SENS VAN AERT

Internship ChrackLab F4D

2023 - 2024

Preface

The past three months I have participated in an Erasmus Internship at Crhack Lab Foligno 4D in Foligno, Italy as part of my studies in Applied Computer Science. During my internship I had the opportunity to work on two assignments, the first titled “Protocol Daemon” and the second titled "Transizione ecologica Organismi Culturali e Creativi” or “TOCC” for short.

“Protocol Daemon” involved the development of a software solution aimed at lowering the workload of employees, especially project managers. The Protocol Daemon extracts information from emails, saves that information to a database, and displays the information in a simplistic yet intuitive web interface as “Protocol entries”. On this interface users can manage the protocol entries.

My second assignment was a task within the project “TOCC”, this project aims to educate tourists about the history of the city of Bevagna. Two weeks a year there is a medieval market where tourists can learn about medieval practices in the city. TOCC will make it possible to enjoy those experiences when there is no market. Within this project my task was to make an AR portal that is placed in the real world. When tourists walk in this portal, they enter a room where information is placed in different interactive ways (pictures, videos, presentations, …).

Although very rewarding, both assignments posed various challenges. They required me to learn new technologies and apply knowledge acquired at Thomas More to them. “Protocol Daemon” taught me email parsing and challenged my proficiency in web development. While “TOCC” made me consider placement of objects in relation to each other in a 3D Plane, work around limitations of technologies and more.

This document provides a detailed overview of both assignments, I will discuss the experiences I had during the internship but also the different technologies used and why I used them, problems I encountered and how I solved them, and of course the final result of both assignments. For the Protocol Daemon I will include a link to a video showing how to use the application.

I am beyond grateful for the opportunity to participate in the Erasmus program, it has allowed me to get to know new cultures and expand my international network. I would like to thank my internship supervisor, Jordi De Roeck, for guiding me on this internship and make sure everything went okay, I also would like to thank the coaches of DI, Jochen Maniën, Kathleen Renders, Bram Heyns, to prepare me for real life projects. Within Chracklab F4D I’d like to thank Paolo for providing me with the opportunity to work at ChrackLab F4D and also the counsel provided, and Gabriele for guiding me through the projects.

Contents

1 Protocol Daemon

Every company must deal with emails and the protocols that come with them. Saving information about the emails like, sender, receiver(s), subject, …. This can take up a lot of time for employees, especially if you are a project manager. This is why the protocol daemon was requested, it could free up a lot of time and effort for employees.

My plan for this project was to make a ‘Protocol Daemon’ where users can send emails to, from these emails important information will get extracted. This information will be saved to a database, users can then view this information on a web interface. On this interface users can also edit this information. It is important to differentiate between protocol entries that are in or outbound, confirmed by the user or not and set to active or inactive.

One requirement was that the application was built on Node.js. This is why I opted for using Express.js as a framework for the backend. To realize the frontend, I used a combination of Ejs as a view engine and Tailwindcss for styling. As a Database Management System, I chose to use MySQL. The application is hosted on a server owned by ChemiCloud, using cPanel I can manage the application on the server.
At the end of the project, I realized a ‘Protocol Daemon’ that saves the information extracted in a database which is then manageable through a web interface.

When users need to protocol an incoming email, they forward that email to the Protocol Daemon. This Daemon then extracts the information, saves it in the database and sends a mail back to the users containing a form with the extracted information. Users can change the information here if needed, when content, they confirm the form. This makes the necessary changes to the protocol entry in the database. The protocol entry is now marked as confirmed.

The flow for outgoing emails is a little different. Users first have to send a mail to the Protocol Daemon. When received, information from this mail will be saved and the Protocol Daemon returns a mail containing the ‘Protocol Number’ which they have to insert in their actual mail. When the user decides to send the actual mail to their client, they can opt to include the Protocol Daemon in CC. If they choose to do so, the Daemon will see the protocol number and change information according to this mail. The protocol entry will be marked as confirmed automatically.

On the web interface, users can see all protocol entries. They are divided into 2 tables, one containing unconfirmed protocol entries, and the other containing confirmed protocol entries. Users can filter on in or outbound mails.

Protocol entries cannot be deleted as per request of the client. If a protocol entry is no longer needed or is faulty, they can instead mark the entry as inactive. On the web interface users can navigate to the inactive page. Here they can also set the entries back to active if needed.

1.1 Analysis

Every project needs an analysis. During the analysis we ask the client for information in an effort to uncover what they expect, what the finished product should look like and what it should do. We transform these requirements into functional and non-functional requirements.

1.1.1 Functional Requirements

Functional requirements are the requirements that tell us what the system should be able to do.

- Login
- Start protocol
- Manage sent mail documents
- Manage received mail documents
- Filter tags
- Search mail documents

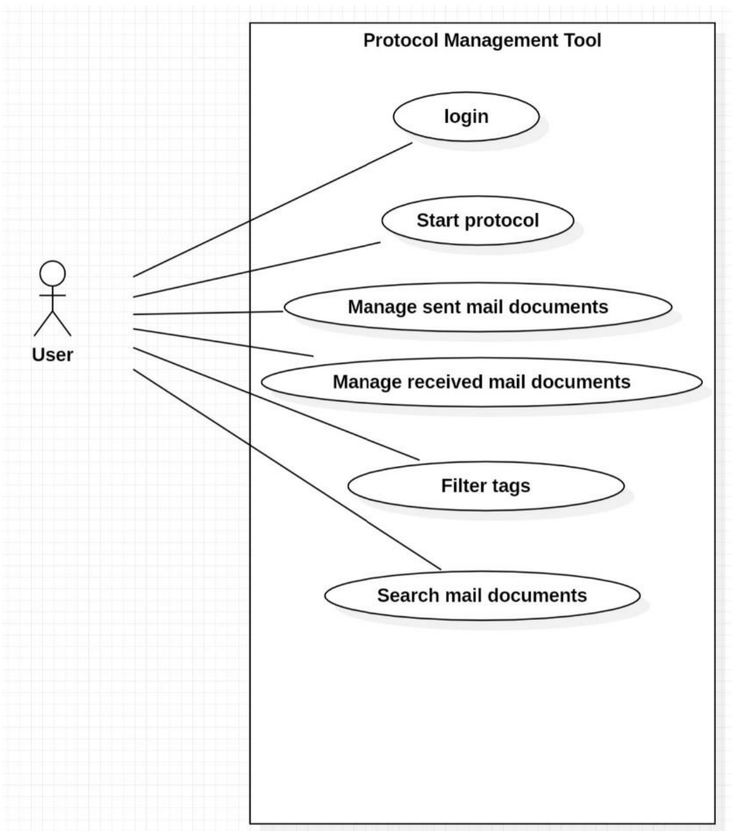


Figure 1 Use Case Diagram

1. As a User, I can login

Normal Flow: System asks user for credentials. Actor fills in credentials and confirms. System checks credentials. System redirects Actor to landing page.

