safe at all to feeling moderately safe.

The question in Figure 4 shows that some respondents felt that the public code should be changed more often whereas some thought that the door should always be kept locked. A few respondents pointed out that the public code meant for residents should be kept private.

Overall, 86% of the interviewees expressed enthusiasm for letting friends and family inside without sharing the actual door code with the variation from moderate excitement to extreme excitement.

3.2 Test-Driven Development with Swift for iOS

Test-Driven Development (TDD) is a methodology used when a feature is needed and for developing that feature new code needs to be written or alternatively something needs to be fixed, and to make a fix, code needs to be rewritten.

The approach in TDD is that the tests for changing code are written before the actual code. This is done in three steps such as red, green and refactor.
(see Figure 5).

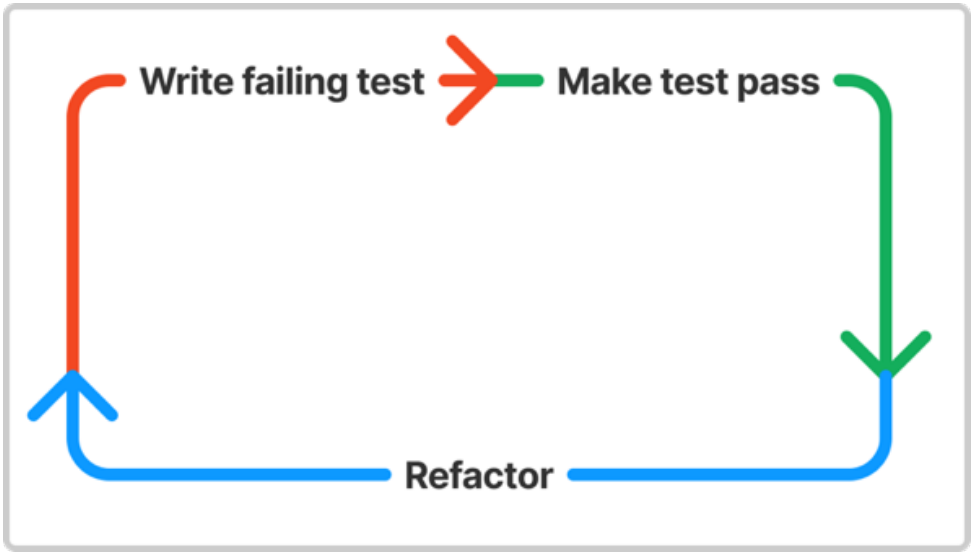

Figure 5 Test driven development flow. Tests are created before the actual code. The first step is to make the test fail. [11]

As Figure 5 shows, the beginning of every new piece of code is a failing test, after which a minimal amount of new code is written to make the test pass. Only at the end, the passing code is refactored. For Swift, tests can be written with XCTest, which is a testing framework for Xcode. Tests are divided into four different testing levels, and these are listed below from bottom to top:

- Unit
- Component

- Integration
- UI and API test

  (Cf. Figure 6.)

Figure 6 Test types. At least unit tests should be carried out. [12]

Each test level serves a different purpose in the software development process and the amount of code it can affect is individual, as illustrated in Figure 6. The tests are run in different phases or some in parallel, but generally they all test different things:

- **Unit tests**

  Unit tests are used to test a single unit, such as a function that has a single purpose. Generally, they are run as part of a build and before every commit.
- **Component test**

  Component tests are used to test a single component.
- **Integration and End-to-end test**

  Integration and end-to-end tests are very similar as they are used to test a bigger part of an application, such as how multiple components or views work together.
- **UI and API test**

  UI testing focuses on how the UI is functioning, and API testing validates if backend logic works.

Other types of testing in mobile development can be:

- **Performance testing**
  Performance tests can be done with the Xcode Instruments tool, which offers for example time profiling. This shows the developer if any part of the application loads slowly and needs changing.
- **User testing**
  User testing is done before development. Its main purpose is to find out who the users are and how the mobile application can be made to meet their requirements. [11; 12]

3.3 Amazon Web Services

Amazon Web Services (AWS) provide a coarse selection of tools to use on the backend side, which can be used simultaneously together or alone. Some of the tools are DynamoDB, AWS Lambdas and API Gateway, which can be used together to create a monolithic backend architecture.

3.3.1 DynamoDB

DynamoDB is a popular Not only Structured Query Language (NoSQL) database offered by AWS. It is designed with performance in mind as a nonrelational database which can handle simple queries. Data is structured in collections instead of tables and rows as in Structured Query Language (SQL) based on a relational database. It is ideal especially for storing many rows of similar data, such as website clicks or voting.

The idea of DynamoDB is that it stores the data in the same format in which the application is going to use it, and in this way, the developer avoids complicated queries through many tables, when all data is fetched from a single table (see Figure 7).

