Unapređenje sustava za kontrolu pristupa zasnovanog na tehnologiji Interneta stvari

Janić, Mihael

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 332

IMPROVEMENTS OF INTERNET OF THINGS-BASED ACCESS CONTROL SYSTEM

Mihael Janić

Zagreb, June 2024

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 332

IMPROVEMENTS OF INTERNET OF THINGS-BASED ACCESS CONTROL SYSTEM

Mihael Janić

Zagreb, June 2024

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Zagreb, 04 March 2024

MASTER THESIS ASSIGNMENT No. 332

Student:	**Mihael Janić (0036510950)**
Study:	Computing
Profile:	Software Engineering and Information Systems
Mentor:	prof. Josip Knezović

Title: **Improvements of Internet of Things-based access control system**

Description:

Improve the Houseleek system developed at the Faculty of Electrical Engineering and Computing as a prototype utilizing web technology and the Internet of Things for access control in offices, classrooms, and laboratories. Improve the administration and configuration interface for embedded system modules to enable remote configuration of access control parameters such as free access time (allowing access to specific rooms during office hours upon motion detection in front of the module), finer granularity of roles and authorization rights per organizational units (departments, laboratories, research groups), and the required method (level) of user authentication during periods when free access is not permitted. Upgrade the software support of embedded modules to enable dynamic refreshment and addition of new functionalities (OTA, Over The Air). Enhance and adapt the existing client application for Android-based mobile devices.

Submission date: 28 June 2024

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 332

Pristupnik:	**Mihael Janić (0036510950)**
Studij:	Računarstvo
Profil:	Programsko inženjerstvo i informacijski sustavi
Mentor:	prof. dr. sc. Josip Knezović

Zadatak: **Unapređenje sustava za kontrolu pristupa zasnovanog na tehnologiji Interneta stvari**

Opis zadatka:

Unaprijediti sustav Houseleek koji je razvijen u Fakultetu elektrotehnike i računarstva kao prototip korištenja web-tehnologije i Interneta stvari za kontrolu pristupa u urede, učionice i laboratorije. Unaprijediti sučelje za administraciju i konfiguraciju sustava ugradbenih modula koji će omogućiti udaljeno konfiguriranje parametara kontrole pristupa kao što su vrijeme slobodnog pristupa (dozvoljen pristup pojedinim prostorijama u uredovnom vremenu pri detekciji pokreta ispred modula), veću granulaciju uloga i prava autorizacije po ustrojbenim jedinicama (zavodi, laboratoriji, istraživačke grupe) te zahtijevani način (razinu) autentifikacije korisnika u vremenskim periodima kada slobodan pristup nije dozvoljen. Unaprijediti programsku podršku ugradbenih modula s ciljem dinamičkog osvježavanja i dodavanja nove funkcionalnosti (OTA, od eng. Over The Air). Unaprijediti i prilagoditi postojeću klijentsku aplikaciju za mobilne uređaje zasnovane na operacijskom sustavu Android.

Rok za predaju rada: 28. lipnja 2024.

I would like to thank my family and relatives who gave me enormous support during my studies, my colleagues who always helped me selflessly, and my mentor Prof. Josip Knezović, Ph.D., for his support during the preparation of my thesis.

Contents

List of Figures

1. Introduction

The Internet of Things (IoT) is a global infrastructure for the information society enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving, interoperable information and communication technologies [13].

In recent years, the integration of Internet of Things (IoT) technologies into access control systems has significantly changed the way physical security is managed. This integration has enabled the development of more flexible, scalable, and efficient solutions tailored to the needs of modern organizations. This thesis aims to enhance the capabilities of the Houseleek system, which was originally developed at the Faculty of Electrical Engineering and Computing, by transforming it into a comprehensive IoT-based access control framework. The goal is to utilize web technologies and advancements in IoT to improve access management in offices, classrooms, and laboratories.

Currently, the Houseleek system functions as a prototype for access control, utilizing embedded system modules for local management. However, to meet current demands for enhanced security and operational flexibility, several improvements are necessary. These enhancements include increasing system stability, conducting extensive testing, improving documentation, and implementing minor fixes and adjustments.

Through these advancements, the Houseleek system aims to set a new standard in IoT-based access control, providing a scalable and adaptable solution that addresses the diverse security needs of both academic and professional environments. This thesis explores the technical implementations, challenges encountered, and the potential impact of these enhancements on improving the current state of access management systems.

1.1. Motivation

This thesis is motivated by the evolving requirements of access control systems and the increasing need for sophisticated yet flexible solutions in academic and professional settings. Traditional access control mechanisms, while effective, often lack the agility and scalability required to address modern security challenges. The integration of IoT

technologies presents an opportunity to revolutionize access management by leveraging interconnected devices and advanced software capabilities.

The Houseleek system provides a foundational basis for further refinement. However, the existing system revealed several areas requiring improvement, including stability, testing, documentation, and minor functional enhancements. These shortcomings necessitated a focused effort to stabilize the system, ensure robust functionality, and prepare it for future enhancements leveraging web technology and IoT principles.

The decision to enhance the Houseleek system with web technology and IoT principles was motivated by practical considerations rather than initial capabilities. The primary objectives include:

- **Foundational Stability and Reliability:** Initially, the focus was on stabilizing the system by addressing existing issues and establishing a robust foundation. This involved rectifying initial flaws to ensure the Houseleek system operates reliably in controlled environments.
- **Preparation for Incremental Improvements:** Future phases include incremental functional enhancements such as enhancing remote configurability, refining role granularity to better manage user access, and enabling basic dynamic updates. These improvements aim to gradually enhance the system's capabilities and adaptability.
- **Exploration of IoT Principles:** Integrating fundamental IoT-based access control principles aims to explore real-time monitoring capabilities and basic adaptive access permissions. This exploration seeks to lay the groundwork for potential future developments in security and operational efficiency.
- **Potential for Adaptability:** Looking forward, the adoption of IoT technologies in the Houseleek system presents opportunities for scalability and adaptation to future growth and organizational changes. This scalability aims to align the system with emerging security demands and technological advancements.

2. Project Houseleek

Project Houseleek is an access control system designed to manage entry into rooms within organizational environments, leveraging a hierarchical structure of user roles for precise authorization. The name "Houseleek" originates from the Croatian word *čuvarkuća*, reflecting the system's primary purpose. The logo of the system is shown in Figure 2.1.

The system scheme is shown in Figure 2.2 and consists of several integral components designed to enhance security and operational efficiency:

Centralized Server

At its core, Houseleek operates through a centralized server infrastructure. This server manages authentication, authorization, and access control policies across all connected devices and modules, ensuring a cohesive and synchronized approach to security management.

Houseleek Remote Module

The system includes specialized modules deployed at entry points. These modules facilitate real-time monitoring of access attempts, enforce access rules based on configured parameters (such as office hours and motion detection), and communicate securely with the centralized server to verify credentials and grant access accordingly.

Web Application

Houseleek features a web-based administration interface that enables remote manage-

Figure 2.1: Houseleek project logo

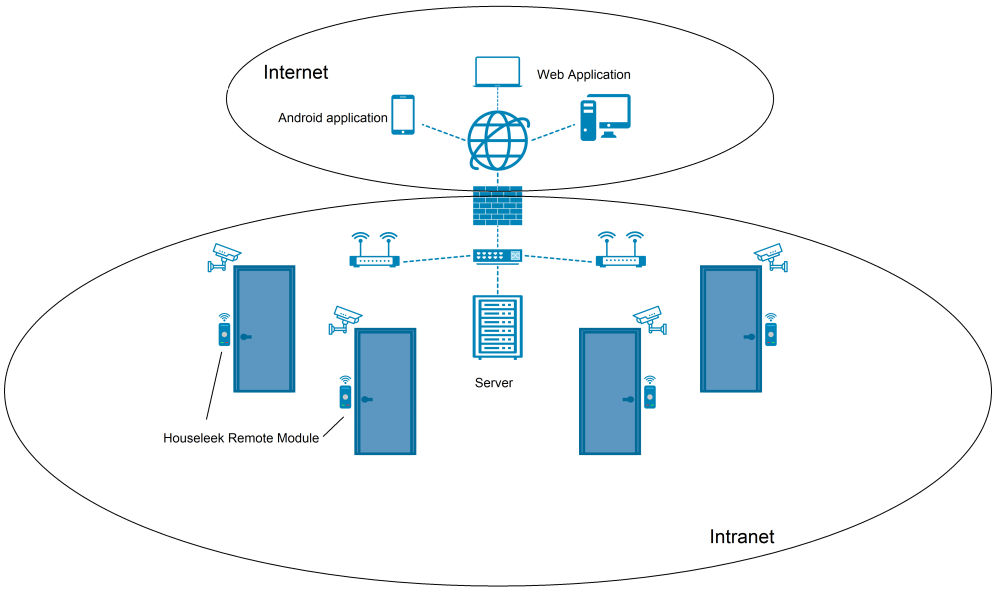

Figure 2.2: Houseleek system scheme [14]

ment and configuration of access control parameters. Administrators can define access policies, manage user roles, monitor access logs, and perform system diagnostics from any web-enabled device, enhancing administrative flexibility and efficiency.

Android Application

Houseleek also provides an Android-based mobile application specifically designed for user verification and access control. Users can authenticate themselves through the application or via NFC on their smartphones if supported. This application enhances the user experience by enabling convenient, on-the-go access management, ensuring seamless integration with the Houseleek system's functionalities and maintaining operational continuity.

2.1. Software

The software component of Project Houseleek encompasses a robust infrastructure designed to ensure seamless integration, efficient management, and reliable operation of its access control system. Built upon modern technologies and frameworks, the software architecture incorporates several key elements.

- **Laravel**: Powers the web-based administration interface for centralized access control management.
- **Laradock**: Docker-based PHP development environment simplifying setup and

configuration.

- **Android Application**: Developed in Android Studio for mobile access control.
- **Git on GitLab**: Version control system for collaborative development and code integrity.

2.1.1. Laravel

Laravel is a PHP-based web framework renowned for its elegant syntax and comprehensive feature set, designed to streamline web application development. It integrates various characteristics from technologies such as ASP.NET MVC, CodeIgniter, Ruby on Rails, and others, making it a versatile choice for developers. Developed by Taylor Otwell in June 2011, Laravel follows the model-view-controller (MVC) architectural pattern [26].

Key features of Laravel include:

- **Routing and Controllers:** Laravel simplifies route definition and controller handling, enhancing application structure and organization.
- **ORM (Object Relational Mapper):** Laravel's Eloquent ORM facilitates database interaction through intuitive active record implementation.
- **Blade Templating Engine:** Laravel provides Blade, a powerful templating engine enabling efficient PHP-based views with loops, conditional statements, and layouts.
- **Authentication and Authorization:** Built-in authentication features include user management, login, registration, password reset, and email verification.
- **Artisan CLI:** Laravel's command-line interface, Artisan, offers tools for automating repetitive tasks, database migrations, and application scaffolding.