2. As a User, I can start the protocol.

Normal Flow: Actor forwards mail to System mailbox. system starts the protocol and generates a *protocol code**. When system has finished the protocol, it sends an email to the Actor with a link. Actor opens the link. System asks for the user to login. User logs in. System displays the result of the protocol and asks Actor for confirmation. Actor confirms. System creates protocol entry.

Alternatives:

- Start protocol with a mail the Actor has sent: Actor sends a mail with System mailbox in cc.
- Edit protocol entry: Actor changes result of the protocol. Actor confirms changes. System creates protocol entry with changes made by the Actor.
- Cancel protocol entry: Actor cancels the protocol entry. System confirms cancelation.
- Mail has attachment: The mail which goes through our protocol has an attachment. The system saves these file(s) on the server in a folder under the naming convention of *attachment document name*.

3. As a User, I can manage sent mail documents.

Prerequisites: User must be logged in.

Normal Flow: Actor navigates to sent mail documents page. System presents all sent mail documents. Actor chooses option to add mail. System presents form. Actor fills in form and confirms. System creates mail document. System redirects user to previous page.

Alternatives:

- Edit event: Actor chooses option to Delete mail document. System Deletes mail.

- Delete event: Actor chooses option to update mail document. System presents form. Actor fills in form and confirms. System updates mail document. System redirects user to previous page.

4. As a User, I can manage received mail documents.

Prerequisites: User must be logged in.

Normal Flow: Actor navigates to received mail documents page. System presents all received mail documents. Actor chooses option to add mail document. System presents form. Actor fills in form and confirms. System creates mail document. System redirects user to previous page.

Alternatives:

- Edit event: Actor chooses option to Delete mail document. System Deletes mail document.

- Delete event: Actor chooses option to update mail document. System presents form. Actor fills in form and confirms. System updates mail document. System redirects user to previous page.

5. As a User, I can filter on tags.

Prerequisites: User must be logged in.

Normal Flow: System displays filter options. Actor selects filter option(s). System displays filtered mail documents.

6. As a User, I can search mail documents.

Prerequisites: User must be logged in.

Normal Flow: System presents search bar. Actor enters keyword(s) in search bar. System displays mail documents that match keyword(s).

protocol code: The protocol code is a code generated by the protocol from which we can get certain information, ex. `dd-mm-yy-project_code-random_number-i`, `dd-mm-yy-project_code-random_number-o`. with 'i' meaning inbound and 'o' meaning outbound.

attachment document name: The attachment will not be saved in the database but a path to it will. The name under which the attachment will be saved is as follows; `protocol_code-number_of_attachment-attachment.extension`.

1.1.2 Non-Functional Requirements

Non-functional requirements tell us how our system should do certain things, these do not change the function of our system.

- Files need a naming convention (date-project_name-sender-recipient-...)
- Application should be built on Node.js
- Mails should be split by in and outbound

- Mails are to be read from a SMTP server

1.1.3 Data Model

During the analysis phase it is also important to look at what we will need to store in the database. We can deduct this from the requirements of the client. The form of the Data Model changed a few times over the course of the project, it simplified a lot because of some of the requirements being nice to haves and falling away due to time limitations.

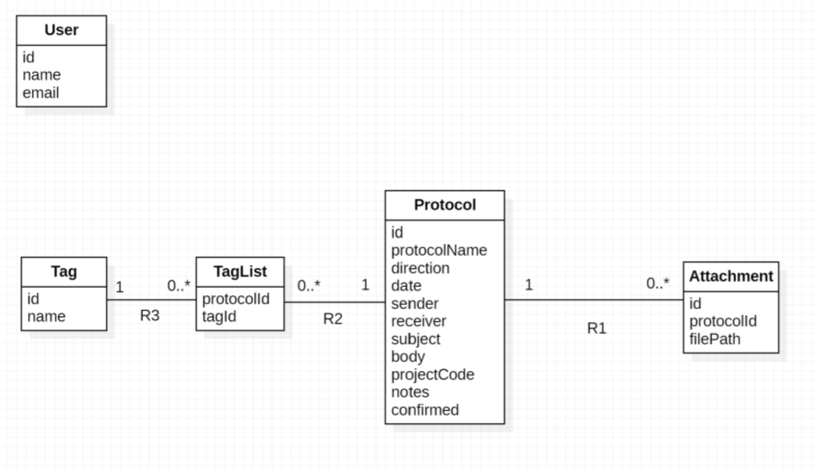


Figure 2 Data Model

Above we can see the first version of the data model. In the centre there is a table called ‘Protocol’, this is where the protocol entries will be stored, it contains attributes like that save information about the mail. The following attributes might need a little more information for people outside of the project to understand:

- protocolName: it gives a meaningful way to identify a protocol entry.
- direction: This identifies if a mail was in or outbound.
- projectCode: If the mail concerns a project, the code if this project can be included.

Other than the ‘Protocol’ table, we see the ‘Attachment’ table, here the path to the saved attachment is stored. Employees can use this path to find the attachment on the server.

‘Tag’ and ‘TagList’ help us connect tags to the protocol entry which can help us on the web interface.

Lastly, ‘User’, in this table we save the name and the email of a certain employee or user for authentication purposes.

In the last version of the data model, we can see that we lost almost all tables, this is mostly because we decided to let the nice to haves fall due to the time limitations.

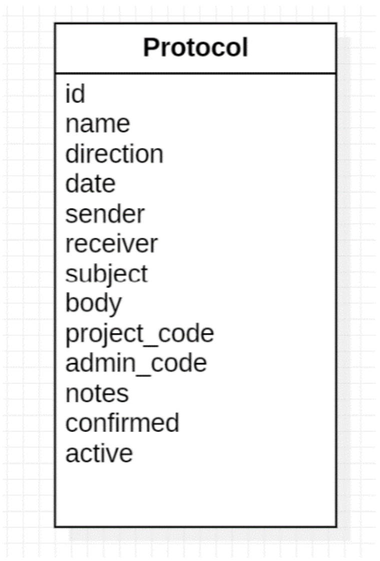


Figure 3 Protocol Table

In this table we chose for a different naming convention, snake_case instead of camelCase, and we added some attributes:

- admin_code: This means essentially the same as project_code but is used by the administration department of ChrackLab.
- Active: because the client didn't want protocol entries to be deleted, they can set protocol entries to inactive.

1.2 Development

Under this section I will discuss what I have developed. I will talk about the different functions, pages, ... I made, what they do and why I made them. Hardships encountered while developing will also be discussed.