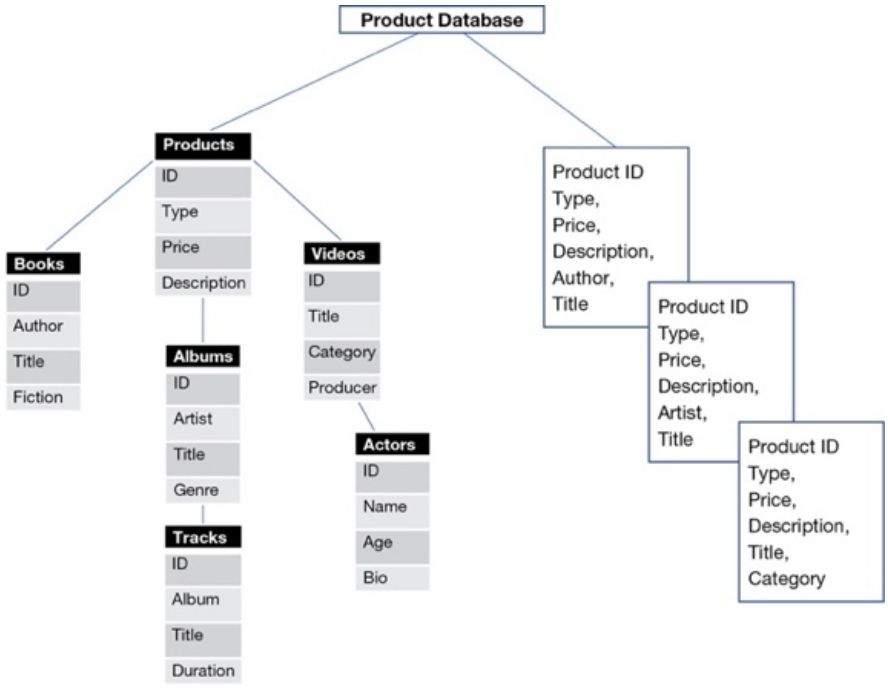

Figure 7 Relational database (on the left), non-relational DynamoDB database (on the right). DynamoDB allows data fetching with primary keys.

In Figure 7 above, the data structure between the relational database on the left and the non-relational database on the right is opened. In DynamoDB tables, data is stored in groups of attributes, called items, where each item can be retrieved by their primary key or other attribute that uniquely identifies them. [13.]

3.3.2 AWS Lambdas

The Lambda function allows / Lambda functions allow triggered calls to upload and run code against the database through an API Gateway. They can be written in multiple languages such as JavaScript or Python. The idea behind Lambdas is that a single function for a single purpose is needed at a specific time. As writing lambda functions, AWS allows full management of RAM/CPU processing time as Lambda code structure affects how much the lambda functions consume. [13.]

3.3.3 API Gateway

The API Gateway allows APIs to be published in locations hosted by AWS. An API can be partially or fully the source code of an application, offering HTTPS (Hypertext Transfer Protocol Secure) endpoints for mobile applications to communicate with the backend. The API Gateway, therefore, is a door to any service needed in AWS, and it can be either protected with authentication or not. Overall, API Gateway provides a great set of features of mobile application development backend:

- **Security** – It supports other AWS tools for authentication, such as AWS Cognito and IAM.
- **Traffic throttling** – Calls with same query can be responded from API cache, which means that it decreases the number of calls going all the way to the database.
- **Multiple version support** – Many API Versions can be hosted at the same time.
- **Metering** – Possibility to control access levels.
- **Access** – API checks authorization and actions accordingly. [13.]

3.3.4 Simple Notification Service

AWS SNS allows applications to deliver push-based notifications or messages to one or many destinations. Messages can be delivered by the created topic or to a single recipient if their number is known. Topics are channels which can have subscribers. If the message is sent to the topic, all the numbers which are subscribed will receive the message. SNS can be triggered for example in Lambda functions with a publish command. [14.]

4 Overall System

The SmartAccess system is designed to provide end users a mobile application which they can use to generate time sensitive access codes for a smart lock at the main door of their residence (see Figure 8).