Laravel emphasizes security and developer productivity, allowing developers to build robust web applications efficiently. The latest stable release, Laravel 11.x, was launched in March 2024 and is available for download on GitHub [16].

2.1.2. Docker

Docker is an open source engine that quickly wraps up any application and all its peculiar dependencies in a lightweight, portable, self-sufficient container that can run virtually anywhere on anybody's infrastructure [3]. Docker container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container

image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings [**?**]. To build the image, you'll need to use a Dockerfile. A Dockerfile is simply a text-based file with no file extension that contains a script of instructions. Docker uses this script to build a container image [10].

2.1.3. Laradock

Laradock serves as the foundational PHP development environment for Project Houseleek, leveraging Docker to provide a comprehensive suite of pre-configured services. Designed with flexibility and ease of use in mind, Laradock supports a wide array of functionalities essential for PHP development [15]:

- **Versatile PHP Version Management:** Laradock allows effortless switching between PHP versions (including PHP 8.1, 8.0, 7.4, 7.3, 7.2, 7.1, and 5.6), enabling developers to match specific project requirements seamlessly.
- **Database Flexibility:** Developers can choose from various database engines such as MySQL, PostgreSQL, MariaDB, and more, ensuring compatibility with diverse application needs.
- **Modular Architecture:** Each service runs independently in its Docker container, facilitating easy customization and scalability without impacting other components. Key containers include PHP-FPM, NGINX, MySQL, and Redis, among others.
- **Easy Configuration and Customization:** Laradock simplifies the setup process with Docker-compose files and environment variables, allowing developers to tailor containers to project specifications effortlessly.
- **Comprehensive Toolset:** The workspace container within Laradock includes essential tools like PHP CLI, Composer, Git, Node.js, and various development utilities, ensuring a productive environment for PHP application development.
- **Integration with Laravel:** Built on the Laravel PHP framework, Laradock seamlessly integrates with Laravel applications, enhancing development efficiency and enabling rapid iteration.

By adopting Laradock, Project Houseleek benefits from a standardized development environment that supports agile development practices, facilitates version control via GitLab, and enhances deployment flexibility. Laradock's robust architecture and extensive feature set empower developers to build, test, and deploy PHP applications efficiently within the Docker ecosystem.

2.1.4. Android

Android is a versatile mobile operating system based on a modified Linux kernel and other open-source software, primarily designed for touchscreen devices such as smartphones and tablets. Developed by the Open Handset Alliance, with Google as the main contributor, Android was first unveiled in 2007, and the HTC Dream became the first commercial Android device in 2008.

The Android Open Source Project (AOSP) is the foundational, free, and open-source version of Android, licensed under the Apache License. Most commercially available devices run a proprietary version of Android developed by Google, which includes Google Mobile Services (GMS) with essential apps like Google Chrome, Google Play, and Google Play Services. Firebase Cloud Messaging is used for push notifications.

Over 70 percent of AOSP-based smartphones are integrated into Google's ecosystem, with some vendors adding custom interfaces, such as Samsung's One UI and HTC Sense. Alternative ecosystems and AOSP forks include Amazon's Fire OS, Oppo's ColorOS, Vivo's OriginOS, and Honor's MagicUI. Custom ROMs like LineageOS offer further customization options [1].

Android's source code has also been adapted for various other devices, including game consoles, digital cameras, and PCs. Notable derivatives include Android TV and Wear OS, both developed by Google. Apps for Android are typically distributed through proprietary stores like Google Play Store, Amazon Appstore, and Samsung Galaxy Store, as well as open-source platforms like F-Droid.

Since 2011, Android has been the best-selling OS for smartphones globally and has dominated the tablet market since 2013. As of May 2021, Android boasts over three billion monthly active users, the largest installed base of any operating system worldwide. The Google Play Store hosts over three million apps as of January 2021. The latest version, Android 14, released on October 4, 2023, includes updates tailored for foldable phones, tablets, and Chromebooks [1].

2.1.5. Git on GitLab

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency [7]. GitLab, a web-based DevOps lifecycle tool, provides a Git repository manager offering wiki, issue-tracking, and CI/CD pipeline features using an open-source license. Project Houseleek employs Git and GitLab for version control to ensure effective collaboration, seamless integration, and robust management of the codebase [4].

Version Control:
Using Git, developers can manage changes to the source code over time. GitLab's repository manager provides a web-based interface for Git repositories, making it easier to visualize the development process, track progress, and collaborate on code.

Collaboration:
GitLab facilitates collaborative development by allowing multiple developers to work on the same project simultaneously. Features like merge requests, code reviews, and branch management streamline the collaboration process and ensure high-quality code.

CI/CD Integration:
GitLab's CI/CD pipelines automate the testing and deployment of code, enhancing the development workflow. This integration ensures that every change is automatically tested and deployed, reducing the risk of bugs and accelerating the release process.

Issue Tracking and Wiki:
GitLab includes an issue-tracking system and a wiki, providing a centralized platform for project documentation and task management. This integration supports project planning, bug tracking, and knowledge sharing, contributing to the overall efficiency of the development process.

By leveraging Git and GitLab, Project Houseleek adheres to best practices in software development, promoting transparency, collaboration, and iterative improvement. The robust version control system and comprehensive toolset provided by GitLab support the project's agile development methodology, ensuring continuous delivery and high-quality software.

2.2. Hardware

The hardware foundation of the Houseleek system is centered around a System-on-a-Chip (SoC) microcontroller with integrated Wi-Fi capabilities. This specialized hardware configuration is tailored for embedded applications, providing a compact and efficient platform to manage access control in various environments. Key components of the hardware include:

System-on-a-Chip (SoC) Microcontroller
The heart of the Houseleek system, the SoC microcontroller integrates essential components such as a central processing unit (CPU), memory interfaces, input/output devices, and Wi-Fi connectivity onto a single chip. This integration optimizes space,

power consumption, and performance, making it ideal for embedded applications where compactness and efficiency are paramount.

Integrated Wi-Fi
The SoC microcontroller includes built-in Wi-Fi functionality, enabling seamless connectivity to the network infrastructure. This feature allows the Houseleek system to communicate with the centralized server and other networked devices without the need for additional external modules, simplifying deployment and enhancing flexibility in placement.

Memory and Storage Interfaces
The SoC microcontroller incorporates programmable memory such as NOR flash, OTP ROM, or ferroelectric RAM for storing firmware, configuration settings, and access logs. This onboard memory facilitates rapid data access and retrieval, crucial for real-time access control operations.

Peripherals and Interfaces
In addition to Wi-Fi, the microcontroller supports various input/output peripherals essential for its function within the access control system. These may include GPIO (General Purpose Input/Output) pins for sensor interfaces, UART (Universal Asynchronous Receiver-Transmitter) for communication with external devices, and SPI (Serial Peripheral Interface) for interfacing with additional peripherals.

Power Efficiency and Reliability
Designed for embedded applications, the SoC microcontroller prioritizes power efficiency while ensuring reliable operation in diverse environmental conditions. This reliability is essential for maintaining continuous access control operations without interruptions or performance degradation.

2.2.1. ESP32

Within the hardware architecture of the Houseleek system, the ESP32 microcontroller stands as a cornerstone for efficient and robust IoT-based access control. Developed by Espressif Systems, the ESP32 combines powerful processing capabilities with integrated Wi-Fi and Bluetooth functionalities, making it an ideal choice for embedded applications requiring connectivity and computational efficiency.

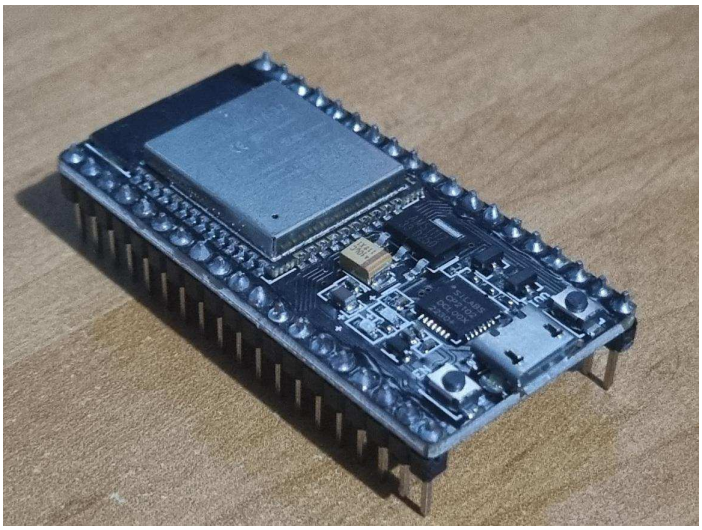

Figure 2.3: ESP32 development board

Key Features of ESP32

- **Microprocessor:** Equipped with either a dual-core or single-core Xtensa LX6 microprocessor running at frequencies up to 240 MHz, capable of delivering up to 600 DMIPS (Dhrystone Million Instructions Per Second). This architecture ensures swift execution of access control algorithms and management tasks.
- **Memory and Storage:** Includes 520 KiB of RAM and 448 KiB of ROM, providing ample space for firmware storage, configuration settings, and temporary data storage essential for operational tasks.
- **Wireless Connectivity:** Supports 802.11 b/g/n Wi-Fi standards for seamless network integration and dual-mode Bluetooth (v4.2 BR/EDR and BLE), facilitating communication with mobile devices and other Bluetooth-enabled peripherals.
- **Peripheral Interfaces:** Features 34 programmable GPIOs for sensor inputs and device control, a 12-bit SAR ADC supporting up to 18 channels for analog signal acquisition, multiple UART, SPI, and I²C interfaces for peripheral communication, and dedicated interfaces for SDIO, SPI, and Ethernet MAC with DMA support.
- **Security Features:** Implements robust security measures including secure boot, flash encryption, and hardware-accelerated cryptographic algorithms (AES, SHA-2, RSA, ECC). Supports standard Wi-Fi security protocols (WPA, WPA2, WPA3) ensuring secure data transmission.
- **Power Management:** Optimized for low power consumption with features

like a low-dropout regulator, RTC power domain, and ultra-low deep sleep current of 5 uA. Supports various wake-up sources including GPIO interrupts and capacitive touch sensor interrupts, ensuring energy-efficient operation.

