UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Kaarel Sõrmus

Live feed and video streaming application

Master's Thesis (30 ECTS)

Supervisor: Satish Narayana Srirama, PhD

Tartu 2020

Live feed and video streaming application

Abstract:

Nowadays many companies support working remotely and have multiple offices in different parts of the world. For companies it is important to have good communication channels, which would support collaboration between colleagues. The purpose of this thesis is to create a Computer Supported Cooperative Work web application that could support collaborative work within a company. The application created during the thesis enables live video feed between two offices, shows who are present at the office, and creates a platform for the company's employees to share ideas and news.

Keywords:

Web application, Internet of Things, Computer Supported Cooperative Work

CERCS: P170 Computer science, numerical analysis, systems, control; T120 Systems engineering, computer technology

Reaalajas postituste ja videoülekande edastuse rakendus

Lühikokkuvõte:

Tänapäeval on paljudel ettevõtetel kontorid mitmes linnas ning osades ettevõttetes on võimalik kodukontorites tööd teha. Ettevõtete jaoks on väga tähtis, et töötajatel oleksid head suhtlemise ja koostöötamise võimalused. Magistritöö eesmärk on luua koostööplatvorm, mis toetaks ettevõttesisest suhtlust ja ühistööd. Tarkvara toetab kontoritevahelist videoülekannet ning suudab tuvastada kontoris olevaid inimesi. Lisaks võimaldab veebirakendus kolleegidel ideid ja uudiseid jagada.

Võtmesõnad:

Veebirakendus, asjade internet, koostööplatvorm

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine; T120 Süsteemitehnoloogia, arvutitehnoloogia

Table of Contents

List of abbreviations

AAD – Azure Active Directory

API – Application programming interface

CSCW – Computer Supported Cooperative Work

GPIO – General-purpose input/output

HTML – Hypertext Markup Language

HTTP – Hypertext Transfer Protocol

IoT – Internet of Things

IP – Internet Protocol

JSON – JavaScript Object Notation

JWT – JSON Web Token

MAC – Media Access Control

NoSQL – Not only SQL

NPM – Node Package Manager

PIR – Pyroelectric InfraRed

REST – Representational state transfer

RTSP – Real Time Streaming Protocol

SDK – Software development kit

URL – Uniform Resource Locator

XML – Extensible Markup Language

List of Figures

1 Introduction

Many companies have stand-alone offices in different parts of the world. Good communication and collaboration between the employees are crucial for a company's success. Cooperation is the building block of productive and efficient teamwork opportunities. There are several tools, like Microsoft Teams and Skype, that enable remote communication and collaboration. However, making use of Internet of Things (IoT) solutions, collaboration tools can bring even more possibilities for cooperation.

IoT has become a hot topic during the past years. There are millions of devices that collect different types of data from the physical world and these gadgets transfer data to applications and services all over the network [1, 2]. This enables building complex systems that are aware of their surroundings. For example, IoT is improving automated systems like assembly lines. Modern assembly lines are equipped with multiple sensors and actuators, that can sense and reposition physical objects on assembly lines [3]. This avoids possible jamming and damaging of assembly line machine tools [3]. In addition, IoT devices are used more and more, from surveillance systems to smart homes [2]. For example, many households have lights that are controlled over the Internet [4]. Furthermore, there are smart refrigerators that can keep track of the products stored inside it and provide information about the consumption history and nutritional facts about the goods [5].

The development of IoT has new potentials in collaboration, by helping to create new and more efficient Computer Supported Cooperative Work (CSCW) systems [6]. CSCW is a research field which is devoted to building tools that improve collaboration between people. The most basic example of CSCW are applications that provide sending E-mails or hold video conferences [7]. These systems enable people from all over the world to communicate without being in the same room. As IoT improves over time, it creates new opportunities in the development of collaborative tools. For example, IP cameras can be used to broadcast the image of a video conference [8].

1.1 Problem statement and requirements

Many companies have multiple offices in different parts of the world. Employees from different offices often work together and have to communicate a lot. It is important that people create a collective community and are provided with opportunities to communicate with ease. Usually, the communication is mostly work-related, but it is beneficial for people to get to know each other and not become too distant. There are multiple applications that provide communications between employees. For example, Skype permits people to create video calls and chat, but IoT creates an opportunity to build an ecosystem that enables more features to boost communication and collaboration between employees.

A CSCW application has multiple development steps (described in Section 2.1). One of the first steps is to describe the technological requirements. The main requirements for the application developed during the thesis are the following:

1) A web environment which supports sharing posts between users;
2) A real time video solution able to stream video activated by movement detection;
3) A solution that registers people who are currently in the office.

The first requirement is the basis of the application. The web application enables users to log in with their user and see the posts created by the collective. Additionally, it interconnects other features. For example, when movement is detected in an office, the web application should be able to request a real time video stream from that office, similarly it should be able to show who are present in an office at current time.

1.2 Similar applications

The previous section mentioned that CSCW application development has multiple steps. One of the other steps is to find applications that already exist and satisfy the technological requirements. There were no applications found which features all the previously mentioned requirements, but there are applications that have some of these features. This section describes some of these applications and their respective features.

Skype

Skype is an application, which enables people to message, call with and without call. Multiple participants can join a video call and share their camera’s picture to others [9]. In addition, Skype can show people’s current status – if they are online, busy or away. However, this status cannot directly indicate if a person is at an office or not. Although, Skype does provide video streaming, it does not have a feature that supports creating posts nor does it have a way to register people who are present at an office.

Microsoft Teams

Microsoft Teams [10] is similar to Skype. It provides video calls and chatting, but it is especially useful for teams and companies that use Azure Active Directory (AAD) as their user management tool. This is because, if the employees already have AAD accounts, then they do not need to register another account for Microsoft Teams, but they can use their existing AAD account. Using their account, they can instantly start collaborating with other members of the company. Although Microsoft Teams is a solid system for collaboration, it does not provide functionality for creating posts and displaying present employees at an office.

Twitter

Twitter [11] is social networking service that provides users to create posts, share their pictures and ideas. Lately Twitter has implemented a live streaming feature, where a person can start a live stream and others can tune in and watch the stream. Twitter users can be private or public – meaning people have an option to share their posts with only their followers or with everybody. Therefore, Twitter does satisfy two of the requirements – it could be used to share posts and create a live stream, but it does not show who are present at an office. Additionally, it would be cumbersome for everybody in a company to create a private account, and only follow people within the company.

The previous applications could be used for video streaming, but they lack the opportunity to activate the streaming based on movement. Additionally, only Twitter enables sharing of posts. None of these applications featured a way to register present users in an office automatically. There are attendance softwares, which enables users to manually register their attendance, but not automatically. Combining different services of these applications would not satisfy the main requirements. To add, it would be efficient for a company to create their own collaboration tool, as it can be managed, refined, developed and broadened to the needs of the company in the future.

1.3 Goal

The goal of this thesis is to create a CSCW application, that could boost and improve collaboration within the company Iglu OÜ. The purpose of the application is to connect people from different offices. Iglu OÜ has two offices, one in Tartu and one in Tallinn. The application implements the main technological requirements mentioned in section 1.1.

The goals are defined as follows:

1) The application enables streaming picture from one office to another by using Internet Protocol (IP) cameras.

2) The application lets users post messages, notes, important information and enables people to share their emotions and new achievements with others.

3) The application provides a solution to register people who are present at an office, which is done using an IoT device, that is able to catch probe requests from employees mobile phones.

1.4 Outline

The thesis is structured as follows. Chapter 2 expands on the concept of CSCW, IoT and Wi-Fi probe requests. It gives an overview of the different technologies that are used to implement the application. These technologies include multiple third-party services, frontend and backend technologies, and hosting services. Chapter 3 introduces the application itself, gives an overview how the technologies and services were set up and used to reach the desired goal. It describes the configuration and implementation of third-party services, frontend technologies, backend technologies, cloud services, microcontroller logic and receiving of the IP camera stream. The chapter also describes how continuous integration and continuous deployment are set up, and how the application was tested. Chapter 4 concludes the work carried out throughout the thesis.

2 Tools and technologies

This section describes the concepts, technologies, services and which are used to create the final application.

2.1 Computer Supported Cooperative Work

Computer supported cooperative work (CSCW) is a research field under Computer Sciences [12]. CSCW's objective is to study how social interactions, design, development of technical systems could boost work efficiency and increase social interactions within a team or a community [12]. The core idea of CSCW is for different applications to enable and raise cooperation, collaboration and coordination between people by connected artefacts and design practices [13]. The field is best described by "*CSCW examines the possibilities and effects of technological support for humans involved in collaborative group communication and work processes*" [12]. CSCW focuses primarily on software developed platforms for end-to-end usage [7]. For example a CSCW application could be a management system where each colleague in a workplace enters their current working status and others would have that information available to them [7]. If one person collected this data and entered everybody's current working status, then it would not be considered a CSCW [7].

The next paragraph is based on the handbook "Taxonomy and Theory in Computer" [7]. When CSCW was first introduced, the primary objective was to enable enhanced communication and to this day the primary objective has not changed. The first important technological development was the possibility to write E-mails. This enabled people from around the world to send information to each other at any time and place. Other breakthroughs followed, like text messaging, instant messaging, video, voice, and text conferences. These tools enable better communication and are nowadays essential for different meetings and announcements. There are coordination tools like team meeting systems/schedulers, team calendars and workflow management systems. These are all technological software applications which are considered as CSCW.

The next two paragraphs are referenced from an article "A Classification Method for CSCW Systems" [14]. CSCW applications have been classified in different ways but most of them are defined by the Johansen's Time-Space matrix. This is one of the first classifications of CSCW tools, where the matrix displays the human-computer interactions according to time and space.

	Same time	Different Time
Same place	Face to face interaction	Asynchronous interaction
Different place	Synchronous distributed interaction	Asynchronous distributed interaction

Figure 1. *Johansen Time-Space matrix* [14]

Figure 1 displays the Johansen's Time-Space matrix. It categorizes CSCW applications in 4 different categories. These categories are defined by physical space (where the CSCW tool can be used) and time (when people can interact with the tool). Regular meetings require parties to be at the same place at the same time, therefore it is a face-to-face interaction. Video conferences, on the other hand, happen at the same time but in different places, therefore video conferences are synchronous distributed interactions. However, since systems have developed and become more and more capable there are applications that consist of multiple different CSCW functionalities and therefore some tools can be defined by multiple categories.

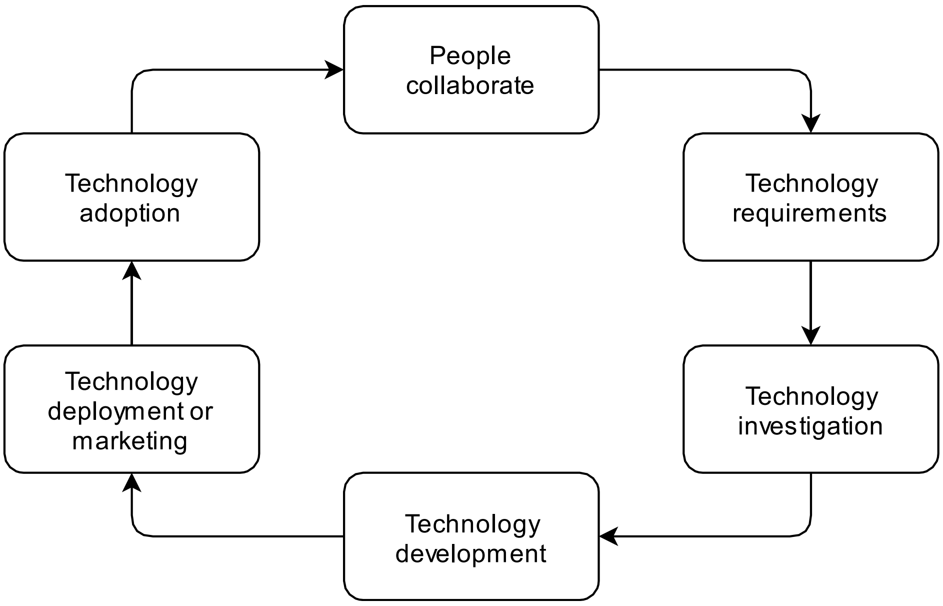


Figure 2. *A Model of collaboration and technology introduction* [7]

The next three paragraphs uses information from the handbook "Taxonomy and Theory in Computer" [7]. The underlying premise of CSCW is that people could use technology to collaborate and therefore work more effectively and pleasantly. In Figure 2 we have a model which displays how a software or CSCW application should be developed. The starting point is at the top of Figure 2 - people collaborating in workplaces. To boost the collaboration and work efficiency CSCW application might be needed. Following the diagram clockwise it is necessary to know the technological requirements of the application, that helps people collaborate. The next step “Technology investigation” is done in order to determine if similar applications exist and are available for use. If this is not the case, the process of making a new software has to be defined.

The fourth stage is the development of the software application. After the development phase, the application is either deployed or marketed. In case the CSCW application is meant for internal use in a company, the application is deployed and released for the employees to be used. If the application is not made for internal use, the application must be marketed and then adopted by others.

One of the problems that exist in CSCW applications is that the collaboration functionalities are not accessible from everyday applications that people use. The CSCW functionalities and data that it presents are isolated from the standard applications that people regularly operate with. For a successful CSCW application that has been deployed within a company, there is a need to integrate different functionalities of other internal applications. It is important for the seamless experience and to make CSCW features easily accessible.