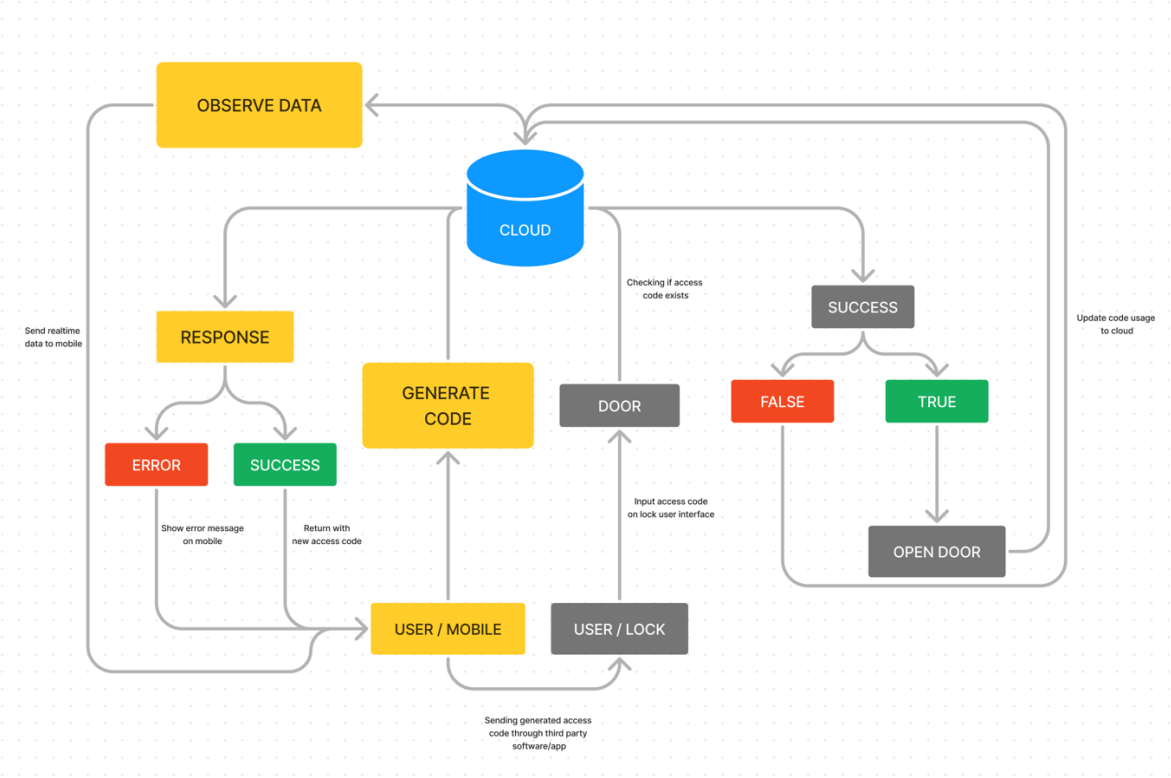

Figure 8 Overall system flow. The finalized system consists of two user interfaces and a mobile and door lock, which both communicate to the same backend.

Primary users of the system are divided into two groups as seen in Figure 8. Group 1 consists of people who live in the apartment buildings and want to generate a temporary access code with a mobile application, and Group 2 consists of those who wants to use the generated access code at the smart lock. To satisfy the needs of Group 1, the following parts were implemented:

- Mobile application interface and access code generation
- Backend configuration

The following part was excluded from the scope but may be implemented later:

- Smart lock at the door and communication to backend

4.1 Architecture

System architecture that serves Group 1 follows the monolithic structure, meaning. It is implemented within single codebase and all the components operate together as a whole. The mobile application is written with Swift, which is a programming language for the iOS platform. The backend service is made by using AWS tools, such as DynamoDB, API Gateway, AWS Lambdas and SNS. (see Figure 9).

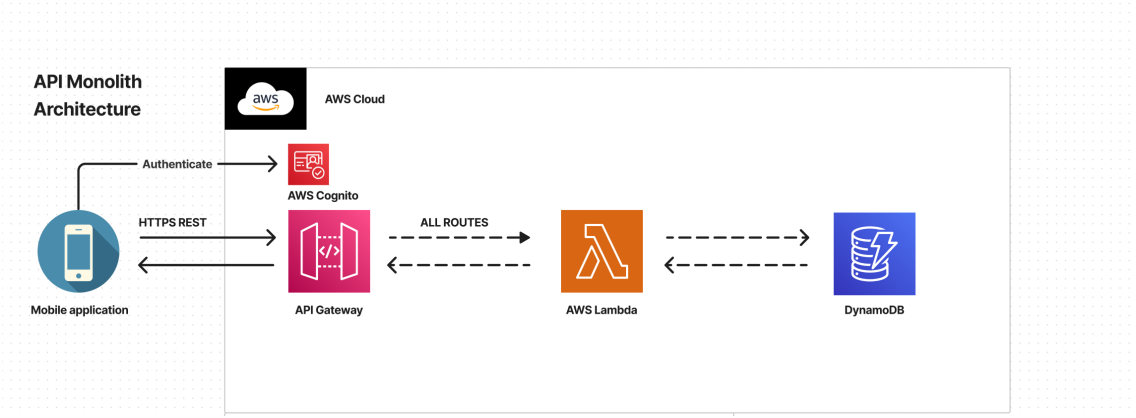

Figure 9 System architecture. Communication from the mobile application to the database goes through the monolithic system.

As seen in Figure 9, the mobile application communicates with the backend services through the API Gateway, which is the entry point for every request made by the user after the user is authenticated. The API Gateway contains configurations for each path, which can call their own respective lambda function for further actions. Lambda functions are written with Python to serve each call individually with interactions to the DynamoDB.

4.2 API Gateway

The API Gateway was used to publish an API for communication between the SmartAccess mobile application and backend services. AWS offers four different types of APIs, but SmartAccess was made with REST API. The API is configured with the combination of resources and methods. Each resource is an endpoint which is used to call the API from the mobile application and each resource

contains a method, such as GET, POST or PUT which defines the type of request that is made.

During the development process all modifications on the API side were made through the API Gateway. This included changes to the endpoints as well as deployment. (see Figure 10).

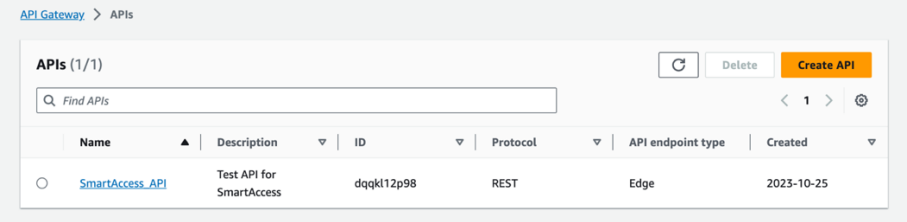

Figure 10 The SmartAccess API in the Amazon Web Services console. The API Gateway can be managed through the console by adding resources and methods for it. These modifications can then be published to production.