2.2.2. NFC

The NFC module used for the Houseleek system is the PN532. The PN532 is a highly integrated transceiver module for contactless communication at 13.56 MHz, ideal for the Houseleek project. It supports various operating modes, ensuring versatility:

Supported Modes:

- ISO/IEC 14443A/MIFARE Reader/Writer
- FeliCa Reader/Writer
- ISO/IEC 14443B Reader/Writer
- ISO/IEC 14443A/MIFARE Card Emulation (MIFARE Classic 1K or 4K)
- FeliCa Card Emulation
- ISO/IEC 18092, ECMA 340 Peer-to-Peer

Key Features

- **Signal Handling and Speed:**
  - ISO/IEC 14443A/MIFARE Compatibility: The PN532 can demodulate and decode signals from ISO/IEC 14443A/MIFARE compatible cards and transponders, handling complete ISO/IEC 14443A framing and error detection (Parity and CRC).
  - High Transfer Speeds: Supports contactless communication with MIFARE and FeliCa at speeds up to 424 kbit/s in both directions.
- **Card Emulation and Communication:**
  - Card Emulation Mode: The PN532 can respond to Reader/Writer commands according to the ISO/IEC 14443A/MIFARE and FeliCa card interface schemes, generating the necessary load modulation signals.
- **NFCIP-1 Communication:**
  - Peer-to-Peer Communication: Compliant with ECMA 340 and ISO/IEC 18092 NFCIP-1 Passive and Active communication modes, allowing communication with other NFCIP-1 compliant devices at speeds up to 424 kbit/s.

– **Connectivity and Power Management:**

- **Host Interfaces:**
  - SPI
  - I^2C
  - High-Speed UART (HSU)
- An embedded low-dropout voltage regulator allows the device to be connected directly to a battery, with a power switch included to supply power to a secure IC.

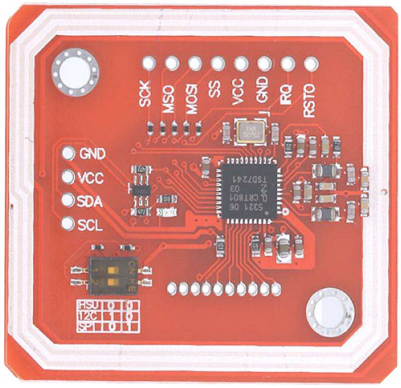

Figure 2.4: NFC module PN532 [2]

Application in Houseleek Project

In the Houseleek project, we utilize the PN532 in ISO/IEC 14443A Reader mode, interfacing it via the SPI interface. This setup allows efficient and reliable contactless communication for the system.

3. System Improvements

The main goal of this thesis was to make improvmeents to the existing Houseleek implementation in order to enhance the functionality and manageability of the whole system. These enhancements were designed to streamline development processes, ensure stability, update the components of the system, and improve user experience across different components. Here's an overview of the key improvements:

Git Repository Refactoring
The initial step involved restructuring the project's Git repository into distinct modules:

- **Server Repository:** This repository houses the backend server code responsible for managing access control policies, user authentication, and communication protocols.
- **Board Firmware Repository:** Dedicated to the firmware code running on ESP32 microcontroller boards, which implements access control logic, sensor interfacing, and communication with the server.
- **Board Schematics Repository:** Contains electronic schematics and PCB designs for the ESP32-based hardware modules, facilitating clear documentation and reproducibility in hardware development.
- **Android Application Repository:** Hosts the codebase for the Android application used for mobile access control management, providing an intuitive interface for administrators and users.

Updating the project documentation)
To facilitate developer onboarding, replication and deployment, comprehensive setup instructions were documented in repositories referenced in Appendix A:

- **Server Setup Guide:** Detailed step-by-step instructions were provided for setting up the server backend for both environment: local server setup for development and remote server setup for testing and production. This included

prerequisites, installation of dependencies using Composer, database setup, configuration of environment variables, and launching the server.

– **Android Application Setup Guide:** Clear guidelines were outlined for setting up the Android application. This included installation steps, configuration of API endpoints, handling permissions, and deploying on Android devices or emulators.

By implementing these improvements, the Houseleek project will hopefully achieve enhanced modularity, stability, and usability. The structured repository layout and thorough documentation ensure that developers and users can replicate the setup process effectively, fostering easier maintenance, scalability, and adoption of the system in various academic and professional environments. The project will serve as the basis for student introduction into the development of the comprehensive, modern system that includes technologies such as web, embedded and IoT.

3.1. The Houseleek Server

Achieving stability for the local setup of the Houseleek system involved rigorous testing and meticulous documentation.

3.1.1. Docker Setup

The server uses Docker to manage its containers.

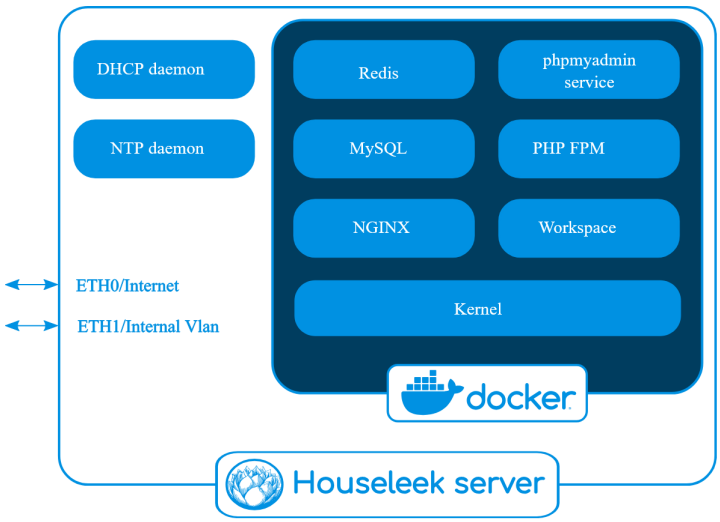

Figure 3.1: Docker containers used by the server

– `certbot`: free, open source software tool for automatically using Let's Encrypt certificates on manually-administrated websites to enable HTTPS [6].

- `nginx`: free, open source HTTP and reverse proxy server [18].
- `mysql`: MySQL is free, open source relational database management system [20].
- `php-fpm`: PHP FastCGI Process Manager. It is a primary PHP FastCGI implementation containing features mainly useful for heavy-loaded sites. These features include advanced process management with graceful stop/start, pools that allow starting workers with different uid/gid/chroot/environment, configurable logging, emergency restart capabilities, accelerated upload support, "slowlog" functionality for logging slow-executing scripts with PHP backtraces, dynamic/ondemand/static child spawning, and detailed status information with various formats like JSON, XML, and OpenMetrics supported [8]. FastCGI is a protocol designed to improve the speed and efficiency of web servers when handling requests for dynamic content, replacing the older Common Gateway Interface (CGI) [25].
- `php-worker`: background processes on servers that run PHP code. They construct pages and handle requests that require backend processing on your website. This technology creates HTML pages to serve your site visitors. PHP workers decide the number of uncached demands that your website can handle at any time. Once a PHP worker has started, it remains diligent until processes are completed, or certain conditions are met [17].
- `phpmyadmin`: free software tool written in PHP, intended to handle the administration of MySQL over the Web [21].
- `portainer`: lightweight service delivery platform for containerized applications that can be used to manage Docker, Swarm, Kubernetes and ACI environments. It is designed to be as simple to deploy as it is to use. The application allows you to manage all your orchestrator resources (containers, images, volumes, networks and more) through a 'smart' GUI and/or an extensive API [22].
- `redis`: fast in-memory database. It provides cloud and on-prem solutions for caching, vector search, and NoSQL databases that seamlessly fit into any tech stack—making it simple for digital customers to build, scale, and deploy the fast apps our world runs on [23].
- `workspace`: Runs cron jobs and other scripts, providing an environment with access to other services. It is used to update dependencies, run migration scripts or generate enrcyption keys.

3.1.2. Repository Structure

The project repository consists of two main directories: `Laradock` and `Server`. Below is an explanation of the structure and key files within these directories.

Laradock Directory

The `Laradock` directory contains all the files required for the Laradock environment. Important files and directories within `Laradock` include:

- `certbot`: Contains the Dockerfile and `run-certbot.sh`, which runs certbot to generate and copy certificates to a shared volume with nginx.
- `nginx`: Contains the Dockerfile, `logrotate` folder (for rotating logs), `sites` folder (with `local.conf.example` and `remote.conf.example` for different setups), and an `ssl` folder with certificates not directly used by our project.
- `mysql`: Contains the Dockerfile and `my.cnf`, which configures SQL mode and charset for MySQL [20].
- `php-fpm`: Contains the Dockerfile and various configuration files.
- `php-worker`: Contains the Dockerfile, `supervisord.d` folder, and `supervisord.conf` file that is used to configure supervisor. Supervisor is a client/server system that allows its users to monitor and control a number of processes on UNIX-like operating systems [5].
- `phpmyadmin`: Contains the Dockerfile for phpMyAdmin.
- `portainer`: Contains the Dockerfile for Portainer.
- `redis`: Contains the Dockerfile and `redis.conf` configuration file for Redis.
- `workspace`: Contains the Dockerfile, `crontab` folder (for scheduled tasks), and various configuration files.
- `docker-compose.yml`: Configuration file used to set up and manage all services. It lists services, their configurations, environment variables, volumes, networks, and ports [11].
- `env-example`: Example environment file containing environment variables used by `docker-compose.yml`.

Server Directory

The `Server` directory contains the code and configuration files for the Laravel (PHP) server.

Important files and directories within `Server` include:

- `app`, `bootstrap`, `config`, `database`, `routes`: Contain the PHP code for the server.
- `public`, `storage`, `resources`: Contain non-code files used by the server, such as images and .apk files for the Android application.
- `composer.json` and `composer.lock`: Configuration files for Composer, a dependency manager for PHP [19].
- `env-example-local` and `env-example-server`: Example environment files for local and remote setups.

3.1.3. Environment Files

The `.env` files configure environment variables for both local and remote deployments.

3.1.4. Environment Variables

The environment variables are configured differently for local development and remote production setups. Below are the details of these configurations, including the differences and the common variables shared between both environments.

Differences Between Local and Remote Environment Configurations

Local Environment Variables

```
APP_NAME=Houseleek
APP_ENV=local
APP_KEY=base64:ajVYkfduHku5pfJ1diGPZ32HNEst2zHQQ3tOSdwtVyE=
APP_DEBUG=true
APP_URL=http://localhost
```

Remote Environment Variables

```
APP_NAME=Houseleek
APP_ENV=production
APP_KEY=base64:ajVYkfduHku5pfJ1diGPZ32HNEst2zHQQ3tOSdwtVyE=
APP_DEBUG=false
APP_URL=https://houseleek.rasip.fer.hr
```

Explanation of Key Variables

- `APP_NAME`: The name of the application.
- `APP_ENV`: Specifies the environment the application is running in. It is set to `local` for development and `production` for the live environment.
- `APP_KEY`: The encryption key used by Laravel. It is crucial for securing user sessions and other encrypted data.
- `APP_DEBUG`: Enables or disables debug mode. Set to `true` for local development to display detailed error messages, and `false` for production to avoid exposing sensitive information.
- `APP_URL`: The base URL of the application. `http://localhost` for local development and the live URL `https://houseleek.rasip.fer.hr` for production.