The next sections in this chapter describes the technologies which are studied and used in the development of the application.

2.2 JavaScript libraries

JavaScript plays an irreplaceable role in web development [15]. It was initially meant to be used on the client side, however these days JavaScript can be used as a server-side programming language [15]. On the frontend there are multiple JavaScript tools that enable easier and smoother experience for the developer and for the end user [15]. There are tools like React and MobX that are used to program the behaviour and dynamic content of a webpage. Secondly there are libraries like MSAL.js and Media Stream Library JS that help implement third party services.

React

This paragraph is referenced from an article “The History of React.js on a Timeline” [16] and React’s official website [17]. React is modern JavaScript library developed by Facebook and it enables developers to create interactive and responsive web applications. It was launched in 2013 and it has been gaining popularity ever since. In 2016 when JavaScript library MobX was launched, React went globally mainstream. At its core, React consists of React components that enables developers to create complex application user interfaces from small and reusable parts. React components can have internal managed states, which are used to compose complex user interfaces. Component states are JavaScript objects, which influences the output of the render. Upon state changes, the component renders the view again with new values. Additionally, components can take states as input, and if a state is overwritten, the component re-renders the changed values.

MobX

MobX [18] is a JavaScript library that can be used together with React. It is a state management library that provides storing and updating of states. React and MobX provide a powerful combination of application state management and rendering of state changes. MobX is very scalable and one of the simplest, least intrusive state management tools for React.

MSAL.js

MSAL.js is a JavaScript library which provides authentication to users who use AAD, Microsoft personal accounts and other identity providers like Facebook, Google and LinkedIn [19]. MSAL.js is a client-side library, which can be used in web applications. MSAL provides features like logging in through redirect or popup, retrieving, refreshing and caching access tokens [20].

Media Stream Library JS

Media Stream Library JS [21] is an open source JavaScript library, which is developed and maintained by Axis Communications. Its objective is to transform media streams to work directly in client browser. This library enables frontend applications to create RTSP (Real Time Streaming Protocol) over WebSocket connection to the camera’s server. This functionality provides rendering of live streams directly in the browser.

Firebase JS SDK

The Firebase JavaScript SDK [22] is an implementation of client-side libraries which is used by applications that need to use Firebase services. Firebase JS SDK simplifies the use of Firebase services like Analytics, Authentication, Firestore, Realtime Database, Storage and many more.

2.3 Firebase services

This section describes tools and services that are used for this project, which are provided by Google's firebase.

Cloud Firestore

Cloud Firestore is a client-accessible Not only SQL (NoSQL) database. It is a cloud service developed and maintained by Google. Firestore can be used in different types of applications – it can be used in backend applications, Android and iOS applications and even web applications. It enables fast, serverless, cloud-native document-based database [23]. These documents are stored in collections, which are like folders that help organize and query data more efficiently [23]. Documents support different data types, like simple strings and integers, but also nested objects and arrays. Firestore is easy to setup and it simplifies storing and querying of data [23]. The main characteristics of Firestore are the following [24]:

1. **Flexibility** – Firestore supports flexible and hierarchical data structures. Data is organized in documents, collections and subcollections;
2. **Expressive querying** – Firestore supports querying specific or all documents in a collection. Documents can be queried by query parameters, which include filtering, combined filters and sorting;
3. **Realtime updates** – Cloud Firestore synchronizes data in an application in real time. Applications do not have to query the database whenever they need data, because Firebase supports data change listeners, which retrieve data whenever a change happened in the database;
4. **Offline support** – FireStore can cache data, which enables users to write, read and listen to data changes even when offline. Once the application is reconnected to the Internet, all the changes are synchronized;
5. **Designed to scale** – Google claims that Firebase can handle extreme workloads and huge amounts of data.

Firebase Realtime Database

Firebase Realtime Database is a NoSQL cloud-based service just like Firestore [25]. Firebase Realtime Database holds a JSON tree instead of collection and document hierarchia like in Firestore [25]. Realtime database has extremely low latency and is an ideal solution for frequent state-changing data within the database [25]. Otherwise Realtime Database is very similar to Firestore, but it does not scale as well as Firestore does and querying data is less organized [26].

Firebase Storage

Firebase Storage [27] is another service that Google provides for Firebase projects. Firebase storage lets users upload and download files. The files are held in Google Storage Bucket and Google Cloud. Firebase storage is highly scalable, supports robust operations and has security rules, which can be defined by a developer.

Firebase Authentication

This paragraph is based on Firebase Authentication's official documentation [28]. In addition to the previously mentioned services, Firebase supports authentication out of the box. Google has developed Firebase Authentication because most applications need to know user identity to provide personalized user experience and personal database queries and updates. Firebase Authentication is integrated with other Firebase services. Firebase Authentication provides different authentication solutions. It provides E-mail and password-based authentication, third party authentications like Google, Facebook, Apple, Twitter, GitHub, Microsoft, Yahoo, phone number authentications and lastly a developer can choose to create a custom authentication system. Authentication in a Firebase Authentication integrated application works by asking the user's credentials. The credentials could be an E-mail address and a password which belong to any third party federated identity providers mentioned above. The credentials are sent to the Firebase Authentication backend services which will verify the credentials and return a response. If the login was successful, the user can access or update the data in Firestore.

Cloud Firestore Security Rules

The next paragraph is using the information from Cloud Firestore Security Rules's documentation [29]. Firestore provides tools to secure the data in Firestore without any server-side authorization code. Firestore enables a developer to write rules which define permissions to data reading, writing, deleting and updating. Rules are defined in the Firebase Console. The rules can be set to selected documents or for the whole collection. If some documents should only be written by authenticated users but read by everybody, then the rules allow different permissions for reading and writing. Besides giving access to data reading and writing, rules are also used to define document field values. For example, a developer can define that a string field cannot have more than ten characters. If somebody tries to exceed the defined number of characters in this document field, then the action would be denied by Firestore. In conclusion Cloud Firestore Security Rules is the way to secure the data in Firestore and allow predefined value types in documents.

Firebase Hosting

Firebase Hosting [30] is another service maintained and developed by Google. Firebase hosting is a static web content hosting solution, which is meant for modern web applications. Hosting provides infrastructure and tools to conveniently deploy and manage web applications. In addition, it provides secure connections, fast deployments, code deliveries and one click rollbacks.

2.4 Cloud Run and Docker

This paragraph uses information from Googles's Cloud Run webpage [31]. Cloud Run is a service developed by Google. Cloud Run is a fully managed computing platform that can host Docker containers. The main features Cloud Run offers are deploying containers simply and quickly, supporting any programming language or libraries, and being able to automatically scale up and down to zero when there is no traffic. Most common uses of Cloud Run are hosting of dynamic web applications and RESTful APIs. Cloud Run leverages the simplicity of containers - applications can be built and run on Cloud Run in seconds after deployment.

Docker is an open-source tool developed by Docker Inc. It is a container virtualization technology, which in essence is a very lightweight Virtual Machine [32]. It makes the deployment of applications easy, fast and lightweight. Docker solves multiple problems from the developer and operations perspective [32]. Some of the problems it solves includes removing application dependency headaches, applications not running on different operation systems and decreases the time between writing the code and code being deployed [33].

Core Docker components and terms are Docker images, Container Registries and Docker containers [33].

Docker images contains the components which are needed to run an application as a container. These components might be configuration files, source code and run time software [34]. Images are built using a series of commands that can define which operating system the container should use and which commands to apply when the application is ran in a container [33].

The next two paragraphs are based on "The Docker Book: Containerization Is the New Virtualization" [33] and "Docker: lightweight Linux containers for consistent development and deployment" [35]. Docker registries are global servers which can hold Docker images. There are private and public registries, Docker Inc has their own registry called Docker Hub. Docker hub provides public and private registries. Public registries permit users to share and store their Docker images with everybody. Private registries are used to hold private software like source code which are not accessible by everyone. Public Docker registries hold images of different software, like database servers, content management systems, development environments, and web servers. Everybody with access can download these images and run them locally. Many corporations like Google and Amazon have their own registries and beside that, users can create their own registries.

Docker containers objective is to help build and deploy applications [33]. These containers are like packages over built applications. Docker containers are built from docker images, where each container contains software [33]. Containers can be created, started, stopped, restarted and destroyed [33]. The container can be run anywhere, on a local computer, in a Cloud Run server or Amazon EC2 hosts.

2.5 Azure Active Directory

This chapter is based on Azure Active Directory's official documentation [36]. AAD is a cloud-based identity service developed by Microsoft. It is an identity and access management service, which gives people in an organization ability to log in and access different resources provided by Microsoft. AAD provides access control to applications and different types of authentications. It also gives tools to create and protect user identities, credentials and provides a single sign-on authentication. Multiple applications can use the same authentication method, which means that users can log in to different applications with the same AAD credentials.

AAD trades user’s credentials for an access token. An access token is a JWT token issued by an Azure authorization service. The token usually contains information about the logged in user and user’s permissions. These tokens are personal and can be used to access different resources protected by AAD. Azure access tokens have an expiry date. If the current date is past the expiry date, the tokens cannot be used to access any resources. Access tokens are usually sent from the Azure authentication service with a refresh token. A refresh token is used to require a new access token from the authentication service in case the previous access token had expired. Refresh tokens have an expiry date as well, but usually they last about a month, whereas access tokens last one hour.

AAD sign-in can be used in web applications, backend applications, desktop and mobile applications. Figure 3 displays the sign-in flow to AAD using a frontend application.

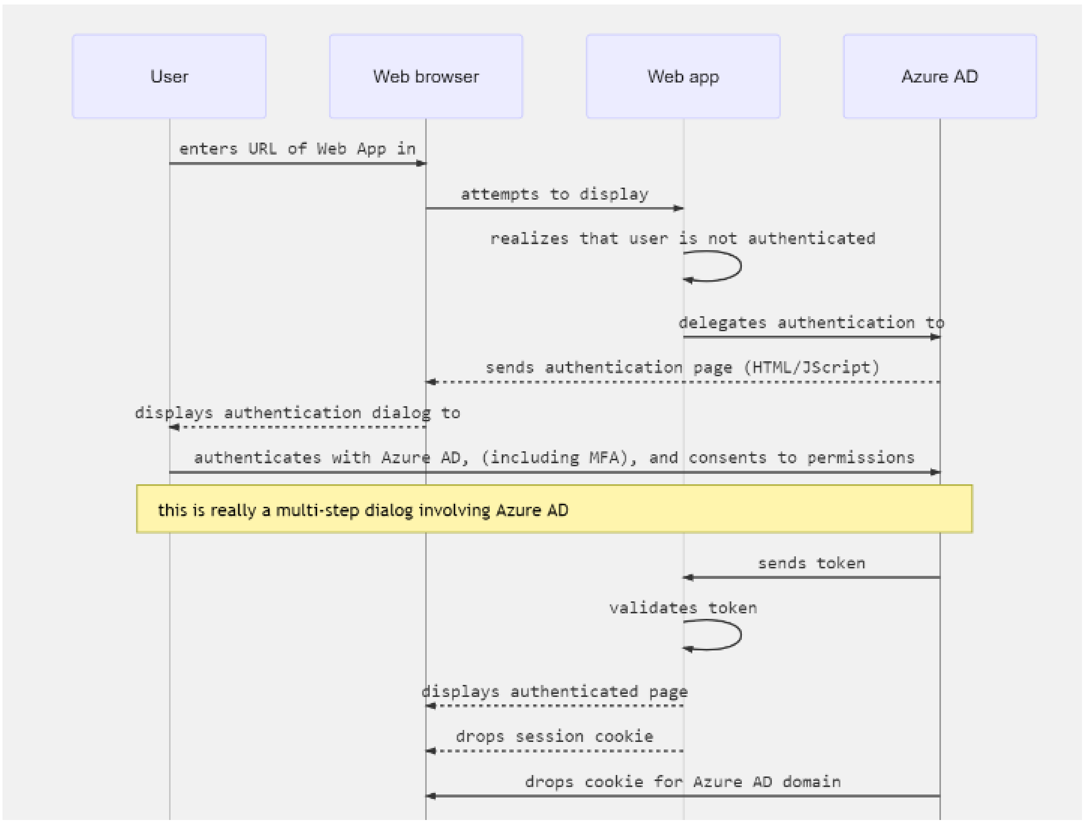

User
Web browser
Web app
Azure AD
enters URL of Web App in
attempts to display
realizes that user is not authenticated
delegates authentication to
sends authentication page (HTML/JScript)
displays authentication dialog to
authenticates with Azure AD, (including MFA), and consents to permissions
this is really a multi-step dialog involving Azure AD
sends token
validates token
displays authenticated page
drops session cookie
drops cookie for Azure AD domain

Figure 3. *AAD web application sign-in sequence diagram* [36]

The sequence diagram in Figure 3 starts when the user goes to the web page which needs AAD authentication. If the user is not logged in, then the user is redirected to Microsoft's login page to enter credentials. If the user enters correct credentials, then the Azure authentication service sends back an access token and a refresh token. Both are saved in the browser's cookies or local storage for further usage. The access token can be used to access AAD protected services.