1.2.1 Nodejs server & Imap setup

The foremost priority was to develop a Nodejs server with which we can parse mails. The first thing I did in development was setting up a Nodejs server, once it was running I setup an Imap connection with the mailserver to retrieve mails, to parse the mails I used MaileParser.

Later in the project it became evident that making use of a framework would help keep the code clean, readable, and more flexible. I opted for using express.js, The final result of the Node server and connection to the mail server looks like this:

```
const express = require('express');
const Imap = require('imap');
const {simpleParser} = require('mailparser');
```

```
const app = express();
```

```
app.listen(3000);
```

```
const imapConfig = {
    user: "user_name",
    password: "password",
    host: "protocol",
    port: 993,
    tls: true,
    tlsOptions: { rejectUnauthorized: false }
};

const imap = new Imap(imapConfig);
```

In the code presented below, you can see how I retrieved mails, and what creatiria I used to define what mails should be retrieved, I used both console.log and logger.debug for debugging (console.log for local development, logger.debug for debugging on the server):

```
imap.once('ready', () => {
    /*  imap.getBoxes((err, mailbox) => {
            console.log(mailbox);
      }); */
    imap.openBox(mailbox, false, (err) => {
        if (err){
            logger.debug('OPEN MAILBOX: ', err);
        }
        imap.addListener('mail', (received) => {
            // We look for unseen emails that have been received today
            imap.search(['UNSEEN', ['ON', new Date()]], (err, results) => {
                if (err) {
                    logger.debug('Error in imap.search: ', err);
                    console.log('ERROR AFTER SEARCH:', err);
                }
                try{
                    // We only fetch the latest email
                    const f = imap.fetch(results[results.length-1], {bodies:
''});
                    f.on('message', msg => {
                        msg.on('body', stream => {
                            simpleParser(stream, async (err, parsed) => {
                                msg.once('attributes', attrs => {
                                    const {uid} = attrs;
        // To mark as Unread, remove the flag '\\Seen'
                                    imap.addFlags(uid, ['\\Seen'], () => {
                                        console.log("Marked as read!");
                                    });
                                });
                            });
                            f.once('error', ex => {
                                console.log(ex);
                                imap.end();
                            });
                            f.once('end', () => {
                                console.log('Done fetching all messages!');
                            });
                        } catch(e) {
                            console.log("Couldn't find messages " + e);
                        }
                    });
                });
            });
        });
    } catch (ex) {
        imap.end();
        console.log(ex);
    }

    imap.once('error', function(err) {
        console.log('Mailbox error: ', err);
        logger.debug('Mailbox error: ', err);
        imap.end();
    });

    imap.once('end', function() {
        db_con.end();
        console.log('Connection to mailbox ended!');
        logger.debug('Connection to mailbox ended!');
    });

    imap.connect();
}
```

1.2.2 Database

After the local Node.js server was running and we could retrieve the last unread mail, I set up the database so that when we extract the information, we can immediately store it in the database. I set up a Mysql database according to the mode discussed in 1.1.3 .
To connect to the database from our Node.js server we need following code:

```
const mysql = require('mysql');
```

```
var db_con = mysql.createConnection({
    host: 'host',
    user: 'user',
    password: 'password'
});
```

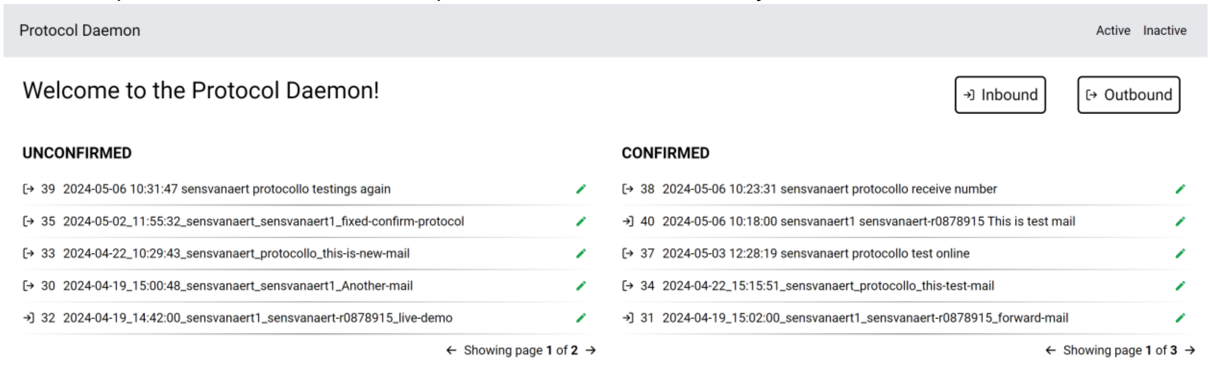
```
db_con.connect((err) =>{
    if (err) {
        logger.debug('CONNECTION ERROR TO DATABASE: ', err);
        return;
    }
    console.log("Connected to the db!");
    logger.debug('Connected to the db!');

    db_con.once('error', (err) => {
        console.log('Error occured(App.js): ', err);
        logger.debug('Error occured(App.js): ', err);
    });
});
```

1.2.2 Landing page

Once the server was setup I created a frontend. Using this frontend, I will explain functions created and used in the backend, only showing code when needed.

The client wanted a simplistic look, that was easy to read and was easy to navigate. They did not want any colorful pages. I used tailwindcss, to accomplish this style.
The application needed an index or landing page, here all active protocol entries are displayed divided by confirmed and unconfirmed. Each protocol entry is led by a symbol that indicates if the mail was in or outbound, then we see the protocol number and after that the protocol name. Protocol entries are sorted by date in a descending order (Newest first). Users can also use the buttons at the top right of the page to filter between inbound and outbound.
When the Protocol Daemon is in use the amount of protocol entries will increase fast. This will result in very long lists that can take a long time to load. To combat this, I made use of pagination. In this test version I retrieve 5 protocol entries at a time. In production this will most likely be increased to 50.



Welcome to the Protocol Daemon!