Common Environment Variables

The following variables are shared between both local and remote configurations, ensuring consistency in logging, database connections, caching, session management, and email configurations.

Log Configuration

```
LOG_CHANNEL=stack
```

`LOG_CHANNEL`: Specifies the log channel. `stack` means it uses a stack of multiple channels.

ArcaneDev LogViewer Middleware

```
ARCANEDEV_LOGVIEWER_MIDDLEWARE=web,auth,systemAdministrator,verified
```

`ARCANEDEV_LOGVIEWER_MIDDLEWARE`: Middleware settings for the ArcaneDev LogViewer package, listing middleware that should be applied.

Laravel Logger Routes

```
LARAVEL_LOGGER_DISABLE_ROUTES=true
```

`LARAVEL_LOGGER_DISABLE_ROUTES`: Disables the routes for Laravel Logger when set to `true`.

Database Configuration

```
DB_CONNECTION=mysql
DB_HOST=mysql
DB_PORT=3306
DB_DATABASE=houseleek
DB_USERNAME=houseleek
DB_PASSWORD=8dYUrFRXHFfpzRAG
```

`DB_CONNECTION`, `DB_HOST`, `DB_PORT`, `DB_DATABASE`, `DB_USERNAME`, `DB_PASSWORD`: These variables specify the MySQL database connection details, including host, port, database name, username, and password.

Cache, Session, and Queue Configuration

```
CACHE_DRIVER=redis
SESSION_DRIVER=redis
QUEUE_DRIVER=redis
BROADCAST_DRIVER=log
QUEUE_CONNECTION=redis
SESSION_LIFETIME=120
```

`CACHE_DRIVER`, `SESSION_DRIVER`, `QUEUE_DRIVER`, `BROADCAST_DRIVER`, `QUEUE_CONNECTION`, `SESSION_LIFETIME`: These variables use Redis for caching, session handling, and queue management. Broadcasting is set to log, and session lifetime is set to 120 minutes.

Redis Configuration

```
REDIS_HOST=redis
REDIS_PASSWORD=null
REDIS_PORT=6379
```

`REDIS_HOST`, `REDIS_PASSWORD`, `REDIS_PORT`: Specifies Redis server details, including host, password (null for no password), and port.

Mail Configuration

```
MAIL_DRIVER=smtp
MAIL_HOST=smtp-mail.outlook.com
MAIL_PORT=587
MAIL_USERNAME=houseleek.project@outlook.com
```

```
MAIL_PASSWORD=HSvD4LoS!HSvD4LoS!
MAIL_ENCRYPTION=TLS
MAIL_FROM_ADDRESS=houseleek.project@outlook.com
MAIL_FROM_NAME="Project Houseleek"
```

MAIL_DRIVER, MAIL_HOST, MAIL_PORT, MAIL_USERNAME, MAIL_PASSWORD, MAIL_ENCRYPTION, MAIL_FROM_ADDRESS, MAIL_FROM_NAME: These variables configure the mail settings, specifying the mail driver (SMTP), server host, port, username, password, encryption protocol (TLS), and the sender's email address and name.

- MAIL_DRIVER: Specifies the mail driver, smtp in this case.
- MAIL_HOST: The SMTP server host.
- MAIL_PORT: The SMTP server port, 587.
- MAIL_USERNAME: The username for the SMTP server, which is the email address used for authentication.
- MAIL_PASSWORD: The password for the SMTP server.
- MAIL_ENCRYPTION: The encryption protocol, TLS.
- MAIL_FROM_ADDRESS: The email address that will appear in the 'from' field of outgoing emails.
- MAIL_FROM_NAME: The name that will appear in the 'from' field of outgoing emails.

3.1.5. Server Backend Stability

Testing focused on ensuring the server operated reliably after setup. This included configuring access control parameters and testing scenarios to validate performance and reliability.

3.1.6. Android Application Testing

Extensive testing ensured seamless integration between the Android application and the server. The key functionalities tested included remote door unlocking and NFC-based access control. These tests verified the robustness and responsiveness of the application in providing users with reliable access management capabilities. The Android application configuration was updated from its older version to align with modern requirements and compatibility standards.

By conducting these tests and documenting the process, I ensured that both the server backend and Android application were stable and performed as intended in

the local environment. This laid the foundation for reliable deployment and further development of the Houseleek access control system.

3.1.7. Local Server

The local server setup for the Houseleek was primarily configured for development, onboarding, and testing of features. To ensure the local server functions correctly, several adjustments were made in configuration files and Dockerfiles. These changes were essential to create a stable and replicable development environment, enabling developers to test new features, and debug issues.

Environment File Adjustment

In the Laradock .env file, the Node.js version, used by workspace container, was updated to ensure compatibility with dependencies:

```
1   WORKSPACE_NODE_VERSION=16.20.2
```

Dockerfile Adjustments

The Dockerfile for PHP-FPM (Laradock/php-fpm/Dockerfile) was updated to use the original Laradock image. The line was changed from:

```
1   FROM letsdockerize/laradock-php-fpm:2.4-${
     LARADOCK_PHP_VERSION}
```

to:

```
1   FROM laradock/php-fpm:2.2-${LARADOCK_PHP_VERSION}
```

This change was necessary because the project could not be built with the old line. The letsdockerize/laradock-php-fpm image was a temporary replacement for the original Laradock PHP-FPM image.

The FROM line in a Dockerfile specifies the base image to be used for the build process. In this case, the base image is laradock/php-fpm:2.2-x, which ensures compatibility with the Laradock environment and leverages the necessary PHP-FPM configuration for the project.

Dockerfile was also updated to use specific version of Swoole extension. Install section was changed from:

```
1   RUN if [ ${INSTALL_SWOOLE} = true ]; then \
2   # Install Php Swoole Extension
3   if [ $(php -r "echo PHP_MAJOR_VERSION;") = "5" ];
     then \
```

```
4   pecl install swoole-2.0.10; \
5   else \
6   if [ $(php -r "echo PHP_MINOR_VERSION;") = "0" ];
     then \
7   pecl install swoole-2.2.0; \
8   else \
9   pecl install swoole; \
10  fi \
11  fi && \
12  docker-php-ext-enable swoole \
13  && php -m | grep -q 'swoole' \
14  ;fi
```

to:

```
1   RUN if [ ${INSTALL_SWOOLE} = true ]; then \
2   # Install Php Swoole Extension
3   pecl install swoole-2.2.0; \
4   docker-php-ext-enable swoole \
5   && php -m | grep -q 'swoole' \
6   ;fi
```

A similar change was also made in Dockerfile for php-woker (Laradock/php-worker/Dockerfile) and Dockerfile for workspace (Laradock/workspace/Dockerfile), as both used conditional Swoole extension install selection.

Swoole is a high-performance networking framework that uses an event-driven, asynchronous, non-blocking I/O model. It can be used to develop high-performance, scalable, concurrent TCP, UDP, Unix socket, HTTP, and WebSocket services [9].

Initially, the Dockerfiles had conditional Swoole extension install selection based on the PHP version because it was set up that way by Laradock developers. For example, if the PHP version was 7.2, it executed pecl install swoole, which installs the latest version of the Swoole extension.

However, the project couldn't be built with the latest version of Swoole, and even if it could, the version changes could lead to compatibility problems in the future. Therefore, the Dockerfiles were modified to install a specific version (swoole-2.2.0) to ensure consistent and reliable builds.

Server Environment File Adjustment

In sever .env adjusted the `APP_URL` to point to the local development environment:

```
1   APP_URL=https://localhost
```

This change facilitates testing and debugging without relying on external URLs, simplifying local development.

Fixing SMTP Configuration

The SMTP settings in the server .env file were updated to ensure reliable email functionality during development. The server environment was changed from:

```
MAIL_DRIVER=smtp
MAIL_HOST=smtp.office365.com
MAIL_PORT=587
MAIL_USERNAME=houseleek.project@outlook.com
MAIL_PASSWORD=MAIL_PASSWORD
MAIL_ENCRYPTION=TLS
MAIL_FROM_ADDRESS=houseleek.project@outlook.com
MAIL_FROM_NAME="Project Houseleek"
```

to:

```
MAIL_DRIVER=smtp
MAIL_HOST=smtp-mail.outlook.com
MAIL_PORT=587
MAIL_USERNAME=houseleek.project@outlook.com
MAIL_PASSWORD=MAIL_PASSWORD
MAIL_ENCRYPTION=TLS
MAIL_FROM_ADDRESS=houseleek.project@outlook.com
MAIL_FROM_NAME="Project Houseleek"
```

The mail host was updated because the old host (smtp.office365.com) was outdated. The updated settings use smtp-mail.outlook.com to ensure the system can send verification links when registering new users, which is crucial for user onboarding and verification processes.

Fixing Email Domain Validation

The existing version of the system allowed user registration only from the `@fer.hr` domain. As part of my work, I wanted to add the functionality of registering external users by accepting all valid email domains. This change enhances the flexibility and usability of the Houseleek system, allowing a broader range of users to register and participate in the system. Inside Server/app/Http/Controllers/AdminRegistrationController.php I changed methods for registration from:

```
public function adminRegisterUser(Request $request)
{
```

```
3       $messages = [
4       'regex' => 'Only emails from @fer.hr domain are
        allowed',
5       ];
6       $validator = Validator::make($request->all(), [
7       'name' => ['required', 'string', 'max:255'],
8       'username' => ['required', 'alpha_dash', 'max:25'
        , 'unique:users'],
9       'email' => ['required', 'string', 'email', 'max
        :255', 'unique:users', 'regex:/(.*)@fer\.hr$/i'],
10      ...
11  }
```

to:

```
1   public function adminRegisterUser(Request $request)
2   {
3       $messages = [
4       'email' => 'The :attribute must be a valid email
        address.',
5       ];
6       $validator = Validator::make($request->all(), [
7       'name' => ['required', 'string', 'max:255'],
8       'username' => ['required', 'alpha_dash', 'max:25'
        , 'unique:users'],
9       'email' => ['required', 'string', 'email', 'max
        :255', 'unique:users'],
10      ...
11  }
```

Similar changes were made to other controllers to ensure comprehensive validation:

- `Server/app/Http/Controllers/Auth/RegisterController.php`
- `Server/app/Http/Controllers/TemporaryAccessController.php`
- `Server/app/Http/Controllers/UserController.php`

3.2. Remote

Ensuring the Houseleek system's remote server stability and enabling the Android application to connect seamlessly to the remote server involved several critical steps:

Environment Variable Configuration

Updating `.env` File: Modified the environment variables in the server's `.env` file to point to the remote server (houseleek.rasip.fer.hr). This step was crucial for ensuring that all services and dependencies were correctly referenced to the remote server.