The base URL for the API in Figure 10 is **https://dqqkl12p98.execute-api.eu-north-1.amazonaws.com/smartaccess**, and it contains five endpoints which allows communication from the SmartAccess application. All five endpoints require an access token, which the user obtains once they are authorized.

- **/codes?residentId={residentId}**
- **/codes/attempt?codeId={codeId}**
- **/codes/code**
- **/codes/validity?codeId={codeId}**
- **/resident?residentId={residentId}**

4.3 Lambda Functions

Lambda functions are trigger events that connect the API Gateway with DynamoDB. The functions are linked to a method in the API Gateway and triggered when that method is called. Lambda functions are setup with permissions, which define the execution role that can access DynamoDB

resources. Every service that Lambda interacts with needs their own permission (see Figure 11). SmartAccess related permissions are described below:

- **AmazonDynamoDBFullAccess**
  Provides full access to Amazon DynamoDB via the AWS Management Console.
- **AmazonSNSFullAccess**
  Provides full access to Amazon SNS via the AWS Management Console.
- **AWSLambdaBasicExecutionRole**
  Provides write permissions to CloudWatch Logs.

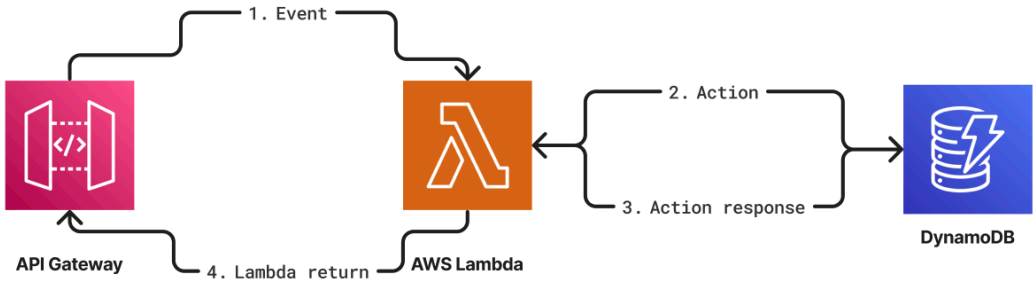

Figure 11 API Gateway communication to DynamoDB. An incoming request from the mobile application is received in the API Gateway, after which Lambda is triggered. Once Lambda received a response to its request from DynamoDB, it returns the response to the API Gateway.

Lambda configuration is made in a file called lambda_function.py, which is an autogenerated file made by AWS. As seen in Figure 11, Lambdas work by receiving an event from the API gateway after the endpoint is called. Inside the event, a body or query string parameters are passed which are needed for the communication with the database. Lambda handles and puts forward the needed content of the event in an action to DynamoDB. The action can be a request for fetching an item with an ID or updating a value. Once the action completes, DynamoDB returns a response to Lambda, which handles it again accordingly and returns the final response to API Gateway. This response is then returned to the mobile application where data is processed and showed to the user.

In SmartAccess, the application communicates with the database through five different Lambda functions, which are listed below:

- **getAccessCodes**
  Fetches all access codes by the user ID. It also maps code validity details for all access types in the return response. This is used to show code generation history to the user.

- **getCodeValidity**
  Fetches single code validity by code ID.

- **getResident**
  Fetches currently logged in resident and community details and maps them together in the response.

- **putCodeValidity**
  Updates validity details of a single code by code ID when code is regenerated. This changes the date until the code is valid, so it can be observed in the mobile and showed accordingly in the code generation history.

- **postCode**
  Creates new code in the DynamoDB database when the user generates new code. The function also handles updating the initial code value to VALIDITY and ATTEMPT tables and sends the generated code to the visitor as a text message.

4.4 Database

Non-relational AWS DynamoDB is used as the mobile application backend, due to its capability to provide fast and predictable performance for read and write operations as well as compatibility with other AWS services.

The purpose of the database is to store data generated by the mobile application as well as to provide real-time data back. This data can be related to the community, user, code validity or history, which is used to keep track of the temporary door codes a single user generates under a specific community.

The SmartAccess DynamoDB database consists of five tables which all store different type of data. Every table contains a unique primary key which is used to query data from the tables, and some have an optional sort key which can be used in combination with the primary key to sort data (see Figure 12).

PRIMARY KEY		Attributes						
Partition key: PK	Sort key: SK							
		residentId	username	firstName	lastName	email	phone	community
RESIDENT#residentId		int	string	string	string	string	string	int or null
		communityId	name	adminCode	password	addressLine1	postalCode	city
COMMUNITY#communityId		int	string	int	string	string	string	string
		codeId	residentId	code	label	receiverPhone	createdAt	accessType
CODE#residentId	CODE#createdAt	int	string	string	string	string	timestamp	number
		codeId	createdAt	result				
ATTEMPT#codeId		int	timestamp	boolean				
VALIDITY#codeId		codeId	createdAt					
		int	timestamp					

Figure 12 Non-relational database schema in DynamoDB. The Database contains five tables for storing access code, user, and community related data.