UNCONFIRMED

- 39 2024-05-06 10:31:47 sensvanaert protocollo testings again
- 35 2024-05-02_11:55:32_sensvanaert_sensvanaert1_fixed-confirm-protocol
- 33 2024-04-22_10:29:43_sensvanaert_protocollo_this-is-new-mail
- 30 2024-04-19_15:00:48_sensvanaert_sensvanaert1_Another-mail
- 32 2024-04-19_14:42:00_sensvanaert1_sensvanaert-r0878915_live-demo

← Showing page **1** of **2** →

CONFIRMED

- 38 2024-05-06 10:23:31 sensvanaert protocollo receive number
- 40 2024-05-06 10:18:00 sensvanaert1 sensvanaert-r0878915 This is test mail
- 37 2024-05-03 12:28:19 sensvanaert protocollo test online
- 34 2024-04-22_15:15:51_sensvanaert_protocollo_this-test-mail
- 31 2024-04-19_15:02:00_sensvanaert1_sensvanaert-r0878915_forward-mail

← Showing page **1** of **3** →

Figure 4 Landing Page

I chose to use Ejs as a view engine to make it easier to script in the html pages.

```
app.set('view engine', 'ejs');
```

Ejs enabled me to use res.render() to render the ejs pages. Below an example is shown of how we can use this function. First, we define the page (the view engine goes looking for a file named index.ejs under views folder) and then we can add variables to send them to the page:

```
res.render('index', { unconfirmedProtocols: unconfirmedRecords,
confirmedProtocols: confirmedRecords, unconfirmedPage, confirmedPage, filter,
recordCount: limit, nbrOfUnconfirmed, nbrOfConfirmed, title: 'Welcome to the
Protocol Daemon!', rootdir: process.env.ROOTDIR || '' });
```

This method is used in most routes in the application.

1.2.3 The Protocol

Before we can show any of the protocol entries, we need to get them somehow. Using imap I connected to the mailserver and retrieved unread received mails from the protocol daemon's mailbox. After a mail is received, we can generate a protocol entry by extracting the necessary information from the mails. When this is done the protocol has to send a mail back, and store the information in the database. This required the use of several functions, I had to convert date formats and apply my knowledge of data science to ensure that the data didn't include null values where we don't want it, remove spaces, or replace characters if needed.

1.2.3.1 Inbound mails

I first started working on inbound emails, the workflow of in and outbound emails differs. Later I extended on this for outbound emails.

Inbound emails are forwarded to the mailbox of the Protocol Dameon. When a mail is received, we look at the subject of that mail and check if it contains “FW:” using regex. If it does, we activate the function “extractEmailInfo”. In this function we make use of regex to extract all information from the body of the mail.

Extracting information from the mail body:

```
function extractEmailInfo(emailString) {
    let recipientList = [];

    const fromMatch = emailString.match(/From: (.+?) <(.+?)@.*>/);
    const toMatch = emailString.match(/To: (.+)/);
    const subjectMatch = emailString.match(/Subject: (.+)/);
    const dateMatch = emailString.match(/Date: (.+)/);
    const noteMatch = emailString.match(/Notes: (.+)/);
    const projectCodeMatch = emailString.match(/Project code: (.+)/);
    const adminCodeMatch = emailString.match(/Administrative code: (.+)/);
    const protocolNumberMatch = emailString.match(/Protocol number: (.+)/);

    const from = fromMatch ? fromMatch[2] : '';
    const to = toMatch ? toMatch[1] : '';
    const subject = subjectMatch ? subjectMatch[1] : '';
    const date = dateMatch ? dateMatch[1] : null;
    const notes = noteMatch ? noteMatch[1] : '';
    const projectCode = projectCodeMatch ? projectCodeMatch[1] : '';
    const adminCode = adminCodeMatch ? adminCodeMatch[1] : '';
    const protocolNumber = parseInt(protocolNumberMatch ?
protocolNumberMatch[1] : '');

    if(to){
        let recipients = to.split(',');
        for(let i in recipients){
            const recipientMatch = recipients[i].match(/<(.+?)@.*>/);
            recipientList.push(recipientMatch ? recipientMatch[1] : '');
        }
    }

    // Date format is invalid, we have to convert it ourselves
    let newDate;
    if( date){
        const parts = date.split(" ");
        const dayName = parts[0].slice(0,-1);
        const day = parseInt(parts[1]);
        const month = parts[2];
        const year = parseInt(parts[3]);
        const time = parts[5];
        newDate = `${dayName} ${month} ${day} ${year} ${time}`;
    }

    const body = emailString.split(/\n\s*\n/).slice(1).join('\n').trim();
    return { from, recipientList, newDate, subject, body, notes, projectCode,
adminCode, protocolNumber };
}
```

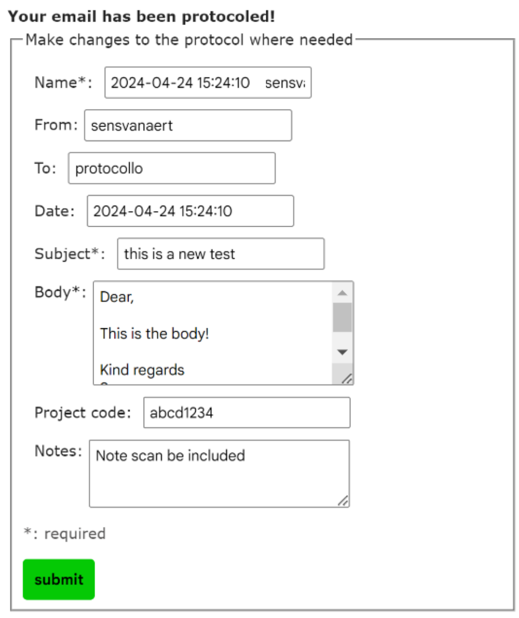
As you can see in the code, we also must convert the date format. Using a built-in function, we could not convert the date format, therefore I had to convert it myself.
From the retrieved mail's body we can extract, the sender of the mail, recipient(s), date, subject, body, and notes, projectCode, adminCode, and protocolNumber if the user chooses to add them.
Finally, when all this information is extracted, we store it in the database.

When trying to save to the database I first encountered a problem with the date, MySQL did not accept this format. I still wanted to use the format I was using now for the form as it is more readable for humans, but I had to convert it again to be able to save it the MySQL database. That is why I wrote the function "formatDateForMySQL()":

```
function formatDateForMySQL(dateString) {
    const date = new Date(dateString);
    const year = date.getFullYear();
    const month = (date.getMonth() + 1).toString().padStart(2, '0');
    const day = date.getDate().toString().padStart(2, '0');
    const hours = date.getHours().toString().padStart(2, '0');
    const minutes = date.getMinutes().toString().padStart(2, '0');
    const seconds = date.getSeconds().toString().padStart(2, '0');

    return `${year}-${month}-${day} ${hours}:${minutes}:${seconds}`;
}
```

Other than extracting information we want to send this information back to the users for review and confirmation. We do this by sending a mail containing a form. That is why; after storing the information, we retrieve the same record. I link the retrieved id of the protocol entry to the form we send to the user. This way we make sure we confirm the right protocol entry with the form. If the user wants to make changes to the entry, they can change that in the form and upon confirmation it will be changed in the database. This action also marks the protocol entry as confirmed.
Below is an example of what the form that the protocol daemon sends looks like. Some fields are marked with a '*' this means that those fields are mandatory to fill in.