Initial Setup Attempts

Following Local Setup Steps: Initially followed the steps used for the local setup, anticipating that the same procedures would work for the remote server. This included setting up the server environment, configuring access control parameters, and attempting to run the server and connect the Android application.

Troubleshooting and Fixing Issues

Identifying and Resolving Problems: Upon discovering that the initial setup did not work as intended, I identified and resolved several issues specific to the remote environment. This process involved debugging configuration errors, addressing connectivity problems, and ensuring compatibility between the server and Android application.

Creating New Documentation

Remote Setup Instructions: Developed comprehensive documentation detailing the remote server setup process. This included specific steps to configure the environment variables, resolve issues encountered during the initial setup, and ensure stable operation of both the server and the Android application.

3.2.1. Remote server

To ensure the stability and secure functionality of the remote server for the Houseleek system, several critical issues were identified and resolved through specific troubleshooting and configuration adjustments.

Fixing Permissions for Logs Folder

To address issues related to permissions for the logs folder which is visible on figure 3.4 , the following command was executed on the remote server:

```
1   chmod 777 -R /var/www/storage/logs
```

This command ensured that the server had the necessary read, write, and execute permissions to manage logs effectively, thus maintaining proper server operations.

Enabling HTTPS

To secure the connection to the remote server, HTTPS was implemented using Let's Encrypt certificates. The following steps were undertaken [24]:

1. **Setup Let's Encrypt with Laradock:**

   In the `docker-compose.yml` file, configure Certbot for obtaining SSL certificates:

```
certbot:
build:
context: ./certbot
volumes:
- ./data/certbot/certs/:/var/certs
- ./certbot/letsencrypt/:/var/www/letsencrypt
environment:
- CN=houseleek.rasip.fer.hr
- EMAIL=houseleek.project@outlook.com
networks:
- frontend

```

   Add volumes used by Certbot to the NGINX service in `docker-compose.yml`:

```
nginx:
volumes:
- ./data/certbot/certs/:/var/certs
- ./certbot/letsencrypt/:/var/www/letsencrypt

```

   Disable port 443 in Laradock's NGINX configuration (`laradock/nginx/sites/default.conf`) temporarily:

```
# For https
# listen 443 ssl;
# listen [::]:443 ssl ipv6only=on;
# ssl_certificate /etc/nginx/ssl/default.crt;
# ssl_certificate_key /etc/nginx/ssl/default.
key;

```

   Stop the NGINX container:

```
docker-compose stop nginx

```

Rebuild NGINX with the `-no-cache` option:

```
1     docker-compose build --no-cache nginx
2
```

Build the Certbot container:

```
1     docker-compose build --no-cache certbot
2
```

Start NGINX:

```
1     docker-compose up -d nginx
2
```

Use Certbot to install certificates:

```
1     docker-compose up -d certbot
2
```

Reactivate port 443 and set the correct certificate paths in `laradock/nginx/sites/default.conf`:

```
1     # For https
2     listen 443 ssl;
3     listen [::]:443 ssl ipv6only=on;
4     ssl_certificate /var/certs/fullchain1.pem;
5     ssl_certificate_key /var/certs/houseleek.
    rasip.fer.hr-privkey1.pem;
6
```

Ensure the SSL directory is set correctly in the NGINX volume settings of `docker-compose.yml`.

Finally, stop and rebuild NGINX with `-no-cache`:

```
1     docker-compose stop nginx
2     docker-compose build --no-cache nginx
3     docker-compose up -d nginx
4
```

3.2.2. Certificate Management and Auto Renewal Setup

Installing Fresh Certificates

To install new certificates, you need to start the Certbot container. Each time the container starts, new certificates will be generated. To check if Certbot is running, execute the following command:

```
sudo docker-compose ps
```

If Certbot isn't running, start the Certbot container by running the following command inside the Laradock directory:

```
sudo docker-compose up -d certbot
```

Be cautious, as there is a limit on the number of certificates that can be issued per week (currently 5). This limit resets every Monday. Unnecessary restarts of the Certbot container can quickly exhaust this limit, so ensure Docker operations are intentional.

Setting Up Auto Renewal

To automate the renewal of certificates, set up a cron job on the host server (the server running Docker, not inside the Docker container). If this project is moved (to a different directory or server) or if the server is wiped clean, you will need to repeat these steps.

Editing Cron Jobs

To edit cron jobs on the server, use the command:

```
sudo crontab -e
```

This will open the cron job editor in the terminal. Add the following lines at the end of the file to set up auto renewal:

```
0 12 * * * /usr/local/bin/docker-compose -f /home/mjanic/server/Laradock/docker-co
0 0 * * * /usr/local/bin/docker-compose -f /home/mjanic/server/Laradock/docker-com
```

Explanation of Cron Job Syntax

The format for cron jobs is as follows:

```
|------------------------------ Minute (0-59)
|     |------------------------ Hour (0-23)
|     |     |------------------- Day of the month (1-31)
|     |     |     |------------- Month (1-12; or JAN to DEC)
|     |     |     |     |------- Day of the week (1-7; or MON to SUN)
|     |     |     |     |
0     12    *     *     * runs the job at 12:00 PM (midday) every day.
0     0     *     *     * runs the job at 12:00 AM (midnight) every day.
```

Explanation of Commands

Docker Compose Path

`/usr/local/bin/docker-compose` specifies the full path to the docker-compose binary. This is necessary to avoid potential issues with the `PATH` environment variable, ensuring the cron job can find and execute Docker Compose correctly.

Docker Compose File Location

`-f /home/mjanic/server/Laradock/docker-compose.yml` points to the location of the `docker-compose.yml` file, which contains information about the Docker containers. This path needs to be updated if the project is moved or the server setup changes.

Executing Commands in Containers

`exec certbot` and `exec nginx` refer to executing commands within the certbot and nginx Docker containers, respectively.

Certbot and Nginx Commands

- `/root/certbot/run-certbot.sh` runs a script inside the Certbot container that renews the certificates.
- `nginx -s reload` sends a signal to the Nginx server inside the Docker container to reload its configuration and certificates without stopping the server.

Fixing MySQL Execution Issues

The MySQL service encountered issues where it would not execute commands, even under the root user. As a last resort, the `--skip-grant-tables` option was added to the MySQL command in `docker-compose.yml`. This option bypasses the permission

checks for MySQL, allowing administrative access to fix user account issues and reset passwords.

These configurations and adjustments ensured that the remote server for the Houseleek system was secure, stable, and operational, meeting the project's requirements for reliability and functionality in a remote environment.

3.3. Android Client

To build android client application I used Gradle Build Tool. Gradle Build Tool is a fast, dependable, and adaptable open-source build automation tool with an elegant and extensible declarative build language [12]. In this case it uses build script to build Houseleek android application.

Gradle Build Files

Generally, a build script details build configuration, tasks, and plugins. Every Gradle build comprises at least one build script.

In the build file, two types of dependencies can be added:

1. The libraries and/or plugins on which Gradle and the build script depend.
2. The libraries on which the project sources (i.e., source code) depend.

Build scripts can be either a `build.gradle` file written in Groovy or a `build.gradle.kts` file in Kotlin. The Groovy DSL (Domain-Specific Language) and the Kotlin DSL are the only accepted languages for Gradle scripts.

Project Structure

The structure of the Houseleek Android app repository is as follows:

```
HouseleekApp/
|-- app/
|-- gradle/wrapper/
|-- build.gradle
|-- gradle.properties
|-- gradlew
|-- gradlew.bat
|-- settings.gradle
```

Explanation of Key Files

`build.gradle`: This top-level build file is where you can add configuration options common to all sub-projects/modules.

```
buildscript {
  repositories {
    google()
    jcenter()
  }
  dependencies {
    classpath 'com.android.tools.build:gradle:8.3.0'
  }
}

allprojects {
  repositories {
    google()
    jcenter()
  }
}

task clean(type: Delete) {
  delete rootProject.buildDir
}
```

Listing 3.1: Top-level build.gradle

`gradle.properties`: Project-wide Gradle settings. IDE (e.g., Android Studio) users should note that Gradle settings configured through the IDE will override any settings specified in this file.

```
# Project-wide Gradle settings.
android.defaults.buildfeatures.buildconfig=true
android.enableJetifier=true
android.nonFinalResIds=false
android.nonTransitiveRClass=false
android.useAndroidX=true
org.gradle.jvmargs=-Xmx1536m
```

Listing 3.2: gradle.properties

`gradlew` and `gradlew.bat`: These scripts are used to execute Gradle commands on UNIX-based and Windows systems, respectively. They help ensure that the correct version of Gradle is used regardless of the version installed on the local machine.

`settings.gradle`: This file includes the modules to be built. For the Houseleek app, it includes the `app` module.

```
include ':app'
```

Listing 3.3: settings.gradle

Configuring the Houseleek Android application for development involved several key adjustments across different configuration files:

HouseleekApp/app/build.gradle

Added namespace houseleek.rasip.fer.hr to the build.gradle file:

```
namespace 'houseleek.rasip.fer.hr'
```

This namespace declaration ensures that the application's resources are properly scoped within the specified namespace, facilitating organization and resource management. It is used as the Java package name for its generated R and BuildConfig classes.

HouseleekApp/app/src/main/AndroidManifest.xml

Modified the package declaration: Removed the package declaration package="houseleek.rasip.fer.hr" from the AndroidManifest.xml file. This change allows the Android system to assign the appropriate package name automatically based on the application's structure and configurations.

HouseleekApp/app/src/main/res/values/strings.xml

Updated strings related to API configuration: Changed from:

```
<!-- Strings related to api -->

<string name="client_secret">MSwvd9B3b7Jb4nqdW4jLy4ZIKC0Mt1oFhsvXA75v</
  string>
<string name="client_id">2</string>
<!--<string name="base_url">http://192.168.1.50/</string>-->

<string name="base_url">https://houseleek.rasip.fer.hr/</string>
```

to:

```
<!-- Strings related to api -->

<string name="client_secret">bbPBt6RQRD6ZAxGFpU3qDMWlkdSEURpvkGwNhlZ8</
  string>
<string name="client_id">2</string>
<!--<string name="base_url">http://192.168.1.50/</string>-->

<string name="base_url">https://houseleek.rasip.fer.hr/</string>
```

To connect to a local server, comment out `<string name="base_url">https://houseleek.rasip.f` uncomment `<string name="base_url">http://192.168.1.50/</string>` and edit it with the local IP address.

HouseleekApp/app/src/main/res/xml/network_security_config.xml

Updated to allow cleartext traffic for local server usage: From:

```
<domain-config cleartextTrafficPermitted="true">
<domain includeSubdomains="true">161.53.67.82</domain>
</domain-config>
```

To:

```
<domain-config cleartextTrafficPermitted="true">
<domain includeSubdomains="true">192.168.1.50</domain>
</domain-config>
```

This change permits cleartext traffic for the local server at 192.168.1.50 to facilitate development and testing scenarios.