The database schema is built to divide data into small but logical pieces that can be individually modified as necessary, as seen in Figure 12. These tables are listed below:

- **RESIDENT**

  Stores resident details. Residents are those who live in the apartment building. The table is accessed with the resident ID.
- **COMMUNITY**

  Stores community details. Community is the building where residents live. The table is accessed with the community ID.
- **CODE**

  Stores code details. Code is created for a single community by a resident. The table is accessed both with the resident ID and creation time.

- **ATTEMPT**

  Stores the latest attempt status for all code. The attempt is the time when the visitor tried to use the code for the community door. Attempt details are used to check if onetime code is used (when status is 200) or not (when status is 500). The table is accessed with the code id.

- **VALIDITY**

  Stores validity details for all codes. Validity is used to check when the code expires and the validity is modified when time sensitive code is regenerated. The table is accessed with the code id.

5 Mobile Application

The SmartAccess mobile application is the UI for Group 1 users. It is available for use for the residents of the apartment building holding an iOS device. Its primary purpose is to allow users to create time sensitive access codes for the buildings, so called communities where they live. On top of this, the user can check the code generation history as well as see the community membership details (see Figure 13). The purpose of the application is to prevent users from sharing the official access code of their building to temporary visitors, bringing safety and flexibility to the current access management solutions.

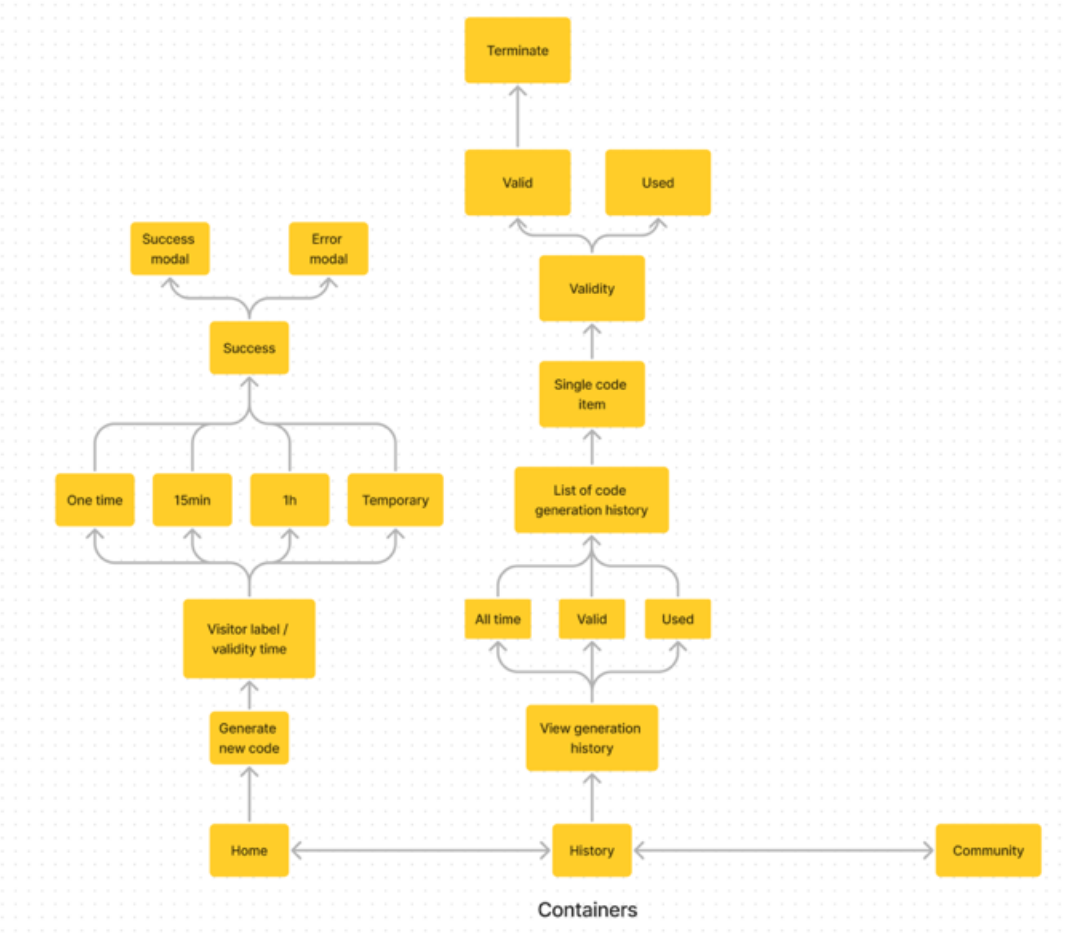

Figure 13 Mobile application flow. The mobile application consists of three flows that serve a different purpose.

The application consists of three main views as seen in Figure 13. These views are Home, History and Community. The home view allows users to create new codes and see information relevant to them, such as the official access code for the community or the community's name. In the history view, the user can view

the whole access code generation history and reactivate already expired access codes. The community view allows the user to see their community details, but not edit them. On top of this, the application contains some other limitations and rules to users, such as the following:

- Users are only able to create one code at a time. If another code is already active, generation and regeneration buttons are deactivated, and the user receives a prompt that suggests the user to share a code that is already active.
- The user can only create access codes to the communities where they live, but they can only be registered to one community at a time.
- For now, the user is not able to delete the created access codes.