2.6 Spring Boot and Spring Security

Spring Boot is a Java framework developed by Pivotal. Spring Boot provides an environment to develop backend services [37]. It is usually used to create server-side micro services or APIs. Spring Boot applications come preconfigured out of the box, meaning that Spring Boot applications are easy to setup [37]. Spring boot offers embedded web servers, simplified dependencies for build configurations, automatic configuring of third-party libraries, metric and health checks and minimal XML configurations [37].

Spring Security is a Java framework that works well with other Spring applications. It is a highly customizable authentication and access-control framework [38]. It provides authentication and authorization to Java applications [38]. In 2018 AAD was integrated into Spring Security version 5.0 [39]. Since then Spring Security provides authentication and authorization to Spring applications with some configuration. Spring Security can be set up to authenticate application's API endpoints. When a frontend application wants to access resources from a Java application that has AAD and Spring Security set up, then an access token has to be provided [39]. Spring Security checks and authenticates the validity of the token. Developer can use Spring Security to add role-based authentication to some endpoints, which means some AAD users can access the endpoint while others may not [39].

2.7 Internet of things

IoT devices are interconnected smart devices that are equipped with ubiquitous intelligence. They are everyday objects embedded with different sensors, actuators and they use wired or wireless connection to the Internet. These objects could be cameras, lights and even smart refrigerators (that keep track of the expiry date of groceries). IoT's purpose is to provide an infrastructure that could create a bridge between the physical world and information systems [2]. IoT has been defined by [40] as *"A world where physical objects are seamlessly inte-*

grated into the information network, and where the physical objects can become active participants in business processes. Services are available to interact with these "smart objects" over the Internet, query their state and any information associated with them, taking into account security and privacy issues." These objects are integrated into an information network and they also have identities, physical attributes and virtual personalities [2]. What makes smart devices so powerful, is that they can collect enormous amounts of data from the real world, communicate and forward it [3].

This paragraph is referenced from book „Internet of Things - Global Technological and Societal Trends" [41]. The IoT objects are becoming key components of businesses and social processes. They use different sensors to collect information from the environment and exchange this data with an application over the Internet or with other IoT devices through network communication. Besides just sending data about their environment, IoT devices can be made to react autonomously to real world events in their environment and then trigger some service or send the data to the Internet. Services can run functions based on the information retrieved from the IoT device. For example, if a movement sensor senses movement, it can send this information using Internet connection to another application over the network. The application reacts to the received data and can turn on a camera to record the movement. This can be done without any human interaction or interference. These independent Internet services and applications are characterized by data analysis, event transfers and interoperability, and being able to receive all kinds of different information from different IoT devices.

2.7.1 IP camera - Axis-1065-LW

The following two paragraphs are based on article "Cost effective IP camera for video surveillance" [42]. IP cameras can be classified as IoT. IP cameras are a combination of simple cameras and network video technology. They are web servers that also have a sensor which can collect data from the physical world. Their primary purpose is to collect and compress live videos and send it to the Internet without any additional computer or microcontroller. IP cameras are mostly used as surveillance devices, because their picture can be viewed remotely and their activity needs no human intervention.

IP camera's most important features are encoding speeds, video resolution, frame bitrates, distortion and power dissipation. For a regular user the relevant features are video resolution, compression rate and frame rate, as these are the characteristics that mostly influence bandwidth and video quality. MJPEG and MPEG4 are standard encoders inside most cameras used for video compression. One of the newest types is H.264 encoder, which is more efficient than MJPEG for video coding and transferring, but it increases the cost of image compression complexity, needs more processing power, more memory and therefore also draws more power.

Axis-1065-LW is an IP camera produced by Axis Communications, which is capable in many ways. It can create a wireless connection to an access point and host its own web server [43]. The web server has its own browser user interface and it supports different API's. There are API-s which can provide information about the camera, trigger different camera's functions, control different streaming properties, and provide a token, which can be used to access the cameras live stream [44]. The camera supports many features and services, but most importantly it supports MJPEG and H.264 encoding, can stream at 1920·1080 resolution at 25 frames per second, has a 110 degree horizontal field of view, supports a microphone, and a small speaker [43].

When it comes to streaming, the camera supports streaming over HTTP with the same web server it hosts but it also includes a media server for RTSP streaming [44].

Streaming over HTTP

Axis's official API documentation [44] states that HTTP streaming provides snapshot images and multipart images which are provided by the web server built into the camera. Snapshot images are simple pictures in JPEG format. The picture is taken and returned to the client when the corresponding API call is made. Motion JPEG video API returns a continuous flow of JPEG images.

Streaming over RTSP

This paragraph is referenced from official documentation of Real Time Streaming Protocol 2.0 [45]. RTSP is an application-level protocol which enables control over real time streaming of data. RTSP can be thought of as a network remote control for a media server [45]. This protocol is meant to be run between a client and the server, where the client can request multimedia data [45]. RTSP is a bi-directional request and response protocol that is used to access multi-media content. The official RTSP documentation [45] mentions that the protocol tries to stay agnostic about the transferred media type and delivery protocol.

Axis-1065-LW supports RTSP over WebSocket connection [44]. The WebSocket protocol takes care of the handshake between the client and the server and is responsible for data transfer between the parties [46].

2.7.2 ESP-32

The next paragprah is referenced from Espressif documentation [47]. ESP-WROOM-32 is a microcontroller chip which is produced by Espressif Systems. Although, it is a low-cost development kit board, it provides a lot of features. It supports Wi-Fi, Bluetooth, legacy Bluetooth connections, and Bluetooth Low Energy which means that it can communicate with a lot of devices and applications. Wi-Fi enables communication through the router and Bluetooth can be used to connect using smartphones. Additionally, ESP-32 supports different sensors and other peripherals from touch and movement sensors to secure digital card support and Ethernet Internet connection.

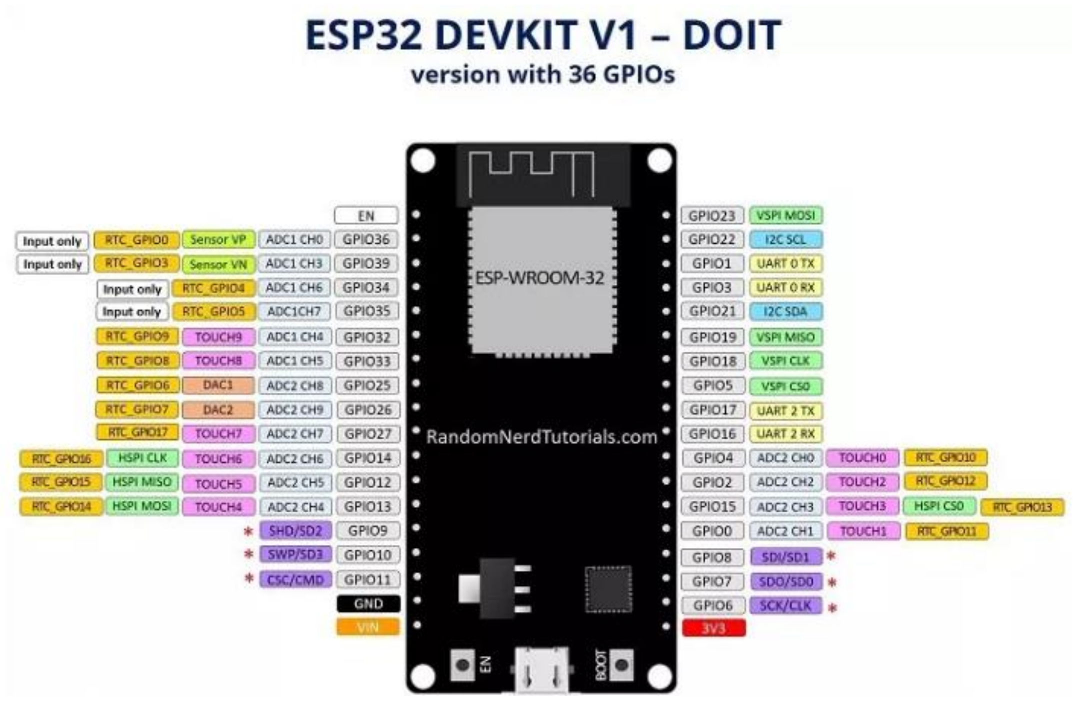

ESP32 DEVKIT V1 – DOIT
version with 36 GPIOs
ESP-WROOM-32
RandomNerdTutorials.com
EN
Input only RTC_GPIO0 Sensor VP ADC1 CH0 GPIO36
Input only RTC_GPIO3 Sensor VN ADC1 CH3 GPIO39
Input only RTC_GPIO4 ADC1 CH6 GPIO34
Input only RTC_GPIO5 ADC1CH7 GPIO35
RTC_GPIO9 TOUCH9 ADC1 CH4 GPIO32
RTC_GPIO8 TOUCH8 ADC1 CH5 GPIO33
RTC_GPIO6 DAC1 ADC2 CH8 GPIO25
RTC_GPIO7 DAC2 ADC2 CH9 GPIO26
RTC_GPIO17 TOUCH7 ADC2 CH7 GPIO27
RTC_GPIO16 HSPI CLK TOUCH6 ADC2 CH6 GPIO14
RTC_GPIO15 HSPI MISO TOUCH5 ADC2 CH5 GPIO12
RTC_GPIO14 HSPI MOSI TOUCH4 ADC2 CH4 GPIO13
* SHD/SD2 GPIO9
* SWP/SD3 GPIO10
* CSC/CMD GPIO11
GND
VIN
GPIO23 VSPI MOSI
GPIO22 I2C SCL
GPIO1 UART 0 TX
GPIO3 UART 0 RX
GPIO21 I2C SDA
GPIO19 VSPI MISO
GPIO18 VSPI CLK
GPIO5 VSPI CS0
GPIO17 UART 2 TX
GPIO16 UART 2 RX
GPIO4 ADC2 CH0 TOUCH0 RTC_GPIO10
GPIO2 ADC2 CH2 TOUCH2 RTC_GPIO12
GPIO15 ADC2 CH3 TOUCH3 HSPI CS0 RTC_GPIO13
GPIO0 ADC2 CH1 TOUCH1 RTC_GPIO11
GPIO8 SDI/SD1 *
GPIO7 SDO/SD0 *
GPIO6 SCK/CLK *
3V3
EN
BOOT

Figure 4. *Pin layout of ESP-WROOM-32* [48]

Figure 4 shows the layout of the ESP-WROOM-32. The microcontroller has 36 different pins, where some pins have specific features and others multiple features [48]. These include pins which can sense electrical charges like human skin and pins for converting analog signals to digital and the other way around [48]. For example, a peripheral like a Pyroelectric InfraRed (PIR) motion sensor can be connected to the analog to digital pin.

This paragraph is referenced from Adafruits study material about PIR [49]. PIR Sensor can sense motion and its main objective is to sense if a human has passed through the sensors range. PIR motion sensors detect infrared radiation. Usually motion sensors have 3 pins – power, ground and a digital output. If PIR senses infrared radiation change within its range, then this information is sent to the digital output. PIR motion sensors have adjustable controls that enable changing the sensors sensitivity and the time delay of when the sensor is active.

There are different ways to access the data, that ESP-32 collects. One of them is through Firestore database. Firebase ESP-32 is an Arduino Library for ESP-32 microcontrollers [50]. It features a set of functions for ESP-32 to communicate with Firestore. These functions include reading, storing, updating and deleting data from the database [50]. Besides that, it supports file streaming, reading and writing database rules, creating timestamps, receiving stream event callbacks, creating data backups, build Firebase cloud messages and finally it supports parsing and building of JSON structures [50]. This library can be used to support data flow between the microcontroller and a cloud database. For example, when ESP-32's peripheral PIR sensor registers movement, the microcontroller can send this data automatically to Firestore.

2.8 Wi-Fi Probe request

This paragraph is based on "Implementing 802.11 probe request scanner using WARP platform" and "Mobile Device Detection Through WiFi" articles [51, 52]. IEEE 802.11, also referred to as Wi-Fi is surrounding us everywhere. Devices like smartphones and laptops, that have a Wi-Fi interfaces periodically scan nearby access points. They do this in order to find wireless networks that are available for connection. It does not matter if the smartphone is already connected to an access point, the scanning is always done whenever the Wi-Fi interface is enabled. During the scanning procedure the device sends out probe requests (MAC frames). The nearby access points respond to the probe request sender with information about the access point. Even if the device is already connected to an access point, it still continuously sends out probe requests to look for networks with better signal.

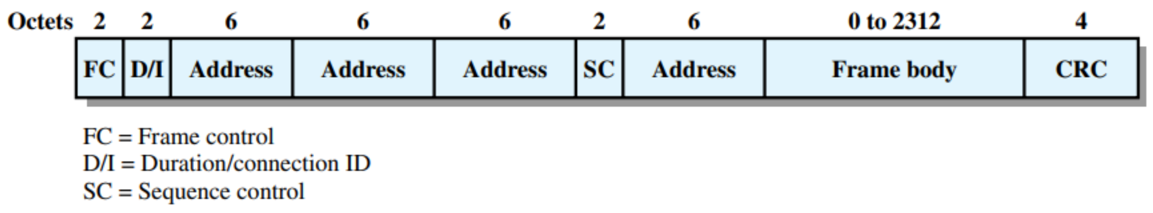


Figure 5. *IEEE 802.11 MAC Frame Format* [53]

Figure 5 shows the format of probe requests that Wi-Fi interfaces send continuously. MAC frame format consists of the following fields as also displayed in Figure 5 from left to right:

Frame Control – Shows the frame type (control, management or data);

Duration/Connection ID – Indicates the time the channel is allocated for a successful transmission;

Address 1 – Receiver MAC address;

Address 2 – Transmitter MAC address;

Address 3 – Destination MAC address;

Sequence Control – Numbers the frames sent between the same receiver and transmitter;

Address 4 – Present only when data frames are transmitted between access points;

Frame Body – Payload of the frame, contains information from higher layers;

Frame Check Sequence – For checking the integrity of the frame [53].