HouseleekApp/build.gradle

Changed the Gradle plugin version in the build.gradle file to:

```
com.android.tools.build:gradle:8.3.0
```

The Gradle plugin version has been updated from 4.1.3 to 8.3.0 to utilize the latest features and improvements.

HouseleekApp/gradle.properties

Adjusted Gradle properties:

```
android.defaults.buildfeatures.buildconfig=true
android.enableJetifier=true
android.nonFinalResIds=false
android.nonTransitiveRClass=false
android.useAndroidX=true
org.gradle.jvmargs=-Xmx1536m
```

Added properties to control build features and optimize the build process (android.defaults.buildfeatur android.nonFinalResIds, and android.nonTransitiveRClass).

HouseleekApp/gradle/wrapper/gradle-wrapper.properties

Modified the distribution URL in gradle-wrapper.properties to:

```
https://services.gradle.org/distributions/gradle-8.4-bin.zip
```

The Gradle distribution has been updated from 6.5 to 8.4 to ensure compatibility with the latest Android Gradle plugin and to benefit from new features and improvements.

HouseleekApp/app/src/main/java/houseleek/rasip/fer/hr/LoginActivity.java
Modified email validation logic:

```
private boolean isUsernameOrEmailValid(String usernameOrEmail) {
  if (usernameOrEmail.contains("@")) {
    // It must be an email, validate with regex
    String regexPattern = "^[a-zA-Z0-9_+&-]+(?:\.[a-zA-Z0-9_+&-]+)*@
  (?:[a-zA-Z0-9-]+\.)+[a-zA-Z]{2,7}$";
    return Pattern.compile(regexPattern).matcher(usernameOrEmail).
  matches();
  } else {
```

The email validation logic has been updated to use a regex pattern to check for a valid email format, rather than just checking if it contains @fer.hr. This makes the email validation allow users that registered as external users by accepting all valid email domains using best practices [27].

Generating New Client ID and Secret To enhance security and functionality, new client credentials (client ID and secret) were generated for the Android application: This step is necessary when installing Houseleek server from scratch (when the database is empty).

1. **Generate Client Credentials:** Run the following command on the server using Laravel Passport inside the workspace container (after docker-compose exec workspace bash):

   ```
   1    php artisan passport
   2
   ```

   This command generates new client credentials required for authentication between the Android application and the remote server.

2. **Integrate Client Credentials:** Update the Android application with the newly generated client ID and secret by editing them in the ‘strings.xml‘ file.

3.4. Building and Deploying APK for Android

I wrote following steps to build an APK for Android in Android Studio and make it available on the server:

3.4.1. Building the APK

1. **Open Android Studio**: Launch Android Studio and open your project.

2. **Initiate the Build Process**:
   (a) Navigate to the top menu bar.
   (b) Select `Build -> Build Bundle(s) / APK(s) -> Build APK(s)`.
3. **Wait for the Build to Finish**: Allow the build process to complete. This might take a few minutes.
4. **Locate the Built APK**:
   (a) Once the build is finished, a notification will appear in the bottom right corner of Android Studio.
   (b) Click `Locate` on the popup to open the folder containing the built APK.
5. **Find the APK**:
   - Open the debug folder from the popup that appears in the bottom right corner after the build is completed.
   - Alternatively, navigate to `HouseleekApp/app/build/outputs/apk/debug` in your project directory.
   - Look for the file named `Houseleek-1.2.0-release.apk`.

3.4.2. Deploying the APK to the Server

1. **Copy the APK**:
   - Copy the `Houseleek-1.2.0-release.apk` file from the debug folder.
2. **Paste and Overwrite on Server**:
   - Navigate to the server files directory `Server/public/storage`.
   - Paste the copied APK file into this directory.
   - Make sure to overwrite the existing APK. This ensures the new version is available for download and correctly named.

3.4.3. Important Notes

- **Consistent Naming**: Ensure the APK file is named exactly `Houseleek-1.2.0-release.apk` when pasting it into the server directory. If it is named differently, it will not replace the existing APK, and the new version will not be available for download.
- **Backup**: It is a good practice to back up the existing APK file before overwriting it, in case you need to revert to the previous version.

By following these steps, you can successfully build the APK in Android Studio and update the server with the latest version.

Figure 3.2: Laradock structure

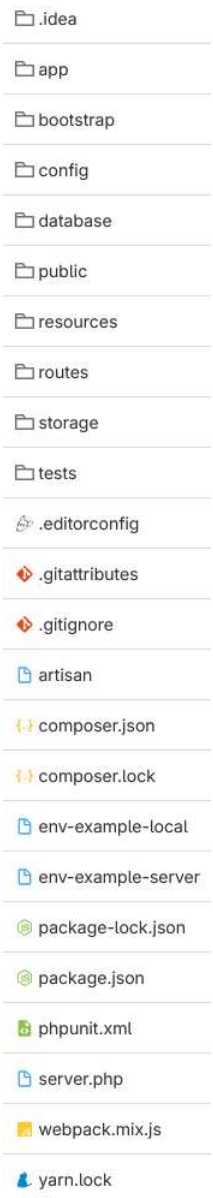

Figure 3.3: Server file structure

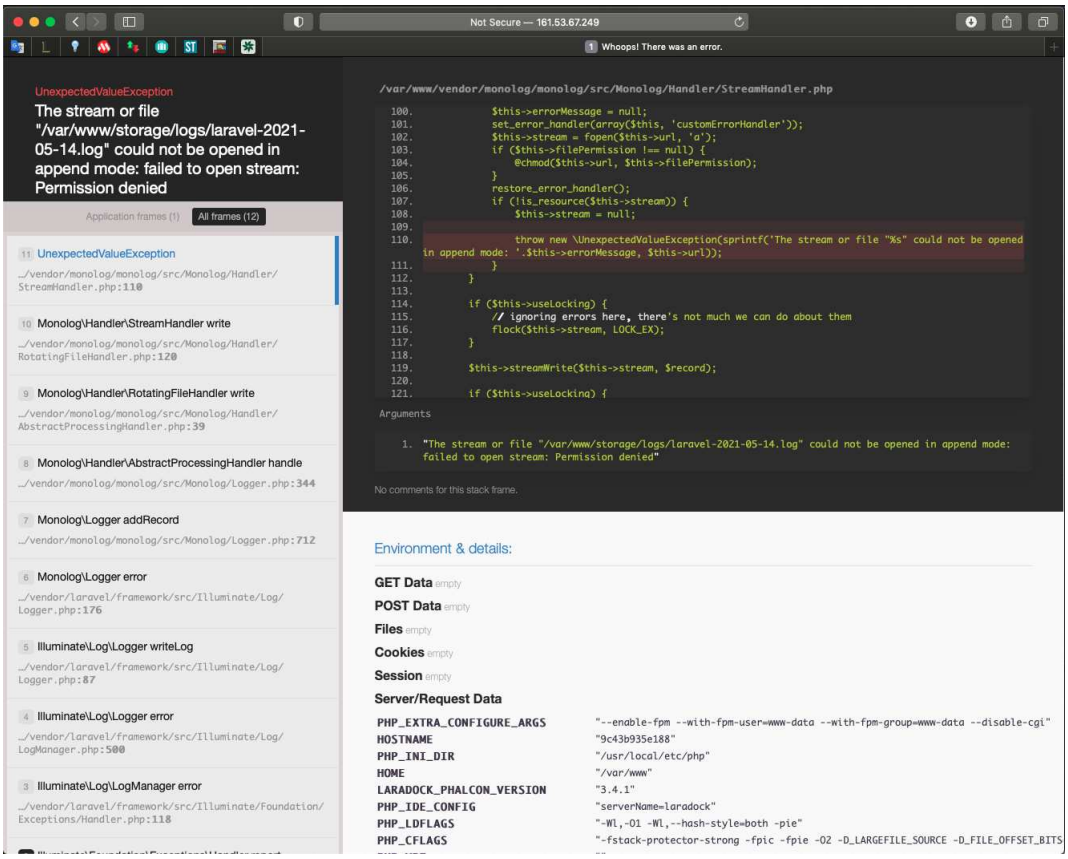

Figure 3.4: Error due to missing permissions for the logs folder

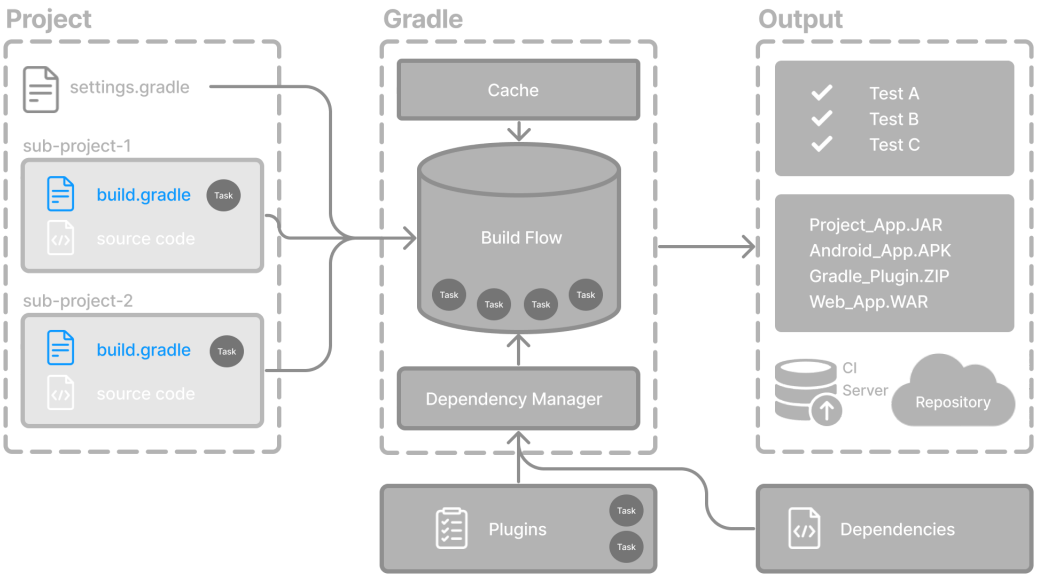

Figure 3.5: Build File Basics

4. Conclusion

The improvements made to the Houseleek system have significantly enhanced its functionality, stability, and user experience. By splitting the Git repository into separate components for the server, board firmware, board schematics, and Android application, we achieved a more modular and maintainable codebase. This restructuring facilitated targeted improvements and streamlined development processes.

The local setup process involved meticulous testing and documentation to ensure that both the server backend and Android application operated reliably. Through detailed documentation, I provided clear instructions for setting up the server and Android application, enabling others to replicate the working environment with ease.