Your email has been protocoled!

Make changes to the protocol where needed

Name*:	2024-04-24 15:24:10 sensv
From:	sensvanaert
To:	protocollo
Date:	2024-04-24 15:24:10
Subject*:	this is a new test
Body*:	Dear, This is the body! Kind regards
Project code:	abcd1234
Notes:	Note scan be included

*: required

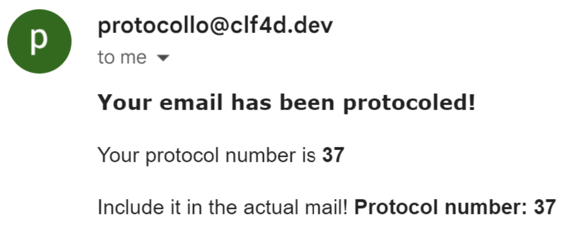
Figure 5 Confirmation form

When creating the route for the confirmation of the protocol entries, the post method did not work. The route was unreachable. To fix this I had to create a route with the same name but using the get-method. Now the route with the post-method was reachable and the protocol entries could be confirmed using the form.

1.2.3.2 Outbound mails

The process for outbound emails consists of 2 steps, first the user sends a mail directly to the Protocol Daemon. The Protocol Daemon then answers with a mail containing the id of the protocol entry in the database, also known as the protocol number. After this, the user can send their mail to the person it is intended to with Protocol Daemon in CC and the protocol number included in the mail. The Protocol Daemon handles the rest and saves everything in the database.

We open the mailbox in the exact same way as before and check for the latest unread email. This time it is a mail directly sent to the Protocol Daemon. We again extract all information, apply the necessary data cleaning, and save it to the database. When this goes well, we retrieve the protocol entry and send a mail back containing the protocol number and an example how users should use the protocol number in their actual email.


protocollo@clf4d.dev
to me

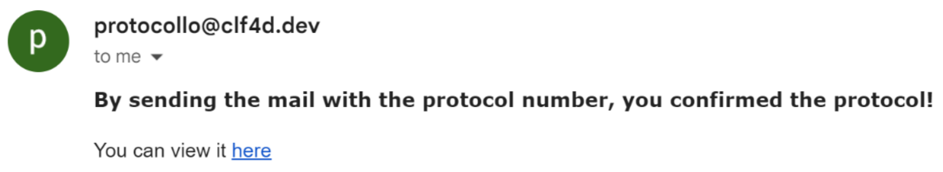
Your email has been protocoled!

Your protocol number is **37**

Include it in the actual mail! **Protocol number: 37**

Figure 6 mail with protocol number

Now we need to listen to emails that contain a protocol number, when we receive this, we can edit the information and confirm the protocol entry. We do this as follows: We first check if the email is forwarded if not, then we check if Protocol Daemon is in CC, if it is, then we know we have an email which possibly has a protocol number.
When it does, we extract the information and edit the protocol entry. If all is successful, we can mark the protocol entry as confirmed. We now send a mail to the user with confirmation.


protocollo@clf4d.dev
to me

By sending the mail with the protocol number, you confirmed the protocol!

You can view it here

Figure 7 mail with confirmation of protocol

1.2.3.3 Sending mails

Before I could send an email, I had to import the NodeMailer module, and create a transport through which I can send the mail:

```
const nodemailer = require("nodemailer");
```

```
const transporter = nodemailer.createTransport({
    host: __MAIL_HOST,
    port: 465,
    secure: true,
    auth: {
        user: __MAIL_USER,
        pass: __MAIL_PASS,
    },
});
```

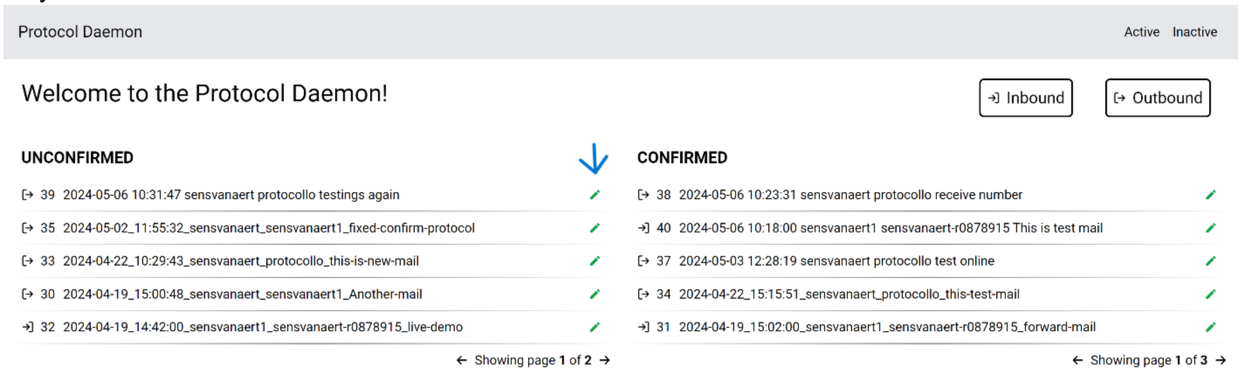
I use the function ‘send_mail()’ to send the mail. I use the information gathered from the previous steps to send a reply on the initial mail from the user.

```
async function sendMail(userMail, replyMessageId, subject, body) {
    try{
        // send mail with defined transport object
        const info = await transporter.sendMail({
            from: '<'+ __MAIL_USER +'>', // sender address
            to: userMail, // list of receivers
            replyTo: userMail,
            inReplyTo: replyMessageId,
            references: [replyMessageId],
            subject: "Re: " + subject,
            html: body
        });

        console.log("Message sent: %s", info.messageId);
        // Message sent: <d786aa62-4e0a-070a-47ed-0b0666549519@ethereal.email>
    } catch(ex) {
        console.log("Couldn't send message:" + ex);
    }
}
```

1.2.4 Edit protocol entries

Once the protocol entries are saved in the database and we can view them in the database, we also need to update or edit them. Users can do this on the web interface clicking on the pencil of the protocol entry they want to edit.



Welcome to the Protocol Daemon!