5.1 User Interface Design

Through the design process, the design goal was to provide a smooth and responsive UI for the target group as well as to give an opportunity to create two types of access codes in the building where they live.

The design process started by planning the scope of the mobile application. The scope was defined using the user survey analysis and focusing on the most crucial part, which was the code generation as well as all the related actions with it in the mobile application. During the initial planning phase, particular emphasis was also placed on defining the strategies on how to ensure the smooth generation of the access codes and what type of components this requires. This meant that some common features in typical mobile applications, such as signup, login and community management were not considered in the design process and implementation.

The second phase of the design process involved sketching wireframes for the application, where the aim was also to establish the visual identity.

Wireframing resulted in the first skeleton of what the application looks like, by planning the components to be used and their alignment in the views. (see Figure 14).

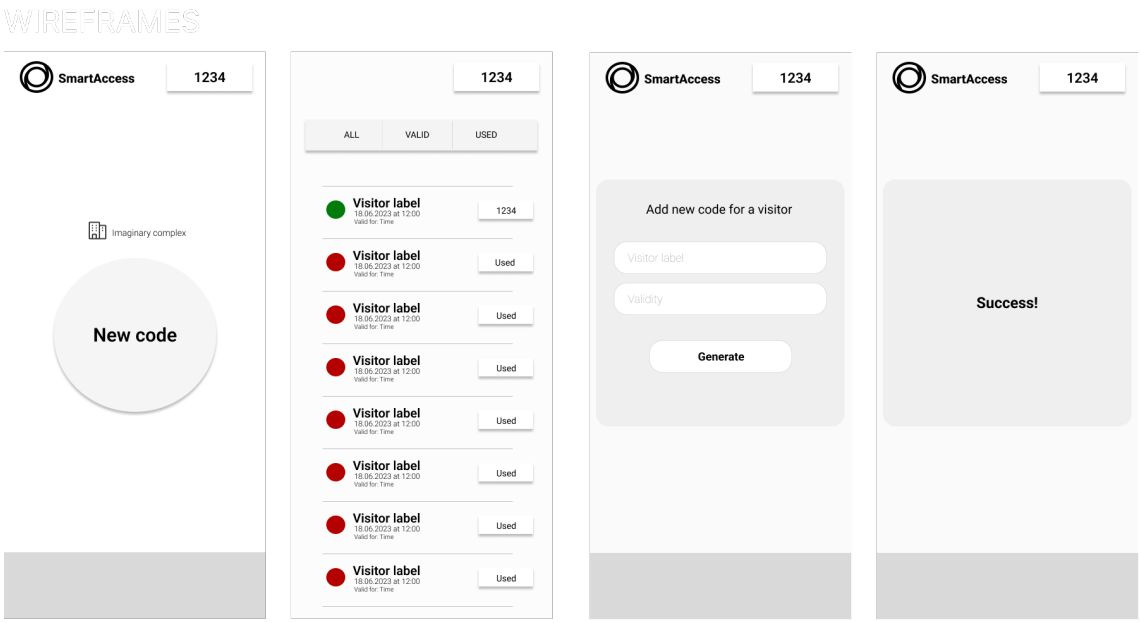

Figure 14 Design wireframes. The first step of designing the application user interface was to make a rough plan on how the application and its views would look like.

Wireframing the application was a crucial first step, as during this process some parts could be removed or changed. This sort of modelling usually includes playing with imaginary data, as seen in Figure 14.

The third phase involved designing the software architecture, wherein the pre-existing backend service underwent a thorough check in the eyes of the mobile application. Any anomalies discovered were addressed for enhancement. In addition, the mobile application made its first call to the backend, which allowed testing the API endpoints with the mobile application.

In the final step, which was initiated before the development of the mobile application, the initial wireframes underwent modifications, resulting in the creation of the final design.

The final design included the definition of the whole design system, such as selection of typography, component styling and establishment of a cohesive colour scheme (see Figure 15).

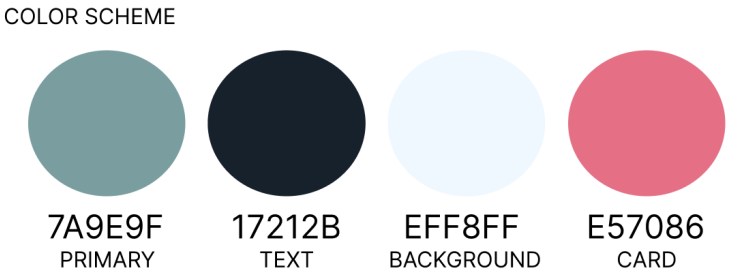

Figure 15 Application colour scheme. The colour scheme defined what colours would be used in specific areas of the application, such as buttons, background, and text.

The purpose of the colour scheme was to obtain a consistent look for the mobile application, as the background, buttons and highlights were following the same pattern, as seen in Figure 15. The final design was supposed to give space to the components and make them easy to drag and click. The goal was to have the application simple for the purpose and appealing for the eye (see Figure 16).