Addressing the challenges of the remote server setup required troubleshooting permissions issues and enabling HTTPS for secure communication. Fixing these issues ensured that the server was secure and operational, providing a stable foundation for the Houseleek system.

The Android application was adapted to connect seamlessly with the remote server by updating the base URL and generating new client credentials. These adjustments ensured that the application maintained its functionality and security when interfacing with the remote server.

The goal of this work was not only to improve technical aspects but also to enhance user involvement. By automating processes through Docker for portability and Infrastructure as Code (IaC), we aimed to simplify onboarding for new students and contributors. This approach made it easier for newcomers to get involved with the project quickly, reducing setup time and ensuring consistency across different development environments.

Overall, these improvements have strengthened the Houseleek system's capability to manage access control in departments and rooms. The enhanced administration and configuration interface, along with the dynamic refreshment and addition of new functionalities, have made the system more robust and adaptable.

Bibliography

[1] Android (operating system), 2024. URL https://en.wikipedia.org/wiki/Android_(operating_system).

[2] Components 101. Pn532- nfc rfid module, 2018. URL https://components101.com/wireless/pn532-nfc-rfid-module.

[3] Inc. Amazon Web Services. What is docker? | aws, 2024. URL https://aws.amazon.com/docker/.

[4] GitLab B.V. The most-comprehensive ai-powered devsecops platform | gitlab, 2024. URL https://about.gitlab.com/.

[5] Agendaless Consulting i Contributors. Supervisor: A process control system, 2024. URL http://supervisord.org/.

[6] Electronic Frontier Foundation. About certbot, 2024. URL https://certbot.eff.org/pages/about.

[7] git. git - scm, 2024. URL https://git-scm.com/.

[8] The PHP Documentation Group. Php: Fastcgi process manager (fpm) - manual, 2024. URL https://www.php.net/manual/en/install.fpm.php.

[9] The PHP Documentation Group. Php: Introduction - manual, 2024. URL https://www.php.net/manual/en/intro.swoole.php.

[10] Docker Inc. Containerize an application | docker docs, 2024. URL https://docs.docker.com/guides/workshop/02_our_app/.

[11] Docker Inc. How compose works | docker docs, 2024. URL https://docs.docker.com/compose/compose-application-model/.

[12] Gradle Inc. Gradle user manual, 2024. URL https://docs.gradle.org/current/userguide/userguide.html.

[13] ITU-T. Internet of things (iot) [itu-t y.2060], 2012. URL https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-Y.2060-201206-I!!PDF-E&type=items. ITU-T Recommendation Y.2060, 06/2012.

[14] Jure Knezović. Deep learning applications in iot environments for smart building automation. Magistarski rad, Faculty of Electrical Engineering and Computing, Zagreb, 2019.

[15] Laradock. Laradock - php development environment for docker, 2024. URL https://laradock.io.

[16] Laravel. Github - laravel, 2024. URL https://github.com/laravel/laravel.

[17] Cloudways Ltd. What are php workers and why do you need them?, 2024. URL https://www.cloudways.com/blog/php-workers/#php-workers.

[18] nginx. nginx, 2024. URL https://nginx.org/en/.

[19] Jordi Boggiano Nils Adermann i many community contributions. Composer, 2024. URL https://getcomposer.org/.

[20] Oracle. What is mysql?, 2024. URL https://www.oracle.com/mysql/what-is-mysql/.

[21] phpMyAdmin contributors. phpmyadmin, 2024. URL https://www.phpmyadmin.net/.

[22] portainer contributors. Github - portainer/portainer: Making docker and kubernetes management easy., 2024. URL https://github.com/portainer/portainer.

[23] Redis.

[24] rudy from Chill Bits. Setup letsencrypt with laradock – chill bits, 2021. URL https://chillbits.com/2021/06/24/setup-letsencrypt-with-laradock/.

[25] Nord Security. Fastcgi definition – glossary | nordvpn, 2024. URL https://nordvpn.com/cybersecurity/glossary/fastcgi/.

[26] W3Schools. Laravel introduction, 2024. URL https://www.w3schools.in/laravel/intro.

[27] Achim Weilin Zhong. Owasp validation regex repository | owasp foundation, 2024. URL https://owasp.org/www-community/OWASP_Validation_Regex_Repository.

Appendix A
Setup Guides

Houseleek Server Installation Guide

Welcome to the Houseleek server installation guide. This guide will walk you through the installation process for setting up the Houseleek server on different environments.

Installation Guides

- Local Server Installation Guide for Windows: This guide is for setting up the Houseleek server on a using Windows.
- Server Installation Guide For Linux: This guide is for installing the Houseleek server on a server using Linux.

Starting Server After Restart

If server installation is completed and server needs to be restarted, repeat step 8 from installation guide in `Laradock` directory.

Environment Configuration for Houseleek

This document explains the environment configuration variables used for the Houseleek project. The configuration is divided into local and remote (production) setups, with explanations for each section.

Differences Between Local and Remote Environment Configurations

Local Environment Variables

```
APP_NAME=Houseleek
APP_ENV=local
APP_KEY=base64:ajVYkfduHku5pfJ1diGPZ32HNEst2zHQQ3tOSdwtVyE=
APP_DEBUG=true
APP_URL=http://localhost
```

Remote Environment Variables

```
APP_NAME=Houseleek
APP_ENV=production
APP_KEY=base64:ajVYkfduHku5pfJ1diGPZ32HNEst2zHQQ3tOSdwtVyE=
APP_DEBUG=false
APP_URL=https://houseleek.rasip.fer.hr
```

Explanation

- **APP_NAME**: The name of the application.
- **APP_ENV**: Specifies the environment the application is running in. It is set to `local` for development and `production` for the live environment.

- **APP_KEY**: The encryption key used by Laravel. It is crucial for securing user sessions and other encrypted data.
- **APP_DEBUG**: Enables or disables debug mode. `true` for local development to display detailed error messages, and `false` for production to avoid exposing sensitive information.
- **APP_URL**: The base URL of the application. `http://localhost` for local development and the live URL `https://houseleek.rasip.fer.hr` for production.

Common Environment Variables

The following variables are shared between both local and remote configurations.

Log Configuration

```
LOG_CHANNEL=stack
```

- **LOG_CHANNEL**: Specifies the log channel. `stack` means it uses a stack of multiple channels.

ArcaneDev LogViewer Middleware

```
ARCANEDEV_LOGVIEWER_MIDDLEWARE=web,auth,systemAdministrator,verified
```

- **ARCANEDEV_LOGVIEWER_MIDDLEWARE**: Middleware settings for the ArcaneDev LogViewer package, listing middleware that should be applied.

Laravel Logger Routes

```
LARAVEL_LOGGER_DISABLE_ROUTES=true
```

- **LARAVEL_LOGGER_DISABLE_ROUTES**: Disables the routes for Laravel Logger when set to `true`.

Database Configuration

```
DB_CONNECTION=mysql
DB_HOST=mysql
DB_PORT=3306
DB_DATABASE=houseleek
DB_USERNAME=houseleek
DB_PASSWORD=8dYUrFRXHFfpzRAG
```

- **Database Configuration**: Specifies the MySQL database connection details including host, port, database name, username, and password.

Cache, Session, and Queue Configuration

```
CACHE_DRIVER=redis
SESSION_DRIVER=redis
QUEUE_DRIVER=redis
BROADCAST_DRIVER=log
QUEUE_CONNECTION=redis
SESSION_LIFETIME=120
```

- **Cache, Session, and Queue Configuration**: Uses Redis for caching, session handling, and queue management. Broadcasting is set to log and session lifetime is set to 120 minutes.

Redis Configuration

```
REDIS_HOST=redis
REDIS_PASSWORD=null
REDIS_PORT=6379
```

- **Redis Configuration**: Specifies Redis server details including host, password (null for no password), and port.

Mail Configuration

```
MAIL_DRIVER=smtp
MAIL_HOST=smtp-mail.outlook.com
MAIL_PORT=587
MAIL_USERNAME=houseleek.project@outlook.com
MAIL_PASSWORD=HSvD4LoS!HSvD4LoS!
MAIL_ENCRYPTION=TLS
MAIL_FROM_ADDRESS=houseleek.project@outlook.com
MAIL_FROM_NAME="Project Houseleek"
```

- **Mail Configuration**:
  - **MAIL_DRIVER**: Specifies the mail driver, `smtp` in this case.
  - **MAIL_HOST**: The SMTP server host.
  - **MAIL_PORT**: The SMTP server port, `587`.
  - **MAIL_USERNAME**: The username for the SMTP server, which is the email address used for authentication.
  - **MAIL_PASSWORD**: The password for the SMTP server.
  - **MAIL_ENCRYPTION**: The encryption protocol, `TLS`.
  - **MAIL_FROM_ADDRESS**: The email address that will appear in the 'from' field of outgoing emails.
  - **MAIL_FROM_NAME**: The name that will appear in the 'from' field of outgoing emails.

Local Server Installation Guide for Windows

Prerequisites

Make sure you have git and Docker installed. If not, download and install git and Docker.

Installation Steps

1. Run Docker

If you haven't already, run Docker by clicking on Docker Desktop on your Windows start menu.

2. Download or clone project from git repository

Clone the project repository using the git clone command.

3. Navigate to Laradock Directory

Once the project is cloned, navigate to the Laradock directory using the following command:

```
cd Laradock
```

4. Copy Environment File

Copy the `env-example` file to `.env` by executing the following command:

```
cp env-example .env
```

5. Copy NGINX config File

Copy the `local.conf.example` file to `default.conf` by executing the following command:

```
cp nginx/sites/local.conf.example nginx/sites/default.conf
```

6. Configure Server Environment

Navigate to the `Server` directory and copy the `env-example-local` file to `.env`:

```
cd ..
cd Server
cp env-example-local .env
```

7. Return to Laradock directory

Before starting the Docker containers, return to the `Laradock` directory:

```
cd ..
cd Laradock
```

8. Start Docker Containers

Start the Docker containers by running the following command:

```
docker-compose up -d nginx phpmyadmin mysql redis portainer php-worker workspace
```

9. Access Workspace Container

After all containers are up and running run the following command:

```
docker-compose exec workspace bash
```

10. Update Dependencies

The command from step 9 will position you inside the bash of the container where Laravel PHP Server resides. Inside the container, run the following command to update all necessary server files:

```
composer update
```

11. Run Database Migrations

Inside the container, execute the following command to create a fresh database with a default user: houseleek@fer.hr, password:NekiBoljiPasvord456+

```
php artisan migrate:fresh
```

12. Access the Server

After completing all the above steps, the server should be accessible at port 80. Simply enter `localhost` in your web browser.