There are three types of MAC frames: control frames, data frames and management frames. Control frames help the reliable delivery of data frames [53]. There are different types of data frames, but their main objective is to carry upper-level data from the source to the destination station [53]. Lastly, management frames are used to manage communication between access points and stations [53].

ESP-WROOM-32 has Wi-Fi monitoring mode called promiscuous mode which enables ESP-32 microcontroller to listen to probe requests [54].

2.9 Summary

Firstly, this chapter explained what is CSCW and how CSCW applications are categorized and built. Secondly, the chapter gave an overview of multiple modern frontend and backend technologies like React, MobX, Spring Boot, Msal.JS, which are used in the development of the application. Thirdly, the chapter introduced technologies/services like AAD for authorization, multiple Firebase services for hosting and data storing, and Cloud Run for backend hosting. Lastly, the chapter describes what is IoT, what are probe requests, and introduces two IoT devices (ESP-32, Axis 1065-LW) and their capabilities.

3 Application development

This chapter explains the project's idea, architecture, application integration, configuration and implementation processes.

3.1 Project idea

Iglu OÜ is a software development company with two offices, one in Tartu and one in Tallinn. The idea of this project is to create a system which consists of two microcontrollers, two IP cameras and a web application. One IP camera and one open microcontroller are positioned in Tartu office and the other IP camera and microcontroller in Tallinn office. The offices are equipped with smart TV-s which have web browser support.

The web application consists of two views. One of the views is meant to be used on the TV and the other is meant to be used through regular computers and smartphones. The TV-s use the web application to receive a live video feed from the opposing city. The microcontrollers scan the probe requests in order to register employees who are present in the office. For privacy purposes each employee can choose to insert their MAC address and it is not mandatory. In addition to the video feed, the TV view of the web application has a section that shows which employees are at the office and a section which shows the latest posts created by the users.

The other view is meant to be used by the employees themselves using their personal gadgets. This view supports creating posts, reading posts, and includes a section which shows who are present at the office. Additionally, it allows them to insert a MAC address of their device. The MAC address will be tracked by the microcontrollers.

3.2 Integration with existing system

Iglu OÜ has internal Spring Boot backend applications and Angular frontend applications. Angular creates API requests to query information from the backend application and displays it in the browser. Backend is connected to a PostgreSQL database which holds internal organisational information and data about employees of the company. The existing applications use AAD for user management and user authentication. The application built during the thesis uses the same Azure Authentication sign on functionalities. As mentioned in section 2.5, the identity provider enables multiple applications to login with the same userbase. The developed application uses the same database connection as the existing Spring Boot application. Otherwise the built application's frontend and backend are stand-alone and do not depend on existing applications. Additionally, Iglu has developed a very basic version of the same application in the past. Some design and style elements from the first version are used in this thesis project.

3.3 Architecture

The built application's architecture consists of frontend application, backend application, IP camera, ESP-32 microcontroller and third party services.

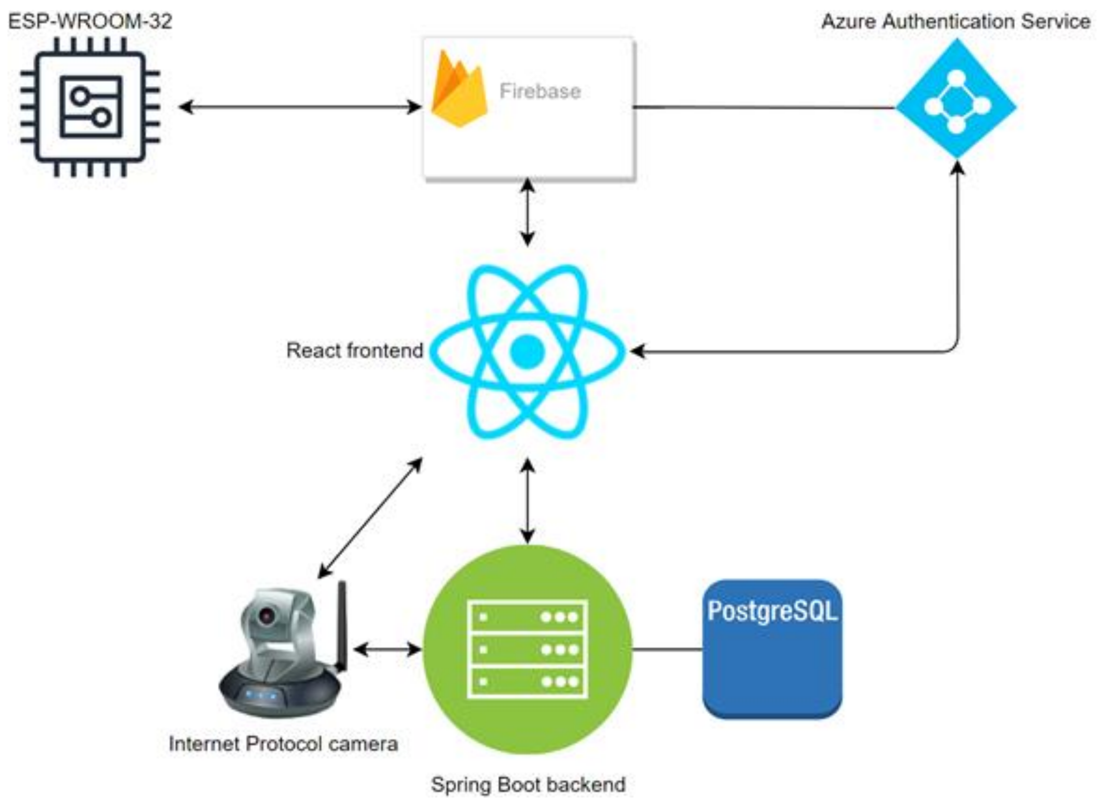


Figure 6. *Architecture of the application*

Figure 6 displays the modules and the architecture of the system. Backend is connected to PostgreSQL database where information about employees is held. Backend is also responsible for authorizing all requests made from the frontend, and can request for IP camera's WebSocket session tokens. That means only authenticated users have access to the camera's stream and database content. IP cameras host their own server and serve a real-time video stream. The frontend application asks for the WebSocket session token through the backend and upon receiving the token creates a RTSP over WebSocket session with the camera's media server.

In Figure 6 Firebase contains all the services that are mentioned in section 2.3, except for Firebase hosting. The frontend application uses Firestore, Realtime Database and storage to manage and query data. Firebase Authentication is used to authenticate users based on AAD. Authentication assures that only users with the company's AAD account can interact with Firebase's services. React application uses MSAL.js to authenticate and receive AAD user's access tokens. With these tokens frontend application can create REST requests to query data from the backend. ESP-WROOM-32 microcontroller does not connect to the Spring Boot backend, but updates and queries data directly to and from the Firebase Realtime Database.

3.4 Services Configuration

This section describes the configurations that were done to integrate all the services with the application.

AAD Configuration

In order to create secure authentications AAD needs Uniform Resource Locator-s (URL) of the applications. These are the URL-s that AAD is permitted to respond to. During the integration process four URL-s were added to the redirect URL list. URL-s "localhost:3000" and "localhost:3000/auth.html" were added in order to achieve authentication from application running locally on a computer. "Localhost:3000" is the URL of the local frontend application. "Localhost:3000/auth.html" is needed, because MSAL.js uses this URL to reauthenticate users silently inside an iframe. The frontend application is hosted in Firebase hosting. The last two URL-s are the hosted application's URL-s.

Cloud Run Configuration

Since Cloud Run runs containerized applications, there are no configurations to be done in terms of how the application should be built. Cloud Run needs some configuring in order to automatically deploy the application through a continuous deployment tool. A Google Cloud service worker has to be created so that the deployment process could access the docker registry and initiate a deploy to Cloud Run. The service worker were given permissions to read, create and deploy images. This service worker's credentials are used to create and deploy a docker image to Cloud Run in a continuous deployment process.

Firebase Configuration

One Firebase project is needed for the whole application. Databases, authentication, storage and hosting services can be created within this project. To access the authentication and database services from the frontend, a configuration snippet for Firebase connections is added to the frontend. The code has different properties and URL-s which refer to created Firebase services.

Firebase authentication can be set up from the Firebase console. For this project Microsoft is used as the sign-in provider. To allow Firebase to authenticate using AAD, the AAD client ID and secret must be provided to the Firebase console. In addition, authorised domains must be configured in the console. Authorised domains are the URL-s which have permissions to log in using the Firebase authentication. For this project authorised domains are "localhost:3000" and the URL of the deployed application. Firestore and Firebase Realtime Database configurations include creating the collections and rules which are described under section 3.7.

In order to initiate Firebase hosting, a JSON file needs to be placed at the root of the frontend application. The JSON file has configurations of which files should be deployed to the cloud service. The application can be deployed using Firebase CLI tool or through a continuous integration pipeline. The latter needs a Firebase token to authenticate the deployment.

3.5 Backend

Spring Initializer was used to create the Spring Boot Application, which is built using Gradle. The three main parts of the application are creating a database connection, creating REST endpoints and authorizing endpoints. Spring Security dependency and AAD dependency are added to authenticate all incoming requests. PostgreSQL dependency is needed in order to create a database connection.

To authorize all REST endpoints with Iglu OÜ AAD users, the instructions from Azure Spring Boot sample project were followed [55]. Firstly, Spring needs to know the client ID of the organization's AAD, which can be retrieved from the organization's Azure portal. Secondly, the application's authentication must be configured to be stateless. This is because backend service is not supposed to hold any user sessions, but rather only authorize requests.

To authorize all the endpoints, a web configuration class must be created, where authorization of all requests can be defined. In order to access backend endpoints, the requests must have a header with the Azure's access token.

3.6 Frontend

Frontend application was initialized using Node Package Manager (NPM). All dependencies within the frontend application are installed using NPM.

3.6.1 Dependencies

React and MobX frameworks are used as the basis of the frontend application. MobX takes care of the state management and React takes care of rendering components. Other JavaScript libraries were used to implement third party services.

MSAL.JS

MSAL.js and Firebase authentication can be used to login with AAD accounts. Both of them are able to receive access tokens upon logging in. With Firebase, the access token is only received after a successful login. AAD access tokens expire 1 hour after being created, and after that time, they cannot be used. If the access token expires, further API calls to the backend will not work. In order to receive another fresh access token through Firebase, users need to reauthenticate. MSAL.js is used to skip the reauthentication process. MSAL.js updates access tokens internally using refresh tokens that AAD provides. MSAL.js updates the access token in a hidden iframe. Every time a backend query is done, the access token is asked through MSAL.js. If reauthentication is needed, it does it silently without any redirects or popups.

Media Stream Library JS

Media Stream Library JS is used to create a RTSP over WebSocket connection with the Axis IP camera. The library is used to generate a stream playable by Hypertext Markup Language (HTML) Media Element. The connection is made with a couple of code lines.

```
const pipeline = new Html5VideoPipeline({
  ws: {uri: `ws://${host}/rtsp-over-websocket?rtspwssession=${to-
ken}`},
  rtsp: {uri: `rtsps://${host}/axis-media/media.amp?video-
codec=${encoding}`},
  mediaElement,
});
```

Figure 7. *Video request code snippet*

Figure 7 displays a code snippet on how the RTSP over WebSocket connection is made from the frontend to the IP camera. Media Stream Library JS provides a Html5VideoPipeline class, which takes 3 parameters – WebSocket URL, RTSP stream URL and the media element. Firstly, WebSocket URL is needed to send video data over a WebSocket connection. A query parameter can be specified to the WebSocket's URL. The parameter's value is a String token provided by the IP camera's web server. This token is used to create a connection to the camera's media server. RTSP URL is used to control the camera's media server. For example, through RTSP an action can be called, that starts playing the video. The media element parameter is an HTML document video tag. After the connection is made, a simple "play" call through the RTSP protocol triggers the video tag to play the stream.

Firebase JS SDK

Firebase JS SDK enables the developer to access Firebase services. The services, that the current project uses are Firestore, Authentication, Storage and Realtime database. After the user has been logged in with MSAL.js, the user is also authenticated with Firebase Authentication. After the user is authenticated, Firebase services can be used. The next paragraph gives an example how a Firebase service is used.