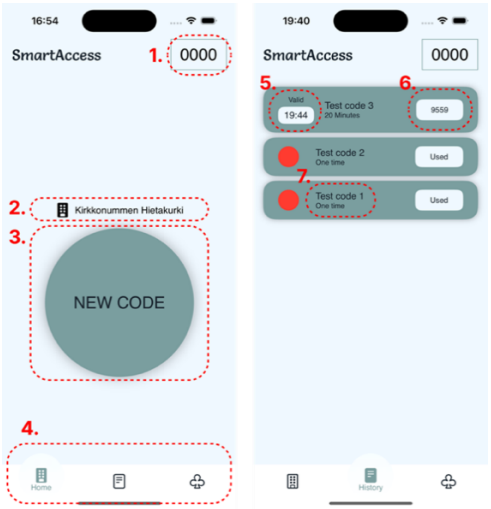

Figure 16 Application final design. The design got its final form after the colour scheme was combined with design wireframes.

The application features that allow user interaction or that are otherwise important from the user perspective are the following, as seen in Figure 16:

1. Admin code for community – The user should not share this code with visitors. The code is replaced by generated codes.
2. Name of the community the user is part of.
3. Code generation button, which the user can press to create a new access code.
4. Bottom navigation to change the view.
5. Validation countdown for codes that are valid for 20 minutes. If a code is expired, it is replaced with a red ball, and if a onetime code is valid, it is replaced by a green ball.
6. Generated access code is shown if a code is valid, otherwise code is replaced with word “used”.
7. Name of the code that the user chose and the validity type.

5.2 Security

All the codes generated through the SmartAccess application are forwarded to the visitors through SNS as a transactional message (see Figure 17).

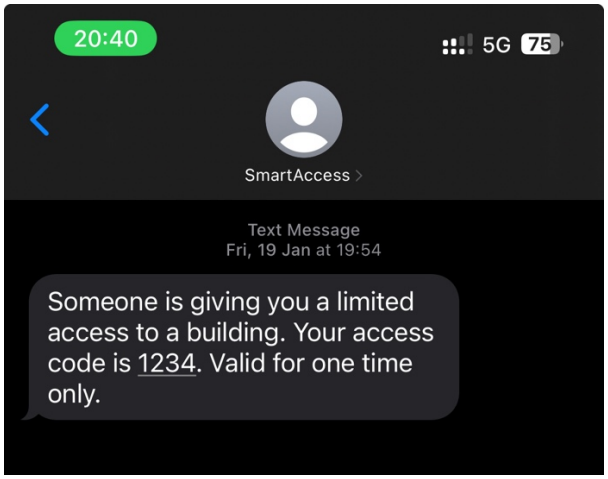

Figure 17 Visitor receiving access code via test message. Access codes are sent to the number provided by the resident during the generation process. Access codes are sent through the Amazon SNS service.

The message is sent once a new code, validity and an attempt is added to the database in the Lambda function. Transactional messages are considered critical, and they can contain sensitive information such as one-time passcodes

which need to be delivered reliably. Messages are sent to the number the resident provides while generating the new code.

5.3 Testing

The SmartAccess testing phase was included in the following tests in the following order:

- **User testing**
  User testing was carried out while validating the UI. In the user testing, two people were made to try out the prototype of the design. Focus was placed on two things:
  - **Notes**
    Notes included, for instance, facial expressions and emotions the test users were expressing while navigating through the application.
  - **Feedback**
    Feedback was thoughts that were expressed verbally.

- **Unit testing**
  Unit tests were written for functions all over the application while reaching a test coverage of 29 %.

- **Performance testing**
  Performance testing included 2 runs made with Xcode Instruments where samples were taken in each part. These runs were related to the code generation and viewing the history. Some latency was found from fetching the code history due to how Lambda was made to call the different database tables. The results were improved by 5 seconds by refactoring the code in Lambda functions.

On top of this, error handling was added in each API call and Lambda functions to detect problems in the code during the development phase.

5.4 Future Vision

In the future if SmartAccess is continued, it will be set up with AWS Cognito authentication and identification.

On top of this, the application can be improved by adding community management on the community view. Once the user can manage communities themselves, they will be able to join and exit communities independently (see Figure 18).

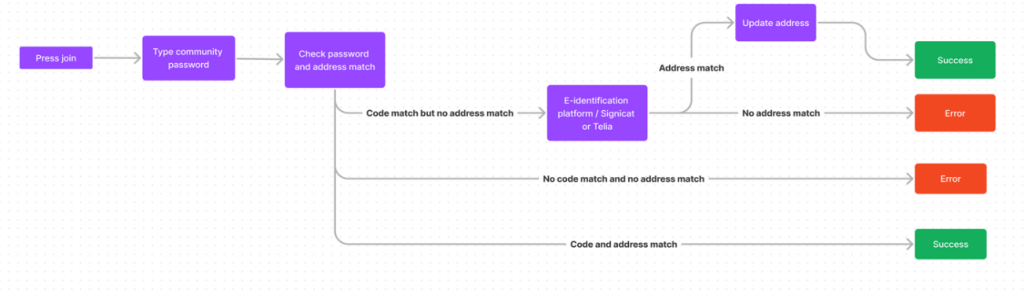

Figure 18 Flow for joining the community. The user's real address is checked from an e-identification platform, such as Signicat, Telia or Stripe.