13. Generating client ID and secret for Android application (optional)

While still inside the container, execute the following command to generate a client ID and secret. These credentials are used by Android applications to securely communicate with the server. If you don't plan to develop or use an Android application with this server, you can skip this step.

```
php artisan passport:install
```

Note

Make sure Docker is running throughout this process.

Server Installation Guide for Linux

Prerequisites

Make sure you have git, docker engine and docker-compose installed.

Installation Steps

1. Run Docker

If you haven't already, run Docker daemon service.

```
sudo systemctl start docker
```

2. Download or clone project from git repository

Clone the project repository using the git clone command.

3. Navigate to Laradock Directory

Once the project is cloned, navigate to the Laradock directory using the following command:

```
cd Laradock
```

4. Copy Environment File

Copy the `env-example` file to `.env` by executing the following command:

```
cp env-example .env
```

5. Copy NGINX config File

Copy the `remote.conf.example` file to `default.conf` by executing the following command:

```
cp nginx/sites/remote.conf.example nginx/sites/default.conf
```

6. Configure Server Environment

Navigate to the `Server` directory and copy the `env-example-server` file to `.env`:

```
cd ..
cd Server
cp env-example-server .env
```

7. Return to Laradock directory

Before starting the Docker containers, return to the `Laradock` directory:

```
cd ..
cd Laradock
```

8. Start Docker Containers

Start the Docker containers by running the following command:

```
sudo docker-compose up -d nginx phpmyadmin mysql redis portainer php-worker
workspace certbot
```

9. Access Workspace Container

After all containers are up and running run the following command:

```
sudo docker-compose exec workspace bash
```

10. Update Dependencies

The command from step 9 will position you inside the bash of the container where Laravel PHP Server resides. Inside the container, run the following command to update all necessary server files:

```
composer update
```

11. Run Database Migrations

Inside the container, execute the following command to create a fresh database with a default user: houseleek@fer.hr, password:NekiBoljiPasvord456+

```
php artisan migrate:fresh
```

12. Access the Server

After completing all the above steps, the server should be accessible at port 80. Simply enter `houseleek.rasip.fer.hr` in your web browser.

13. Generating client ID and secret for Android application (optional)

While still inside the container, execute the following command to generate a client ID and secret. These credentials are used by Android applications to securely communicate with the server. If you don't plan to develop or use an Android application with this server, you can skip this step.

```
php artisan passport:install
```

Note

Make sure Docker is running throughout this process.

Certificate Management and Auto Renewal Setup

Installing Fresh Certificates

To install new certificates, you need to start the Certbot container. Each time the container starts, new certificates will be generated. Certbot is started in step 8. You can check if certbot is running by running:

```
sudo docker-compose ps
```

If Certbot isn't running you can start Certbot container run following command inside Laradock directory:

```
sudo docker-compose up -d certbot
```

Be cautious, as **there is a limit on the number of certificates that can be issued per week (currently 5)**. This limit resets every Monday. Unnecessary restarts of the Certbot container can quickly exhaust this limit, so ensure Docker operations are intentional.

Setting Up Auto Renewal

To automate the renewal of certificates, you need to set up a cron job on the host server (the server running Docker, not inside the Docker container). If this project is moved (to a different directory or server) or if the server is wiped clean, you will need to repeat these steps.

Editing Cron Jobs

To edit cron jobs on the server, use the command:

```
sudo crontab -e
```

This will open the cron job editor in the terminal. Add the following lines at the end of the file to set up the auto renewal:

```
0 12 * * * /usr/local/bin/docker-compose -f /home/mjanic/server/Laradock/docker-compose.yml exec certbot /root/certbot/run-certbot.sh
0 0 * * * /usr/local/bin/docker-compose -f /home/mjanic/server/Laradock/docker-compose.yml exec nginx nginx -s reload
```

Explanation of Cron Job Syntax

The format for cron jobs is as follows:

```
|------------------------------ Minute (0-59)
|     |------------------------ Hour (0-23)
|     |     |------------------- Day of the month (1-31)
|     |     |     |------------- Month (1-12; or JAN to DEC)
|     |     |     |     |------- Day of the week (1-7; or MON to SUN)
|     |     |     |     |
*     *     *     *     *
```

`0 12 * * *` runs the job at 12:00 PM (midday) every day.
`0 0 * * *` runs the job at 12:00 AM (midnight) every day.

Explanation of Commands

Docker Compose Path

`/usr/local/bin/docker-compose` specifies the full path to the docker-compose binary. This is necessary to avoid potential issues with the PATH environment variable, ensuring the cron job can find and execute Docker Compose correctly.

Docker Compose File Location

`-f /home/mjanic/server/Laradock/docker-compose.yml` points to the location of the docker-compose.yml file, which contains information about the Docker containers. **This path needs to be updated if the project is moved or the server setup changes.**

Executing Commands in Containers

`exec certbot` and `exec nginx` refer to executing commands within the certbot and nginx Docker containers, respectively.

Certbot and Nginx Commands

`/root/certbot/run-certbot.sh` runs a script inside the Certbot container that renews the certificates. `nginx -s reload` sends a signal to the Nginx server inside the Docker container to reload its configuration and certificates without stopping the server.

Local Android Application Installation Guide

Prerequisites

Before you begin, make sure you have Android Studio installed on your computer. If not, download and install it from the provided link.

Installation Steps

1. Run Android Studio

If you haven't already, launch Android Studio.

2. Download or Clone Project from Git Repository

Clone the project repository using the git clone command.

3. Open Project in Android Studio

Navigate to Android Studio, go to File -> Open, and then locate the project directory. Inside it, select the "HouseleekApp" directory.

4. Modify `strings.xml`

1. Locate the file `strings.xml` at `res/values/strings.xml`.
2. Double click to open it.

Inside this file, you'll find the following lines:

```
<!-- Strings related to API -->
<string name="client_secret">DvPz1GaqtI0lsvbZL50WlsshKDXsdkrXXGjlNZj8</string> <!
- CHANGE SECRET-->
<string name="client_id">1</string> <!-- CHANGE ID-->
<string name="base_url">http://192.168.0.12/</string> <!-- CHANGE TO SERVER IP-->
```

Modify the values in the first two lines with the secret and ID generated in step 12 from the server README.

Change the value in the third line to your server IP (your local IP if connecting to local server). You can retrieve this by running ipconfig on Windows OS or ip addr on Unix OS.

5. Run the Application

You can run the application either in an emulator or by connecting your smartphone to the PC via USB. Application can be started by using *green play button* at upper right corner of Android studio IDE.

Building and Deploying APK for Android

Follow these steps to build an APK for Android in Android Studio and make it available on the server:

Building the APK

1. **Open Android Studio**: Launch Android Studio and open your project.
2. **Initiate the Build Process**:
    - Navigate to the top menu bar.
    - Select `Build` -> `Build Bundle(s) / APK(s)` -> `Build APK(s)`.
3. **Wait for the Build to Finish**: Allow the build process to complete. This might take a few minutes.
4. **Locate the Built APK**:
    - Once the build is finished, a notification will appear in the bottom right corner of Android Studio.
    - Click `Locate` on the popup to open the folder containing the built APK.
5. **Find the APK**:
    - You can open the `debug` folder from the popup that appears in the bottom right corner after the build is completed.
    - Alternatively, navigate to `HouseleekApp\app\build\outputs\apk\debug` in your project directory.
    - Look for the file named `Houseleek-1.2.0-release.apk`.

Deploying the APK to the Server

1. **Copy the APK**:
    - Copy the `Houseleek-1.2.0-release.apk` file from the `debug` folder.
2. **Paste and Overwrite on Server**:
    - Navigate to the server files directory `Server\public\storage`.
    - Paste the copied APK file into this directory.
    - Make sure to **overwrite the existing APK**. This ensures the new version is available for download and correctly named.

Important Notes

- **Consistent Naming**: Ensure the APK file is named exactly `Houseleek-1.2.0-release.apk` when pasting it into the server directory. If it is named differently, it will not replace the existing APK, and the new version will not be available for download.
- **Backup**: It is a good practice to back up the existing APK file before overwriting it, in case you need to revert to the previous version.

By following these steps, you can successfully build the APK in Android Studio and update the server with the latest version.

Improvements of Internet of Things-based access control system

Abstract

This thesis focuses on enhancing the Houseleek system, an IoT-based access control solution originally developed at the Faculty of Electrical Engineering and Computing. The Houseleek system controls access to departments and rooms using a hierarchical user role structure. Key improvements include the ability to remotely configure access parameters, define specific user roles, and update embedded modules dynamically.

The project involved reorganizing the Git repository into distinct modules for the server, firmware, schematics, and Android application to create a more modular and maintainable codebase. Stability of the server and Android application was ensured through thorough testing and detailed documentation. Setting up the remote server required resolving permission issues and enabling secure communication via HTTPS. The Android application was updated to connect seamlessly with the remote server by modifying the base URL and generating new credentials.

These enhancements significantly improved the Houseleek system's functionality, stability, and user experience, resulting in a robust solution for IoT-based access control. The thesis advances the field of IoT applications in access control and paves the way for future developments in smart building technologies.

Keywords: Internet of Things, Laravel, Docker, Access Control

Unapređenje sustava za kontrolu pristupa zasnovanog na tehnologiji Interneta stvari

Sažetak

Ovaj diplomski rad usmjeren je na unapređenje sustava Houseleek, rješenja za kontrolu pristupa temeljenog na IoT-u, izvorno razvijenog na Fakultetu elektrotehnike i računarstva. Sustav Houseleek kontrolira pristup zavodima i sobama pomoću hijerarhijske strukture korisničkih uloga. Ključna poboljšanja uključuju mogućnost daljinskog konfiguriranja parametara pristupa, definiranja specifičnih korisničkih uloga i dinamičkog ažuriranja ugrađenih modula.

Projekt je uključivao reorganizaciju Git repozitorija u različite module za poslužitelj, firmware, sheme i Android aplikaciju kako bi se stvorila modularnija baza koda koja se može održavati. Stabilnost poslužitelja i Android aplikacije osigurana je temeljitim testiranjem i detaljnom dokumentacijom. Postavljanje udaljenog poslužitelja zahtijevalo je rješavanje problema s dozvolama i omogućavanje sigurne komunikacije putem HTTPS-a. Android aplikacija ažurirana je za besprijekorno povezivanje s udaljenim poslužiteljem modificiranjem osnovnog URL-a i generiranjem novih vjerodajnica.

Ova su poboljšanja značajno poboljšala funkcionalnost, stabilnost i korisničko iskustvo sustava Houseleek, što je rezultiralo robusnim rješenjem za kontrolu pristupa temeljenu na IoT-u. Diplomski rad unapređuje područje IoT aplikacija u kontroli pristupa i utire put budućem razvoju tehnologija pametnih zgrada.

Ključne riječi: Internet stvari,Laravel,Docker,Kontrola pristupa