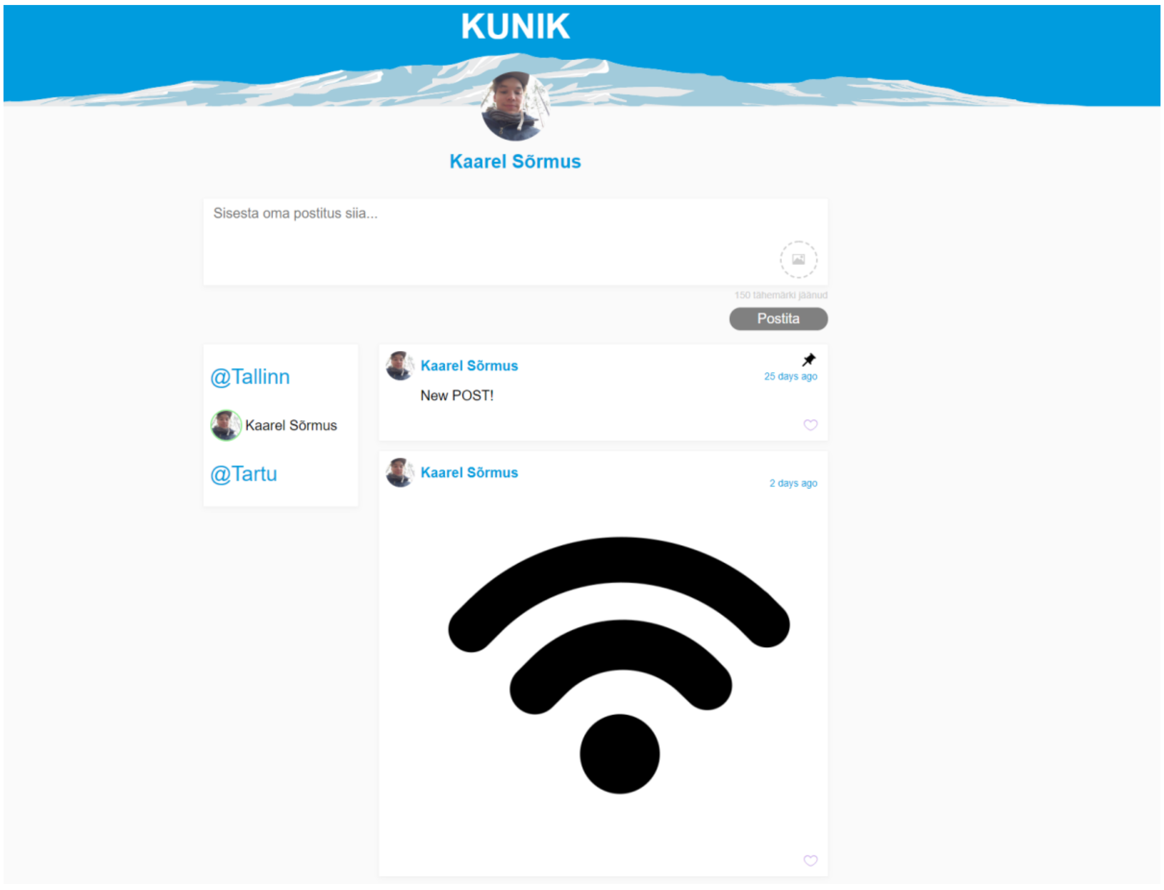
```
autorun(() => this.firebaseStore.postsCollection.orderBy('date',
'desc')
      .limit(this.queryLimit).onSnapshot(snapshot => {
        this.queryPosts(snapshot);
      }));
```

Figure 8. *Firestore collection query code snippet*

Figure 8 displays an example how data is queried from Firestore. On line 1, function “autorun“ is a MobX reaction, which is triggered every time its dependency changes [56]. The code refers to a collection called “posts”, orders the posts by date in descending order, and lastly limits the number of queried documents. Autorun is ran when “queryLimit” changes. For example, when we increase the limit, more posts are queried. The “onSnapShot” function enables real time updates from the “posts” collection. Every time a post in the Firebase is updated or created, the “onSnapshot” function is triggered. On row 3 a function is ran for each snapshot received, which saves the received data to a local state. Other Firebase services are used in a similar fashion with the latter example.

3.6.2 Views

The application has two main views, one where a logged in user can query and create posts. The other view is meant to be running on TV-s, which queries posts and displays the IP camera’s stream. This subsection introduces both of the views.

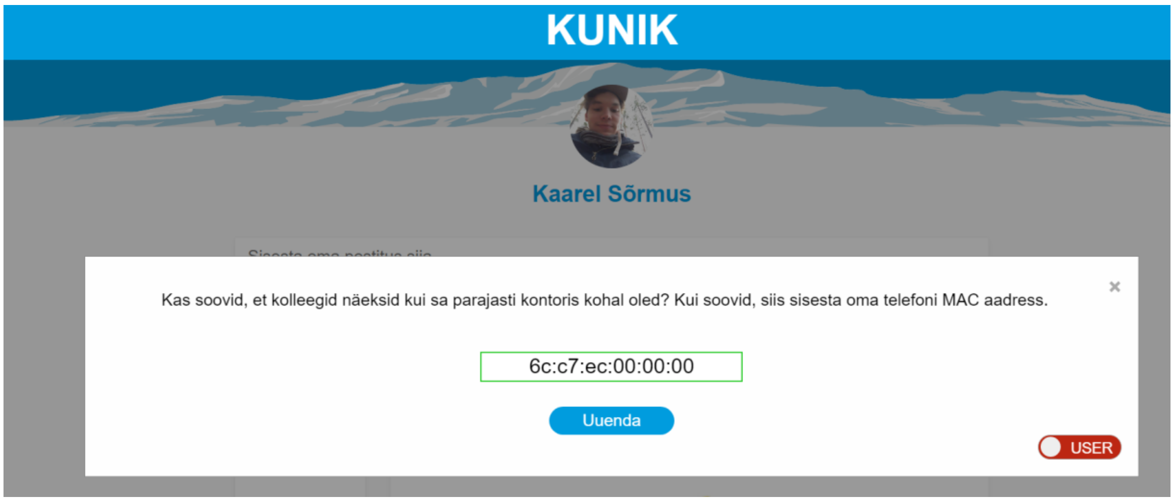


Kaarel Sõrmus
25 days ago
New POST!

Kaarel Sõrmus
2 days ago

Figure 9. *User view*

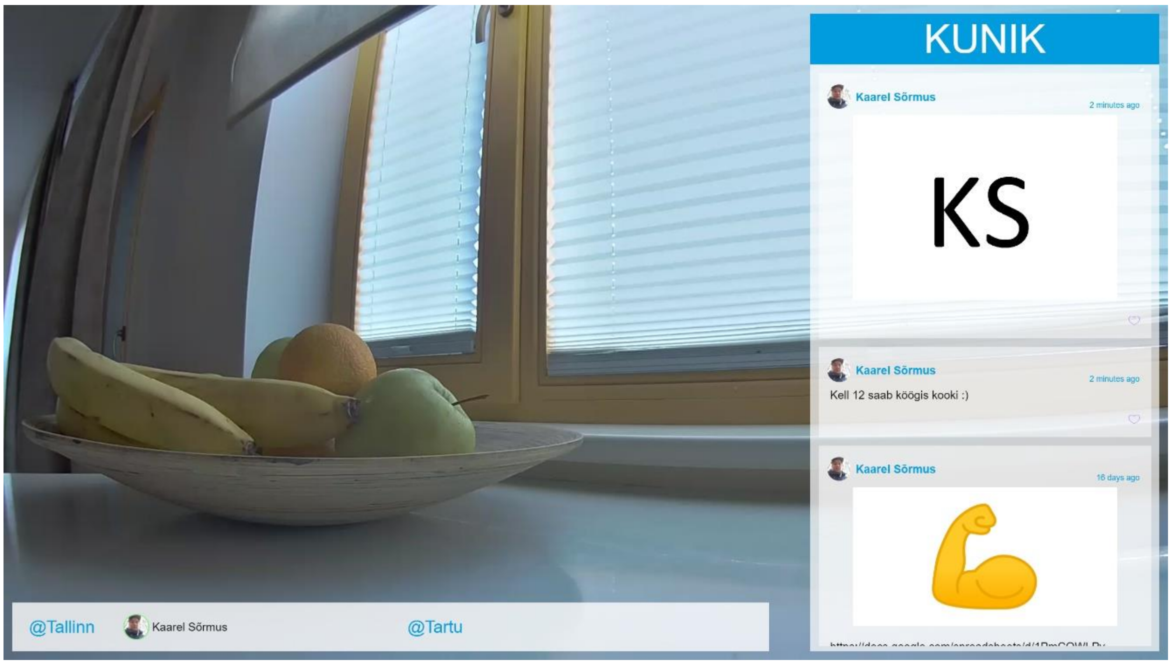
Figure 9 displays the page that the logged in user sees. It contains three sections: a section where a post can be written, a section which shows who is present at Tallinn’s Office and who in Tartu’s Office. Lastly there is a section where users can see posts and like them. Post writing is at the top of the page, under the profile picture. It supports writing posts up to 150 characters and uploading pictures. The section which shows who are present at the office is to the left of the page. There are two subsections “@Tallinn” and “@Tartu”. The subsections indicate the users who are present at the corresponding city. If the user’s MAC address was seen less than 10 minutes ago, the border around the profile is green, if seen over 10 and under 60 minutes ago, the border is yellow. When the user has not been seen over 60 and under 120 minutes ago, the border is red and when the user has not been seen for over two hours, the user will be removed from the list. The section containing posts, consists of two types of posts – pinned and non-pinned posts. As Figure 9 displays, the first post is pinned and is always on top of the post list.



Kas soovid, et kolleegid näeksid kui sa parajasti kontoris kohal oled? Kui soovid, siis sisesta oma telefoni MAC aadress.

Figure 10. *User MAC Address modal*

Figure 10 displays a modal which opens up when the user logs in for the first time or clicks on the profile picture. The modal has an input which asks for user’s phone MAC address. At the bottom right corner is a toggle which lets the user to change between user and admin features (the user needs to have admin rights in order to see this toggle). If the toggle is turned to „ADMIN“ position, the user can pin and unpin posts.



Kell 12 saab köögis kooki :)

Figure 11. *TV view*

Figure 11 displays the view that runs on the TVs in the offices. It contains of three sections. The background is the live stream picture from the opposing city's office. On the right is the posts section where all the latest posts can be seen. On the bottom left of the screen is a strip which contains two subsections "@Tallinn" and "@Tartu". Next to "@Tallinn", users who are present in Tallinn are shown. Same with Tartu. If the users who are present in the city do now fit in the given space (behind the city's name), then the user list will be scrolled to the left automatically. This lets the people see all the users who are present in the city.

3.7 Firebase services

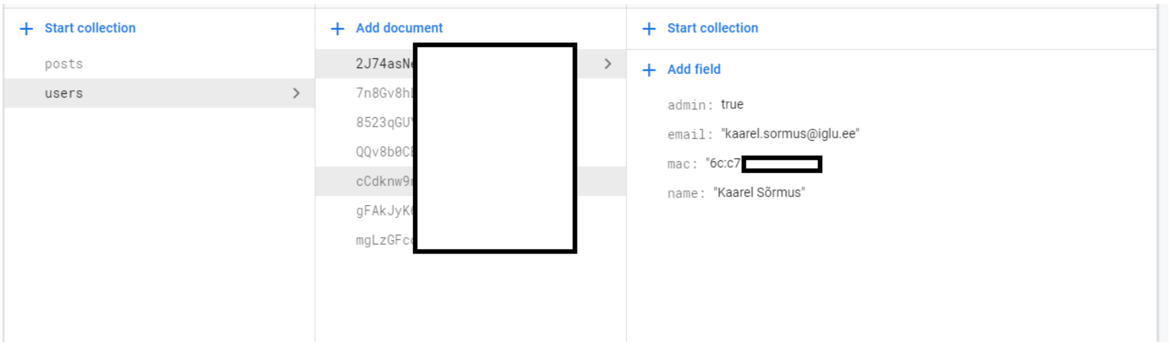
This section describes which Firebase services were used and gives an overview of specifics on what was done to achieve authentication and data storage.

Firebase authentication

Firebase's official documentation was followed to integrate Firebase authentication with Microsoft [57]. The configuration of Firebase authentication is described in section 3.4. Firebase authentication is invoked through the frontend application using Microsoft's login page (if the user is not already logged in into Microsoft Azure). If the user authenticates successfully, the user can access the Firebase data.

Firestore Data Model

In Firestore two collections were created: “users” and “posts”.


+ Start collection
posts
users

+ Add document
2J74asN [redacted]
7n8Gv8h
8523qGU
QQv8b0C
cCdknw9
gFAkJyK
mgLzGFc

+ Start collection
+ Add field
admin: true
email: "kaarel.sormus@iglu.ee"
mac: "6c:c7 [redacted]
name: "Kaarel Sõrmus"

Figure 12. *“Users“ collection*

Figure 12 displays the “users” collection. “Users” collection has documents for each user. Every document’s name is equal to user’s ID. “User” document has four values: “admin”, “email”, “mac”, and “name”. “Admin” field is a boolean value representing if user has admin features. “Email” holds the users E-mail address, “mac” value holds the MAC address of the user’s device, “name” is user’s display name.