UNCONFIRMED

```
39  2024-05-06 10:31:47 sensvanaert protocollo testings again
35  2024-05-02_11:55:32_sensvanaert_sensvanaert1_fixed-confirm-protocol
33  2024-04-22_10:29:43_sensvanaert_protocollo_this-is-new-mail
30  2024-04-19_15:00:48_sensvanaert_sensvanaert1_Another-mail
32  2024-04-19_14:42:00_sensvanaert1_sensvanaert-r0878915_live-demo
```

Showing page **1** of **2**

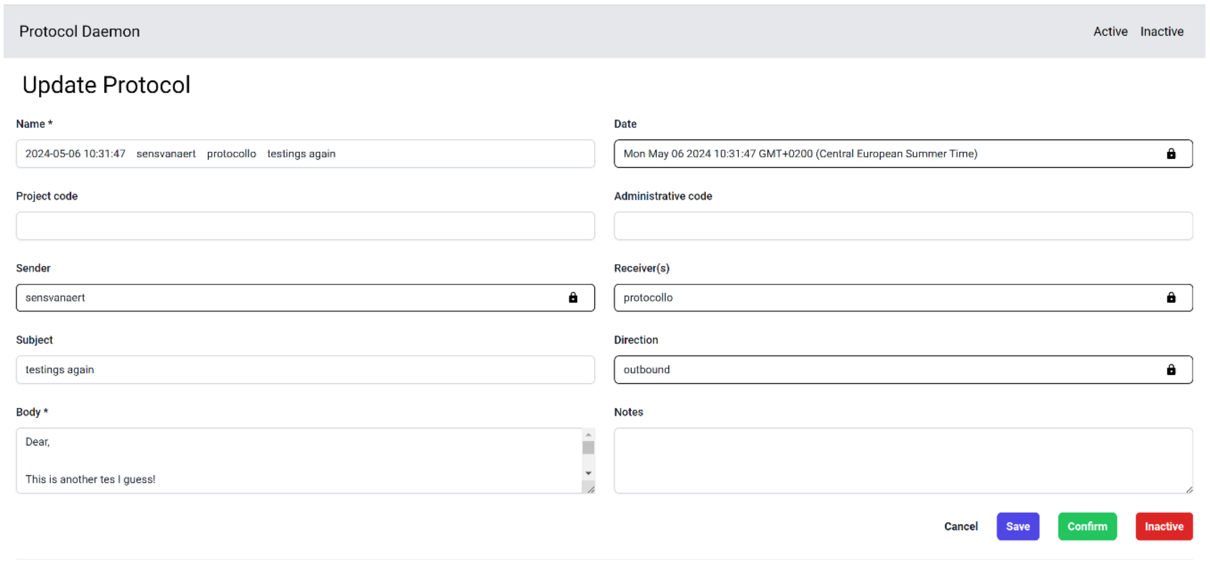
CONFIRMED

```
38  2024-05-06 10:23:31 sensvanaert protocollo receive number
40  2024-05-06 10:18:00 sensvanaert1 sensvanaert-r0878915 This is test mail
37  2024-05-03 12:28:19 sensvanaert protocollo test online
34  2024-04-22_15:15:51_sensvanaert_protocollo_this-test-mail
31  2024-04-19_15:02:00_sensvanaert1_sensvanaert-r0878915_forward-mail
```

Showing page **1** of **3**

Figure 8 Landing page with indicating arrow

This sends them to the edit page of the web interface. Here they can review all information saved about the protocol entry and change some of it. A few fields are locked because they shouldn’t be able to be altered, these are indicated by a lock. Depending on if the protocol entry is confirmed or active the edit page has different buttons at the bottom. With these buttons users can also confirm unconfirmed entries or set protocol entries to inactive or back to active.

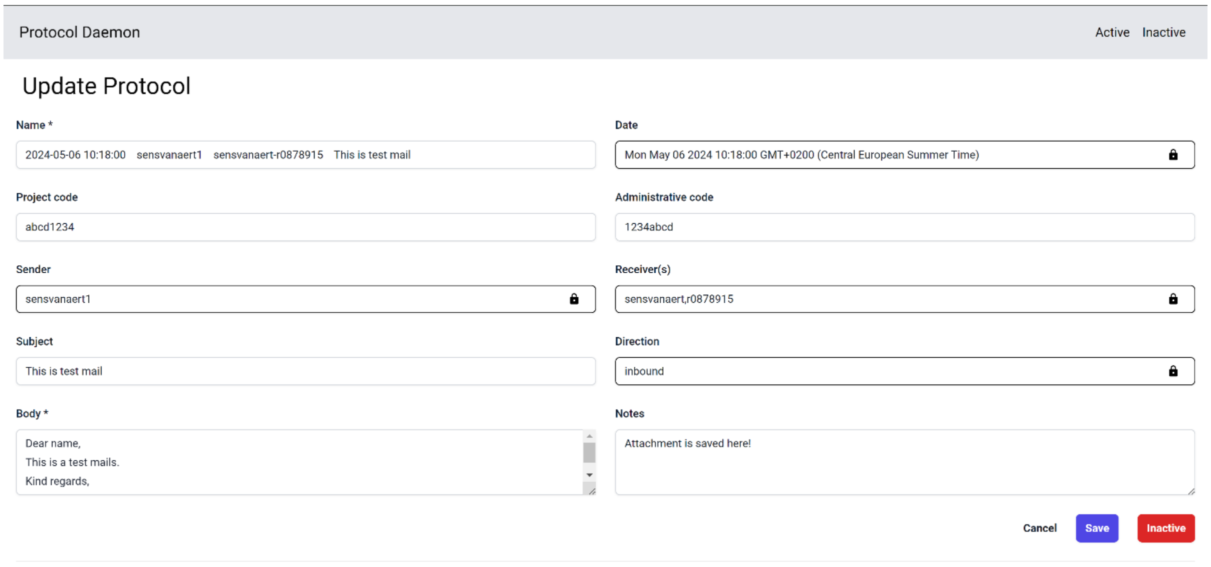


Protocol Daemon Active Inactive

Update Protocol

Name *	2024-05-06 10:31:47 sensvanaert protocollo testings again
Date	Mon May 06 2024 10:31:47 GMT+0200 (Central European Summer Time)
Project code	
Administrative code	
Sender	sensvanaert
Receiver(s)	protocollo
Subject	testings again
Direction	outbound
Body *	Dear, This is another tes I guess!
Notes	