Communities can be joined with the combination of a right password and an address match to confirm that the user lives in the specific building. The user receives the community password for the SmartAccess application from the responsible person of the apartment building, after which they can change their community details in the application. Users can navigate through three paths when managing their community details, as seen in Figure 18. The user's current address is always retrieved from the Finnish population system, which shows the official address of private people. This address is always checked if necessary while the user tries to join a new community. If the user's current address and community password are both right, the user can immediately create access codes for the new community. However, if the user's official address is not updated in the Finnish population system, the address is rechecked and the user is required to make a new strong authentication, after which the address is updated. If the user's address is still not matching with the address in the community, the user needs to wait for the address to change in the Finnish population system to try again.

6 Conclusion

Developing a fully functional application always takes somewhere from four to six months. This project was carried out in around five months, including research and implementation. SmartAccess is an example of how security in apartment buildings can be increased, by giving more power to the residents of a building.

The primary objective of this project was to enhance existing access control methods by suggesting and implementing a functional prototype of how the residents could manage access codes themselves. The exploration of login and community management was deemed beyond the project's scope as the primary focus remained on perfecting the core functionality of the application, which was seamless code generation and management. The idea grew from personal experiences with the current state of access control in apartment buildings and a comprehensive user survey involving over 70 participants revealing concerns about the inadequacy of current control mechanisms.

As a result, the identification of existing imperfections in access management solutions allowed to evaluate the level of security they provide to individuals. The research also provided concrete evidence that access codes in apartment buildings are often left unchanged. Armed with these insights, a working prototype of a mobile application user interface was implemented, which not only enhances safety for communities and residents but also allows residents to generate time sensitive access codes and to deliver them to their visitors by using transactional messages.

Moving forward, this project lays the groundwork for the continued evolution of Smart Access Control System. On the mobile side, the implementation of login and authorization flows will ensure that residents truly live in specific communities and that the addition of community management feature enables the residents to handle address details seamlessly. Simultaneously, on the hardware front, smart locks could be connected to the mobile backend, allowing visitors to use the generated access codes to open doors.

References

1 United Nations Department of Economic and Social Affairs, Population Division (2022). World Population Prospects 2022: Summary of Results. UN DESA/POP/2022/TR/NO. 3, p. 5.

2 Population Division. United Nations. Online. <https://population.un.org/wpp/Graphs/Probabilistic/POP/TOT/246>. Accessed 14 November 2023.

3 Gate codes – Access Control in Gated Communities. 2023. Online. Nimbio. <https://nimbio.com/gate-codes-access-control-in-gated-communities/>. Updated 4 June 2023. Accessed 14 November 2023.

4 Guide: Access Control – An Overview of Access Control Products. 2022. Online. Satori Cyber Ltd. <https://satoricyber.com/access-control/an-overview-of-access-control-products/>. Updated 9 June 2022. Accessed 14 November 2023.

5 Mobile Access Control Guide. Online. Kisi Inc. <https://www.getkisi.com/guides/mobile-access-control-guide>. Accessed 14 November 2023.

6 Bluetooth Technology: How It Works in Access Control. Kisi Inc. <https://www.getkisi.com/guides/bluetooth-access-control>. Accessed 14 November 2023.

7 A Guide to RFID and NFC Access Control Systems. Kisi Inc. <https://www.getkisi.com/guides/rfid-access-control>. Accessed 14 November 2023.

8 Boonkrong, Sirapat. 2021. Authentication and Access Control. E-book. Apress. <https://learning.oreilly.com/library/view/authentication-and-access/9781484265703/>. Accessed 5 December 2023.

9 Windley, Phillip J. 2023. Learning Digital Identity. E-book. O'Reilly Media Inc. <https://learning.oreilly.com/library/view/learning-digital-identity/9781098117689/>. Accessed 5 December 2023.

10 Jan de Vries, Henk & Stjernlof, Lovisa. 2020. Okta Administration: Up and Running. E-book. Packt Publishing. <https://learning.oreilly.com/library/view/okta-administration-up/9781800566644/>. Accessed 5 December 2023.

11 Hauser, Dr. Dominic. 2022. Test-Driven-iOS Development with Swift – Fourth Edition. E-book. Packt Publishing. <https://learning.oreilly.com/library/view/test-driven-ios-development/9781803232485/>. Accessed 5 December 2023.

12 Hyett, Alex. 2023. Unit Testing vs Integration Testing. Online Material. Alex Hyett. <https://www.alexhyett.com/unit-testing-vs-integration-testing/>. Accessed 14 November 2023.

13 Wilkins, Mark. 2019. Learning Amazon Web Services (AWS): A Hands-On-Guide to the Fundamentals of AWS Cloud. E-book. Addison-Wesley Professional. < https://learning.oreilly.com/library/view/learning-amazon-web/9780135301104/>. Accessed 5 December 2023.

14 Prasah Buddha, Jyothi & Beesetty, Reshma. 2019. The Definite Guide to AWS Application Integration: With Amazon SQS, SNS, SWF and Step Functions. E-book. Apress. <https://learning.oreilly.com/library/view/the-definitive-guide/9781484254011/>. Accessed 5 December 2023.