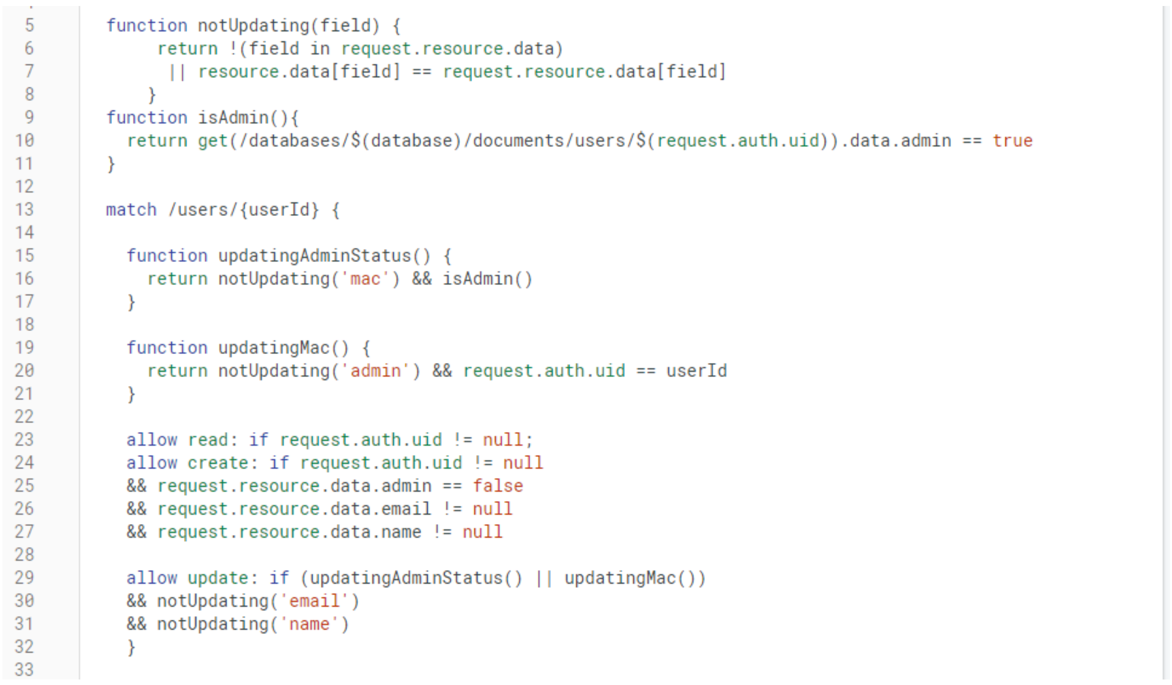
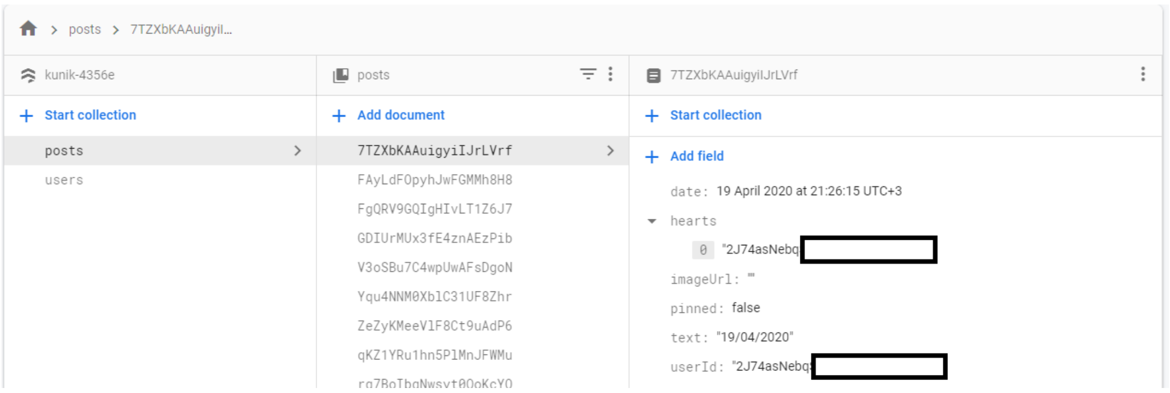

```
function notUpdating(field) {
     return !(field in request.resource.data)
      || resource.data[field] == request.resource.data[field]
    }
function isAdmin(){
  return get(/databases/$(database)/documents/users/$(request.auth.uid)).data.admin == true
}

match /users/{userId} {

  function updatingAdminStatus() {
    return notUpdating('mac') && isAdmin()
  }

  function updatingMac() {
    return notUpdating('admin') && request.auth.uid == userId
  }

  allow read: if request.auth.uid != null;
  allow create: if request.auth.uid != null
  && request.resource.data.admin == false
  && request.resource.data.email != null
  && request.resource.data.name != null

  allow update: if (updatingAdminStatus() || updatingMac())
  && notUpdating('email')
  && notUpdating('name')
  }
```

Figure 13. *“Users“ collection rules*

Figure 13 shows the rules applied to the “users“ collection. Firebase rules are written as simple predicates which must be true in order for the user to read or write to the database.

For example row 23 in Figure 13 defines the reading rule of all “users” collections. If the incoming request is authenticated, then the user may read the collection, therefore this field is allowed to be read by all users who are authenticated. Row 24 defines the rule for creating a document in “users” collection. The document can only be created when the user is authenticated. Additionally, the field “admin” must be false, and fields “email“ and “name” must not be empty. Row 29 defines updating the “user“ field. The “users” collection field “mac“ can only be updated by the user itself. The “admin“ field can only be updated by users who already have admin rights.



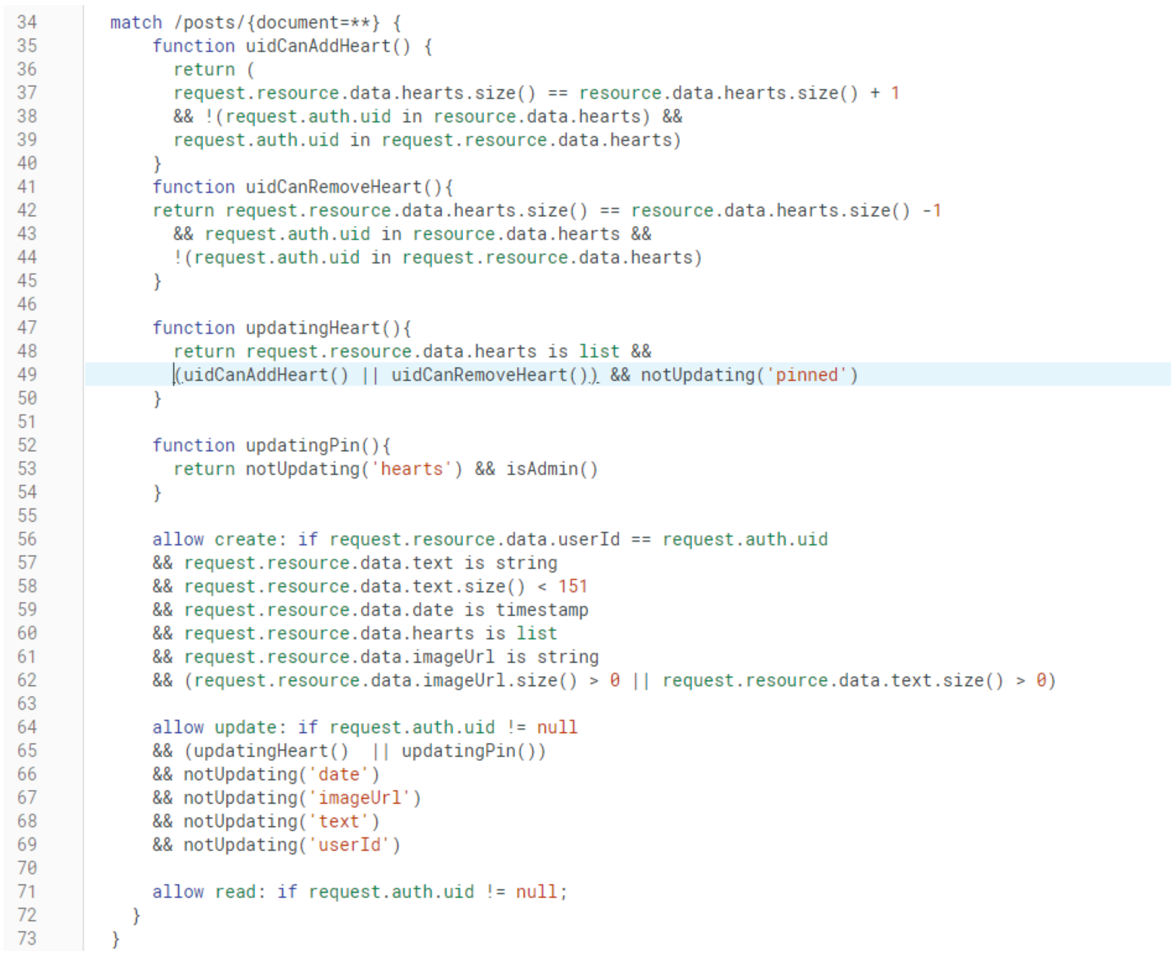
posts
users

7TZXbKAAuigyiIJrLVrf
FAyLdFOpyhJwFGMMh8H8
FgQRV9GQIgHIvLT1Z6J7
GDIUrMUx3fE4znAEzPib
V3oSBu7C4wpUwAFsDgoN
Yqu4NNM0Xb1C31UF8Zhr
ZeZyKMeeV1F8Ct9uAdP6
qKZ1YRu1hn5P1MnJFWMu
rq7BoTbqNwsvt0OoKcYO

date: 19 April 2020 at 21:26:15 UTC+3
hearts
0 "2J74asNebq[redacted]
imageUrl: ""
pinned: false
text: "19/04/2020"
userId: "2J74asNebq[redacted]

Figure 14. *“Posts“ collection*

Figure 14 displays the data model of “posts” collection. Documents have randomly generated names. Each post has six different variables: “Date”, which holds the value of the post creation time, “hearts” which is a string array and holds user ID-s of people who have liked the post. “Pinned” field is a boolean type field which represents if the post is pinned or not. “Text” field holds the post’s text, “imageUrl” holds the URL of the posted image. “UserId” field represents the value of the user’s ID who created the post.



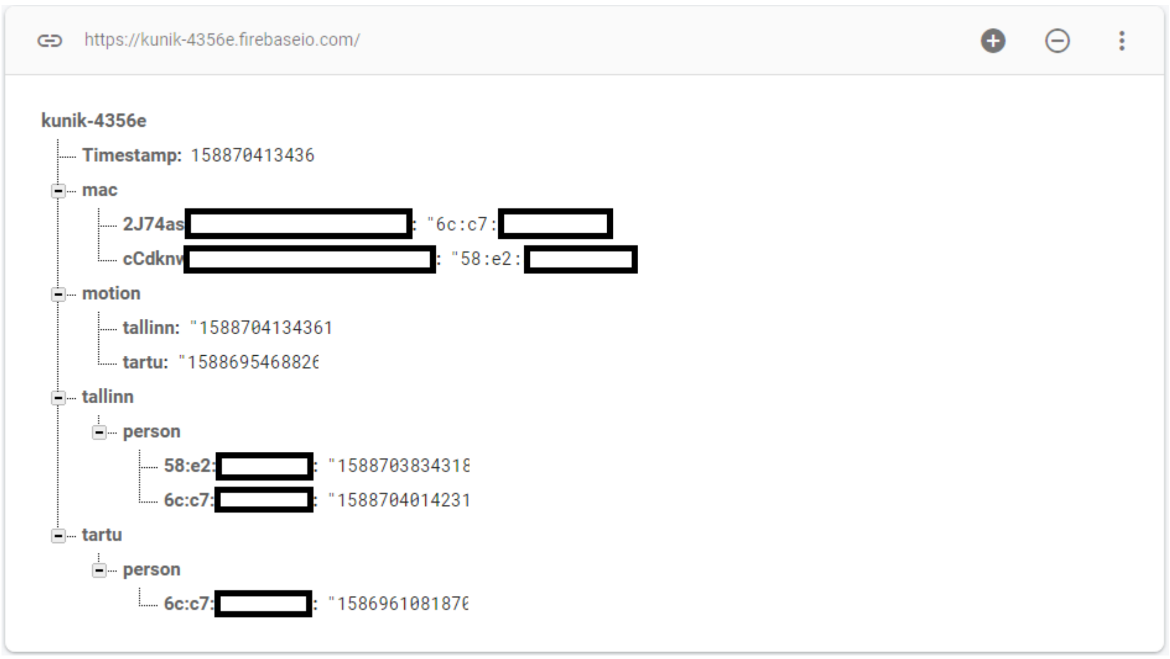
```
34      match /posts/{document=**} {
35          function uidCanAddHeart() {
36            return (
37            request.resource.data.hearts.size() == resource.data.hearts.size() + 1
38            && !(request.auth.uid in resource.data.hearts) &&
39            request.auth.uid in request.resource.data.hearts)
40          }
41          function uidCanRemoveHeart(){
42          return request.resource.data.hearts.size() == resource.data.hearts.size() -1
43            && request.auth.uid in resource.data.hearts &&
44            !(request.auth.uid in request.resource.data.hearts)
45          }
46
47          function updatingHeart(){
48            return request.resource.data.hearts is list &&
49            (uidCanAddHeart() || uidCanRemoveHeart()) && notUpdating('pinned')
50          }
51
52          function updatingPin(){
53            return notUpdating('hearts') && isAdmin()
54          }
55
56          allow create: if request.resource.data.userId == request.auth.uid
57          && request.resource.data.text is string
58          && request.resource.data.text.size() < 151
59          && request.resource.data.date is timestamp
60          && request.resource.data.hearts is list
61          && request.resource.data.imageUrl is string
62          && (request.resource.data.imageUrl.size() > 0 || request.resource.data.text.size() > 0)
63
64          allow update: if request.auth.uid != null
65          && (updatingHeart()  || updatingPin())
66          && notUpdating('date')
67          && notUpdating('imageUrl')
68          && notUpdating('text')
69          && notUpdating('userId')
70
71          allow read: if request.auth.uid != null;
72        }
73      }
```

Figure 15. *“Posts“ Collection Rules*

Figure 15 displays the rules for “posts” collection. Row 71 indicates that posts can be read by all authenticated users. Row 56 defines that creating a post is only allowed if the field’s ”userId“ value is equal to the authenticated user’s ID. On creation each post field is checked. For example, row 58 checks, that the post’s text is not longer than 150 characters. Updating a post is allowed in two cases. Firstly, if the user wants to change the „pinned“ field, then the user has to be admin. Secondly, updating the “hearts” list is allowed for all users as long as the user only removes or adds its own ID to/from the “hearts“ list.