Figure 9 Edit page of unconfirmed active protocol entry

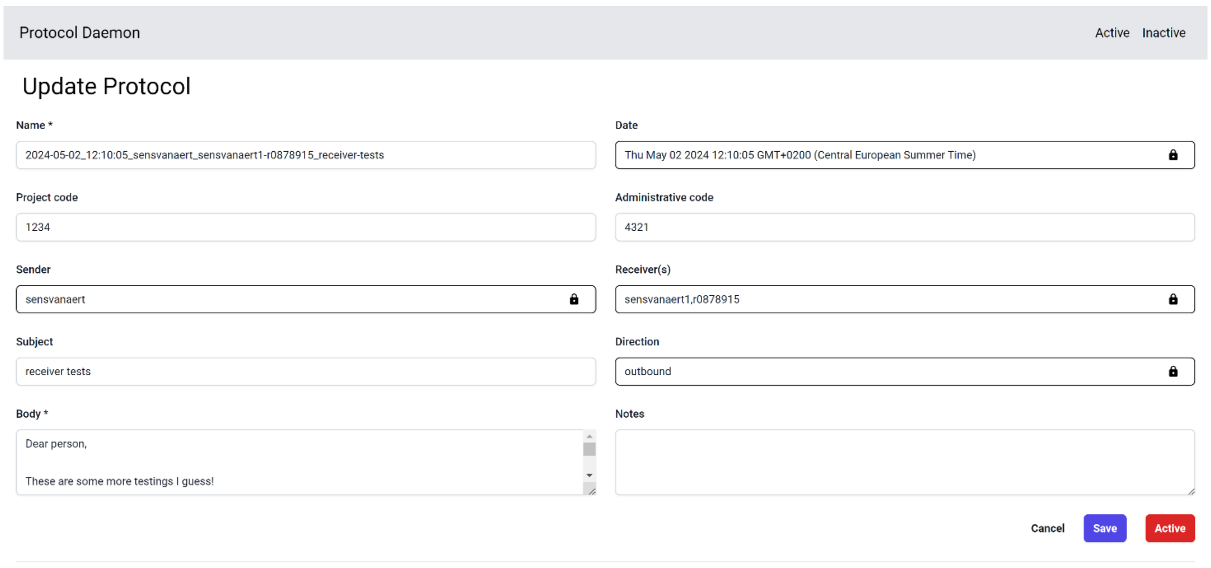


Protocol Daemon Active Inactive

Update Protocol

Name *	2024-05-06 10:18:00 sensvanaert1 sensvanaert-r0878915 This is test mail
Date	Mon May 06 2024 10:18:00 GMT+0200 (Central European Summer Time)
Project code	abcd1234
Administrative code	1234abcd
Sender	sensvanaert1
Receiver(s)	sensvanaert,r0878915
Subject	This is test mail
Direction	inbound
Body *	Dear name, This is a test mails. Kind regards,
Notes	Attachment is saved here!

Figure 10 Edit page of confirmed active protocol entry


Protocol Daemon Active Inactive

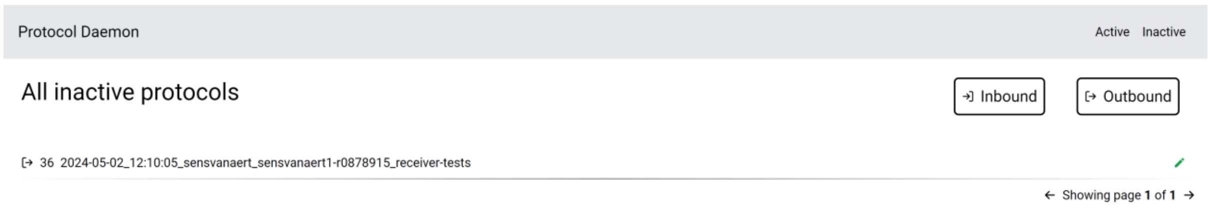
Update Protocol

Name *	2024-05-02_12:10:05_sensvanaert_sensvanaert1-r0878915_receiver-tests
Date	Thu May 02 2024 12:10:05 GMT+0200 (Central European Summer Time)
Project code	1234
Administrative code	4321
Sender	sensvanaert
Receiver(s)	sensvanaert1,r0878915
Subject	receiver tests
Direction	outbound
Body *	Dear person, These are some more testings I guess!
Notes	

Figure 11 Edit page of inactive protocol entry

1.2.3 Inactive protocol entries

When a protocol entry is incorrect or no longer needed, we do not want to delete it. Instead, we set it too inactive. After a protocol entry is set to inactive in can be found on the web interface in a list of inactive protocol entries. Here users can also edit those protocol entries and even set them back to active if needed. The list of inactive protocol entries can also be filtered on in- or outbound mails.


Protocol Daemon Active Inactive

All inactive protocols

36 2024-05-02_12:10:05_sensvanaert_sensvanaert1-r0878915_receiver-tests

Showing page 1 of 1

Figure 12 Inactive protocol entries page

2. TOCC

Bevagna is a city with a rich history, especially a medieval history, every summer a market is held, Il Mercato Delle Gaite. The sad part is that it can only be enjoyed for two weeks a year. That is why they introduced the TOCC project. With this project tourists of the city of Bevagna can enjoy the medieval experience through a web-based experience.

My task in this project is to create a portal that can be placed in the real-world using AR. Users can walk through the portal, when they do, the real world is masked, and they can see a room. In this room information is shown using different methods such as videos and presentations. Users can walk around this room and discover this information.

People can find these portals on the website that is being developed for the project. On this website monuments are placed on a map, the portal I created can be linked to these monuments. The room I created can be used for all monuments; the content can be changed to what is needed.

2.1 The Portal

The most important and first thing I had to make was the portal. While using ARjs and a-frame, I made use of the photosphere as a room and applied it to an a-frame portal entity. People can walk in essentially a big sphere. I made use of Gabriele's work he had done before as a reference.

Later when we found that ARjs was not the best solution for our problem, we switched to zapworks and changed the method we used for the portal. In zapworks we can define an area, if the user walks true this area, we can execute a script. This script activates the mask to hide the real world with the room. When they walk out, this mask disables and a mask to hide the room activates.