Firebase Realtime Database Data Model

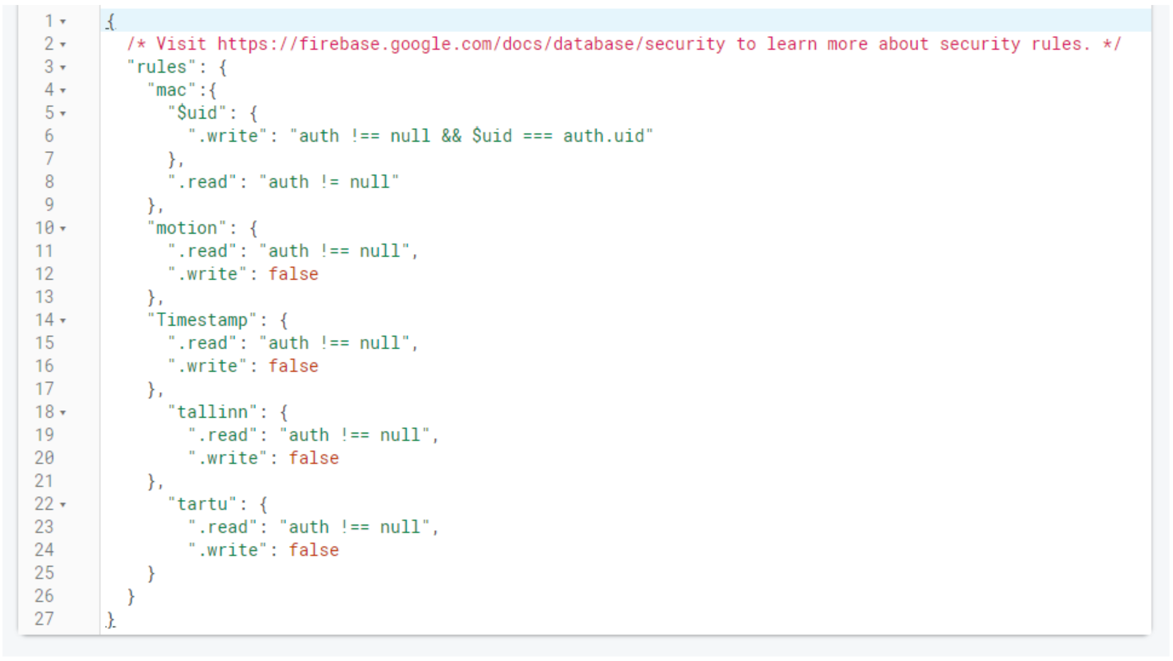
Firebase Realtime Database is used, so that ESP-32 could query and update information from cloud database.



```
https://kunik-4356e.firebaseio.com/

kunik-4356e
├── Timestamp: 158870413436
├── mac
│   ├── 2J74as[redacted]: "6c:c7:[redacted]
│   └── cCdknv[redacted]: "58:e2:[redacted]
├── motion
│   ├── tallinn: "1588704134361
│   └── tartu: "158869546882
├── tallinn
│   └── person
│       ├── 58:e2:[redacted]: "158870383431
│       └── 6c:c7:[redacted]: "1588704014231
└── tartu
    └── person
        └── 6c:c7:[redacted]: "158696108187
```

Figure 16. *Firebase Realtime Database JSON model*

Figure 16 displays which values the Firebase Realtime Database holds. “Timestamp” value is only used by the microcontroller. For microcontroller to know the server’s time, it needs to create a timestamp. When the timestamp is created, microcontroller can query it and have the value saved in the device’s memory. The timestamp is used to update “motion”, “tallinn” and “tartu” objects. “Mac” field holds a JSON key-value pairs where key is user’s ID and the value is the user’s MAC address. Microcontroller queries the “mac” JSON object to know which MAC addresses to follow. “Motion” holds two key-value pairs. Key is the name of the city and the value is a timestamp of when movement was last seen in the corresponding city. “Person” objects under “tartu” and “tallinn” are similar JSON objects. Both hold a list of key-value pairs. The keys are MAC addresses of seen devices and values are the timestamps of when the MAC address was registered. The purpose of these objects is to know when a specific MAC address was last registered in a corresponding city. Person objects can only have MAC addresses that are defined in the “mac” JSON object.



```
 1  {
 2    /* Visit https://firebase.google.com/docs/database/security to learn more about security rules. */
 3    "rules": {
 4      "mac":{
 5        "$uid": {
 6          ".write": "auth !== null && $uid === auth.uid"
 7        },
 8        ".read": "auth != null"
 9      },
10      "motion": {
11        ".read": "auth !== null",
12        ".write": false
13      },
14      "Timestamp": {
15        ".read": "auth !== null",
16        ".write": false
17      },
18        "tallinn": {
19          ".read": "auth !== null",
20          ".write": false
21      },
22        "tartu": {
23          ".read": "auth !== null",
24          ".write": false
25      }
26    }
27  }
```

Figure 17. *Realtime Firebase Rules*

Figure 17 displays the rules that cover who can read and write data from the Realtime Firebase. Row six defines that “Mac” JSON can only be written by authenticated users in case the key of the JSON object is equal to the user’s user ID. “Mac” JSON can be read by authenticated users. “Motion”, “Timestamp”, “tallinn” and “tartu” JSON fields can only be read by authenticated users. Writing is prohibited by everybody, except for the microcontroller. The microcontroller creates a connection to the database using the database’s secret key and therefore the rules do not apply to it.

3.8 ESP-WROOM-32

Arduino IDE is used to program the ESP-WROOM-32 microcontroller. For motion sensing ESP-32 is used with the PIR motion sensor, which are connected by the corresponding pins on the microcontroller and sensor. The code consists of two parts. The first part is to register movement and the second part is to capture probe requests. Microcontroller pushes received information to Firebase Realtime Database.

Motion sensing

The sensor's data pin is connected to the general-purpose input/output (GPIO) pin 23 on the microcontroller. Movement is checked in an endless loop. If the output of the movement sensor's digital pin is high, then the sensor has picked up movement. After motion is detected, a Firebase connection is made and the Firebase Realtime Database value in corresponding city is updated. The frontend application that is creating the live video stream listens to the movement values. If movement was detected, the timestamp is updated, and the frontend application starts the stream. If last movement was seen over 15 minutes ago, the frontend application stops the stream.

Promiscuous Mode

ESP-WROOM-32 provides a Wi-Fi packet monitoring mode called the promiscuous mode. To listen to probe requests, Wi-Fi connections must be turned off. During the implementation, code from an existing probe request monitoring project [58] was used to turn on promiscuous mode and to extract MAC addresses from the probe requests.

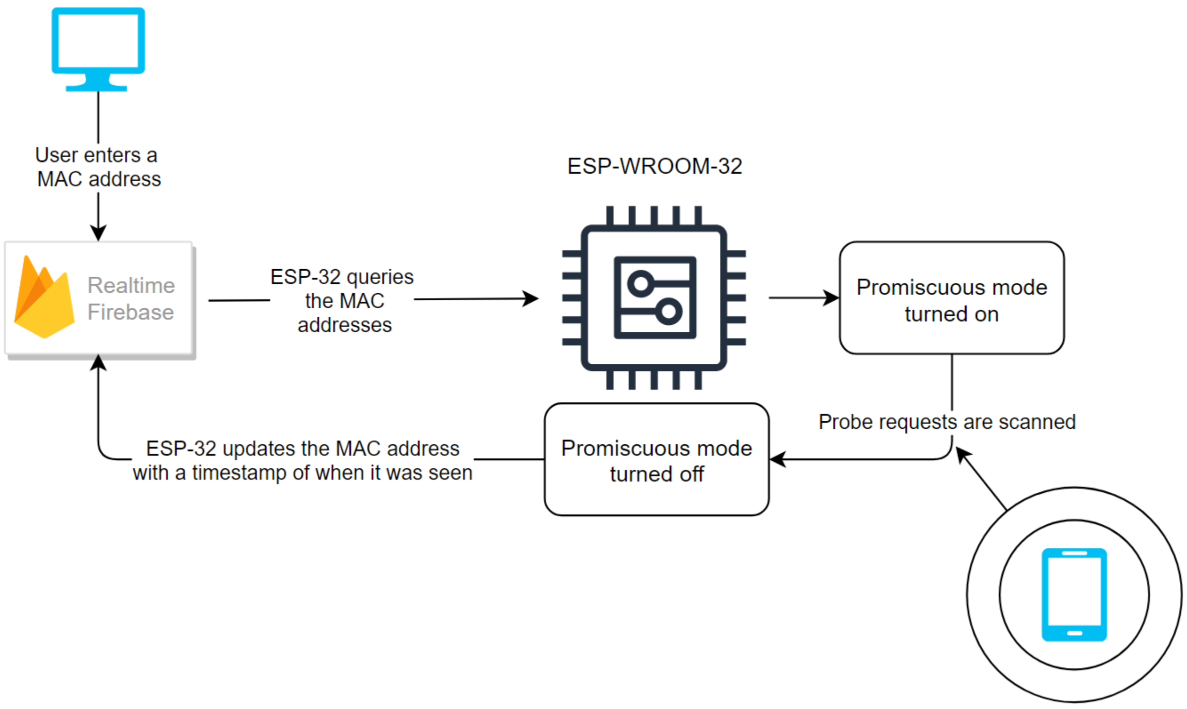

User enters a MAC address
Realtime Firebase
ESP-32 queries the MAC addresses
ESP-WROOM-32
Promiscuous mode turned on
Probe requests are scanned
Promiscuous mode turned off
ESP-32 updates the MAC address with a timestamp of when it was seen

Figure 18. *MAC address capture flow*

Figure 18 depicts the flow of how present users are registered in the vicinity of the ESP-32. The diagram starts at the top left, where the user enters his/her phone's MAC address into the system. Microcontroller queries all the MAC addresses from Firebase Realtime Database. After that, Wi-Fi connection is turned off and promiscuous mode is turned on. In this stage microcontroller starts listening to all the probe requests near it. Promiscuous mode allows to define a call-back function, which is triggered whenever a packet is caught. When this call-back function is called, the MAC address is extracted from the probe request and the code checks if found MAC was queried from the database. If a familiar MAC address was found, then it is certain that the person's phone is close. After a defined time, promiscuous mode is turned off and ESP-32 connects to an access point in order to create a Firebase Realtime Database connection. Firstly, the microcontroller creates a timestamp in the database in order to get the server's current time. Secondly, microcontroller adds the current timestamp to the MAC addresses that were in the vicinity and uploads them to the Firebase Realtime Database. If the microcontroller is in Tallinn, it updates the "person" object under "tallinn" object. After updating the "person" objects, the microcontroller queries the "mac" address list again to know which MAC addresses it needs to register. After that Wi-Fi is turned off and promiscuous mode is turned on. All the above is done in a loop, so ESP-32 always updates which MAC addresses to follow and updates the seen MAC (“person” object) addresses in Firebase Realtime Database. The frontend application always listens to these value changes and updates the views accordingly.

Promiscuous mode is turned on/off in intervals of 30 seconds. Since promiscuous mode cannot be running while the ESP-32 is connected to a network, the updates to the database are done with a slight delay. For example, when a movement is detected while promiscuous mode is turned on, the microcontroller does not create a network connection right away. It waits until the interval is over and then updates the found MAC addresses and movement timestamp in the database.

3.9 Axis 1065-LW

The Axis 1065-LW IP camera is used to receive the live stream picture from the camera’s web server. The camera’s stream and API-s are configured to be only accessible by authorized users. However, the RTSP over WebSocket stream can also be accessed with a camera generated token. How the camera’s stream is accessed by frontend application is as follows. When movement is detected in an office, then the Firebase Realtime Database is updated and the frontend application listens to these changes. When movement is detected less than 15 minutes ago, then the frontend queries for a stream. Firstly, the frontend creates a request to the Java backend application to receive the token. The Spring application creates a request with credentials to the camera’s web application. The camera responds with the token, sends it to the backend application, which forwards it to the frontend application. Frontend uses this token to access the camera’s live RTSP over WebSocket stream.

3.10 Continuous Integration and Continuous Deployment

As of now, Iglu OÜ internal applications are deployed to Linode. However, Iglu is in the process of migrating the existing applications from Linode to Cloud Run, that is why the created applications are deployed independently from the existing ones. The backend application is hosted in Cloud Run and frontend application in Firebase hosting.

The version control used for the project is Bitbucket. Bitbucket has a pipeline service called Bitbucket Pipeline which is easy to setup and configure. The pipeline allows to use different SDK-s and is capable of building docker images. Bitbucket pipeline process is initiated when a commit is merged or pushed to the master branch. The pipeline is configured to build the applications in order to check if the code compiles. After the code has compiled, the pipeline runs the tests in order to avoid application's undesirable behaviour. If the tests pass, the same pipeline is responsible for deploying the frontend and backend applications.

Backend is hosted in Google Cloud Run service. The pipeline builds the backend project using Gradle. The built application is in the "build" folder at the root of the project. Using the Dockerfile in the root of the project, a docker command within the pipeline builds a docker image and pushes the docker image to the Google Container Registry. After a successful push, the script logs into the Google Cloud Platform using Google Cloud SDK and deploys the container image from Container Registry to Cloud Run. In seconds the application is live and accessible.

The frontend is hosted using the Firebase hosting service. After the Bitbucket Pipeline has deployed the backend to Google Cloud, the frontend is built using NPM. The built static application files are generated into the "build" folder. The pipeline then uses the "firebase.json" file, at the root of the project to push the static files to Firebase hosting. This pipeline enables developers to fail fast or test and deploy applications to hosting services in a couple of minutes.

3.11 Testing

Due to critical circumstances, which occurred during the writing of this thesis, no user testing has been made. Currently the application is only tested with unit tests and usability has not been tested comprehensively. Only the author has tested the application in desktop and mobile devices. As mentioned in the previous section the project has continuous integration and continuous deployment integration, which tests the code automatically before it is deployed.

To be confident that the microcontroller works as expected, it was tested by letting it run continuously for two days. The reliability of the camera was tested by running the stream for five hours straight. Since it is an IP camera, which is supposed to be running 24/7, the author is confident that it can handle streaming to one client for multiple hours continuously.

3.12 Summary

Firstly, this chapter concludes how the application was merged with the existing systems, displays the architecture of the built system and briefly describes which configurations were done to use different services. Secondly, the chapter describes how the backend was built and how different frontend libraries were used. Thirdly, the chapter displays and explains the views of the application. Fourthly, the chapter describes how the Firebase data storing is done and how the rules for Firebase are written. Also it gives an overview on how the motion and probe request detection is done using the microcontroller, and how the frontend application receives the stream from the IP camera. Finally, the chapter describes how continuous integration and deployment were achieved, and how the application was tested. The application's code is pushed to Bitbucket. The secret codes and ID-s that were used to merge different services are removed from the code. URL of the repository is available in Appendix I. Repository.

4 Conclusion

Internet of Things (IoT) devices are surrounding us everywhere and have become a huge part in different technological solutions. IoT has introduced new development possibilities and is improving different types of software programs such as Computer Supported Cooperative Work (CSCW) applications. CSCW applications have come a long way to make collaboration easier and more efficient. In the beginning they enabled people to send E-mails, but as technology has developed collaborative tools are now allowing people to communicate and work together asynchronously from a distance.

The purpose of this thesis is to create a CSCW web application for a company. The application's purpose is to unify and draw employees from different offices together. The development of the software was done following CSCW development steps. At first, three main technical requirements were set. After setting the requirements, similar technologies were researched that supported some of those requirements. As none of the found technologies satisfied all the requirements, new CSCW application had to be developed. The built application implements all the set requirements. It enables a web platform where people can share their emotions and news. The application integrates a microcontroller, which scans for probe requests sent from nearby employees phones. This feature is used to register people who are present at an office. Internet Protocol (IP) camera is used to send a live video feed from one office to another. The web application consists of a frontend and backend application. Frontend is based on React and MobX JavaScript frameworks. In addition, multiple JavaScript libraries are used to implement third party services. For example, MSAL.js is used to authenticate Azure Active Directory (AAD) users and Firebase JS SDK is used to communicate with Firebase services like Firestore, Firebase Realtime Database and Firebase storage. Frontend is the basis of the application, it is responsible for authorization, connections with Firebase services and receiving of the live stream. The backend is built using Spring Boot.

The thesis gives an overview of multiple modern frontend technologies, backend technologies, and cloud services which were used in the implementation and hosting of the application. In the practical section the thesis gives an insight on how the application was built, how different services were configured, and describes how the application can quickly be deployed from development to production.

The application could be improved from multiple viewpoints in the future. Firstly, it is very important that the application usability is thoroughly tested and feedback is collected. Using the feedback the application and its features can be improved from the viewpoint of the userbase. Secondly, the features of the application could be broadened. For example, posts could have a comment section and there could be a possibility to create polls in the form of a post. Thirdly, the microcontroller code could be managed remotely. For example, the interval time of promiscuous mode could be changed through the frontend by an admin user. The microcontroller creates Firebase connections in intervals anyway, therefore it could update some variables used in its code. Fourthly, from the viewpoint of CSCW, the application could have a possibility to hold video conferences internally within the company. There are many video conferencing tools, but as mentioned in section 2.1, it is important for collaboration to happen within the same ecosystem. Additionally, as the video conferences would not depend on third party applications, the company could manage and fine tune the video conferencing possibilities.

5 Bibliography

[1] State of the IoT 2018: Number of IoT devices now at 7B – Market accelerating. https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/ (14.05.2020)

[2] Weber R. H., Weber R. Internet of Things Legal Perspectives, Berlin: Springer-Verlag, 2010.

[3] Chui M., Löffler M., Roberts R.. The Internet of Things. *McKinsey Quarterly*, 2010, 2.

[4] Smart lights 'to be most popular IoT device in the next decade' – Philips. https://internetofbusiness.com/smart-lights-popular-iot-device/ (14.05.2020)

[5] Osisanwo F., Kuyoro S., Awodele O. Internet Refrigerator –A typical Internet of) *3rd International Conference on Advances in Engineering Sciences & Applied Mathematics (ICAESAM'2015)*. 2015, pp. 23-24

[6] Robertson T., Wagner I. CSCW and the Internet of Things. *ECSCW 2015: Proceedings of the 14th European Conference on Computer Supported Cooperative Work.* 2015.

[7] Grudin J., Poltrock S. Taxonomy and Theory in Computer, *The oxford handbook of organizational psychology*, 2012. http://www.jonathangrudin.com/wp-content/up-loads/2017/03/Taxonomy.pdf (14.05.2020).

[8] Video Surveillance via RTSP. https://trueconf.com/features/integration/rtsp.html (14.05.2020)

[9] Skype features. https://www.skype.com/en/features/ (14.05.2020)

[10] Microsoft Teams documentation. https://docs.microsoft.com/en-us/microsoftteams/teams-overview (14.05.2020)

[11] Twitter help center. https://help.twitter.com/en/using-twitter#tweets (14.05.2020)

[12] Koch M., Gross T. Computer-Supported Cooperative Work - Concepts and Trends, *Best Papers of the 11th International Conference of the Association Information and Management (AIM),* 2006. https://dl.gi.de/bitstream/handle/20.500.12116/23754/gi-proc-092-010.pdf (14.05.2020).

[13] Boulus-Rødje N., Ellingsen G., Bratteteig T., Aanestad M., Bjorn P. ECSCW 2015: Proceedings of the 14th European Conference on Computer Supported Cooperative Work, 19-23 September 2015, Oslo, Norway. Switzerland: Springer International Publishing. 2015.

[14] PenichetaI V., Marin I., Galluda J. L. M., Tesorieroa R. A Classification Method for CSCW Systems. *Electronic Notes in Theoretical Computer Science*, 2007, 168.

[15] Goel. A. 10 Best JavaScript Frameworks to Use in *2020*. 2020. https://hackr.io/blog/best-JavaScript-frameworks (14.05.2020).

[16] Hámori F. The History of React.js on a Timeline, 2018. https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/ (14.05.2020)

[17] ReactJs. https://reactjs.org/ (14.05.2020).

[18] MobX. https://mobx.js.org/README.html (14.05.2020).

[19] Microsoft Authentication Library for JavaScript (MSAL.js). https://github.com/AzureAD/microsoft-authentication-library-for-js (14.05.2020).

[20] Microsoft Authentication Library for JavaScript (MSAL.js). https://www.npmjs.com/package/msal (14.05.2020).

[21] Media Stream Library JS. https://github.com/AxisCommunications/media-stream-library-js (14.05.2020).

[22] Firebase JavaScript SDK Reference. https://firebase.google.com/docs/reference/js (14.05.2020).

[23] Overview of Cloud Firestore. https://cloud.google.com/firestore/docs/overview (14.05.2020).

[24] Cloud Firestore. https://firebase.google.com/docs/firestore (14.05.2020).

[25] Firebase Realtime Database. https://firebase.google.com/docs/database (14.05.2020).

[26] Choose a database: Cloud Firestore or Realtime Database. https://firebase.google.com/docs/firestore/rtdb-vs-firestore (14.05.2020).

[27] Cloud Storage. https://firebase.google.com/docs/storage (14.05.2020).

[28] Firebase Authentication. https://firebase.google.com/docs/auth (14.05.2020).

[29] Get started with Cloud Firestore Security Rules. https://firebase.google.com/docs/firestore/security/get-started (14.05.2020).

[30] Firebase Hosting. https://firebase.google.com/docs/hosting (14.05.2020).

[31] Cloud Run. https://cloud.google.com/run#overview (14.05.2020).

[32] Anderson C. Docker. *IEEE Software*, 2015, 32.

[33] Turnbull J. The Docker Book: Containerization Is the New Virtualization. Brooklyn: Turnbull Press, 2014.

[34] Rouse M. Docker image. 2020. https://searchitoperations.techtarget.com/definition/Docker-image (14.05.2020).

[35] Merkel D. Docker: lightweight Linux containers for consistent development and deployment. *Linux Journal*, 2014. https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment (14.05.2020).

[36] Microsoft identity platform documentation. https://docs.microsoft.com/en-us/azure/active-directory/develop/ (14.05.2020).

[37] Spring Boot. https://spring.io/projects/spring-boot#overview (14.05.2020).

[38] Spring Security. https://spring.io/projects/spring-security (14.05.2020).

[39] Spring Security AAD: Wire up enterprise grade authentication and authorization https://azure.microsoft.com/en-us/blog/spring-security-azure-ad/ (14.05.2020).

[40] Haller S., Karnouskos S., Schroth C. The Internet of Things in an Enterprise Context. *Future Internet - FIS 2008.* Berlin: Springer, 2008, pp. 14-28.

[41] Vermesan O., Friess P. Internet of Things - Global Technological and Societal Trends, Aalborg: River Publishers, 2011.

[42] Yang M.-J., Tham J. Y., Wu D., Goh, K. H. Cost effective IP camera for video surveillance. *2009 4th IEEE Conference on Industrial Electronics and Applications*, 2009. https://ieeexplore.ieee.org/document/5138638 (14.05.2020).

[43] AXIS M1065-LW Network Camera. https://www.axis.com/files/datasheet/ds_m1065lw_t10066560_en_2003.pdf (14.05.2020).

[44] VAPIX. https://www.axis.com/vapix-library/subjects/t10037719/section/t10035974/display (14.05.2020).

[45] Real Time Streaming Protocol 2.0 (RTSP). https://tools.ietf.org/id/draft-ietf-mmusic-rfc2326bis-33.html#rfc.section.1 (14.05.2020).

[46] ONVIF. https://www.onvif.org/specs/stream/ONVIF-Streaming-Spec-v1706.pdf (14.05.2020).

[47] ESP32-WROOM-32 https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf (14.05.2020).

[48] ESP32 Pinout Reference: Which GPIO pins should you use? https://randomnerdtutorials.com/esp32-pinout-reference-gpios/ (14.05.2020).

[49] Adafruit - PIR Motion Sensor. https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor?view=all (14.05.2020).

[50] Firebase Realtime Database Arduino Library for ESP32. https://github.com/mobizt/Firebase-ESP32 (14.05.2020).

[51] Yeoh C. Y., Rahman A. A. A. Implementing 802.11 probe request scanner using WARP platform. *2014 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*, 2014.

[52] Oliveria L., Schneider D., Souza J. D., Shen W. Mobile Device Detection Through WiFi, *IEEE Access* , vol. 7, 2019.

[53] Stallings W. Data and Computer Communications. New Jersey: Pearson Education, 2007.

[54] Espressif Documentation - Wi-Fi https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_wifi.html (14.05.2020).

[55] Azure Spring Boot. https://github.com/microsoft/azure-spring-boot/tree/master/azure-spring-boot-samples/azure-active-directory-spring-boot-stateless-sample (14.05.2020).

[56] MobX – Autorun. https://mobx.js.org/refguide/autorun.html (14.05.2020).

[57] Authenticate Using Microsoft with JavaScript. https://firebase.google.com/docs/auth/web/microsoft-oauth (14.05.2020).

[58] ESP32-WiFi-Sniffer. https://github.com/ESP-EOS/ESP32-WiFi-Sniffer (14.05.2020).

Appendix

I. Repository

The code of the project is available at:

https://bitbucket.org/kasorm/master-thesis-kunik/src/master/

II. License