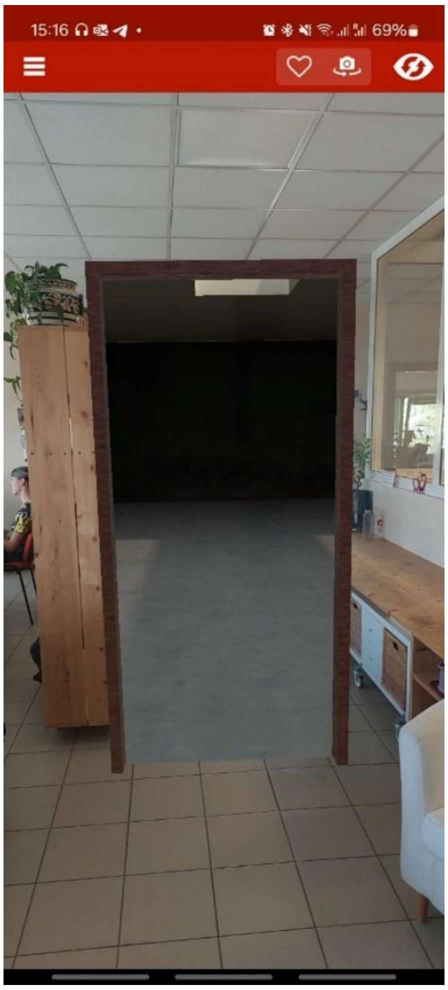


Figure 13 Placed portal

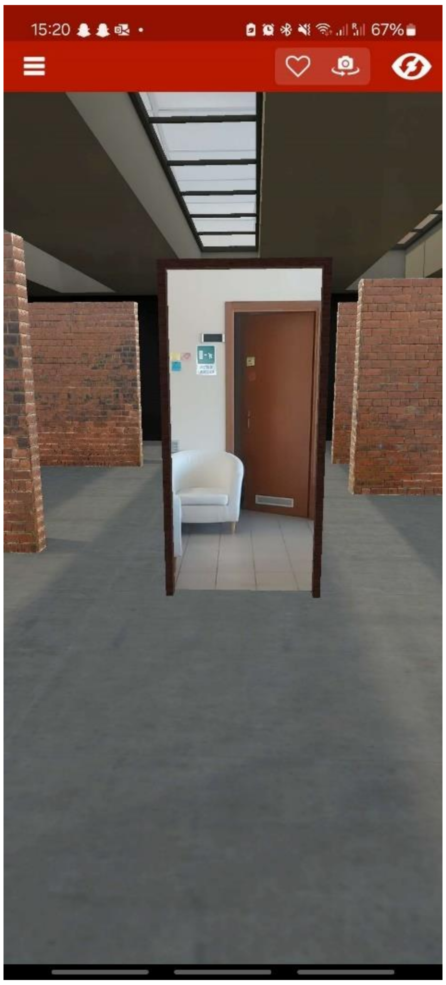


Figure 14 Portal seen from inside the room

2.2 Videos

Empty rooms would be a bit boring, of course we want some content in these rooms. I added videos to the room. I started with adding a plane to the scene and uploaded a video to zapworks studio. I replaced the material of the plane with the video. I now could place the video wherever I wanted in the scene.
But this is not enough, we want to play and pause the video and control the volume. To realize this, I first added a pause and a play png. By connecting states to the place on which the png for pause and play are bound, I could change them whenever a user clicks on them.

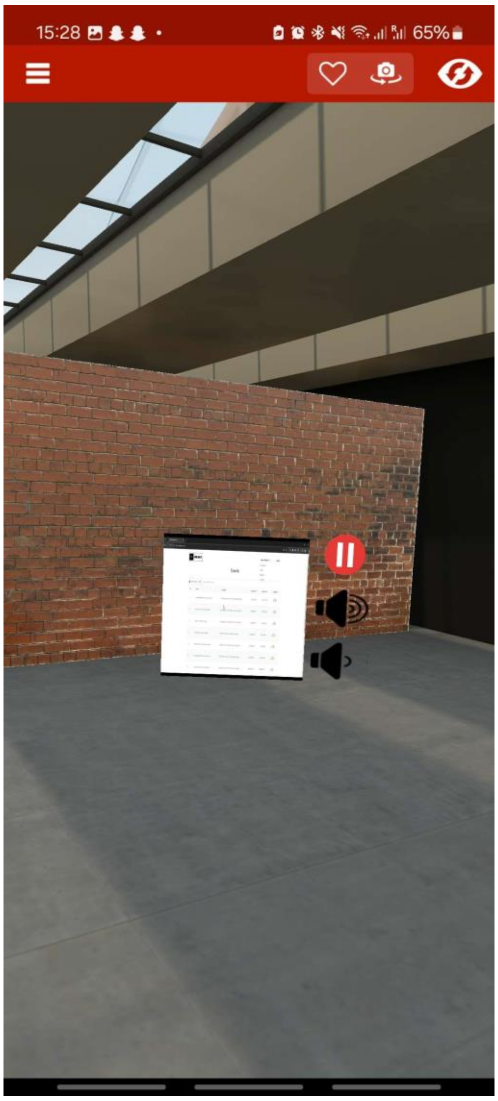


Figure 15 Playing video

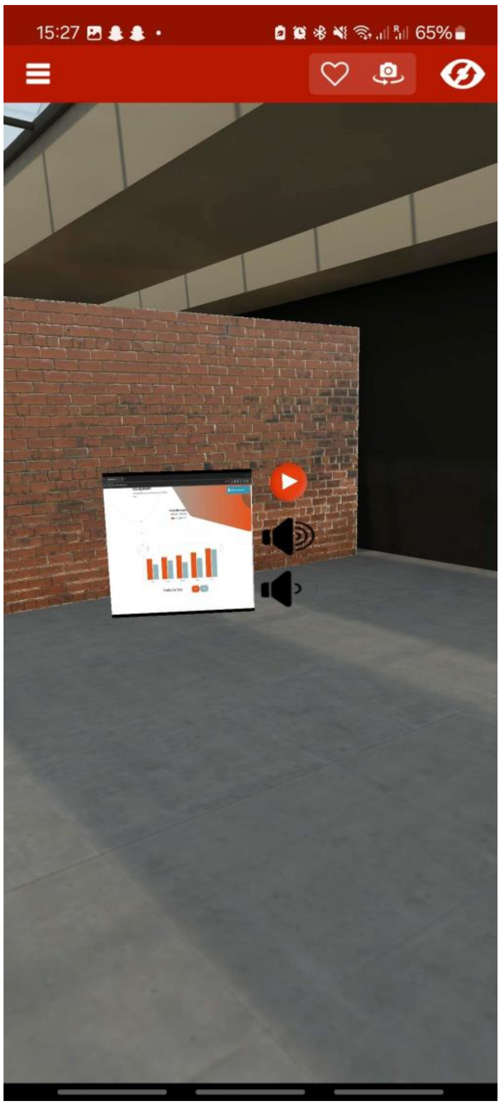


Figure 16 Paused video

On these screenshots you can also see two png's depicted volume up and volume down. Using a script I could control the volume of the video. When a user clicks on volume up, the volume goes up, when a user clicks on volume down, the volume goes down.

Now that we can control the video, we want to have different options on how to show it. The first option is to show it standing still in one place, not moving. The other is to make it always face the user. The first option is easy. We can position the plane wherever we like, and it stays there. The second option was a little harder.

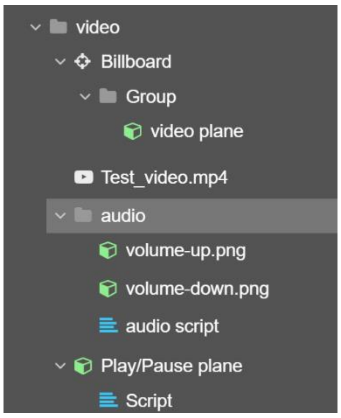
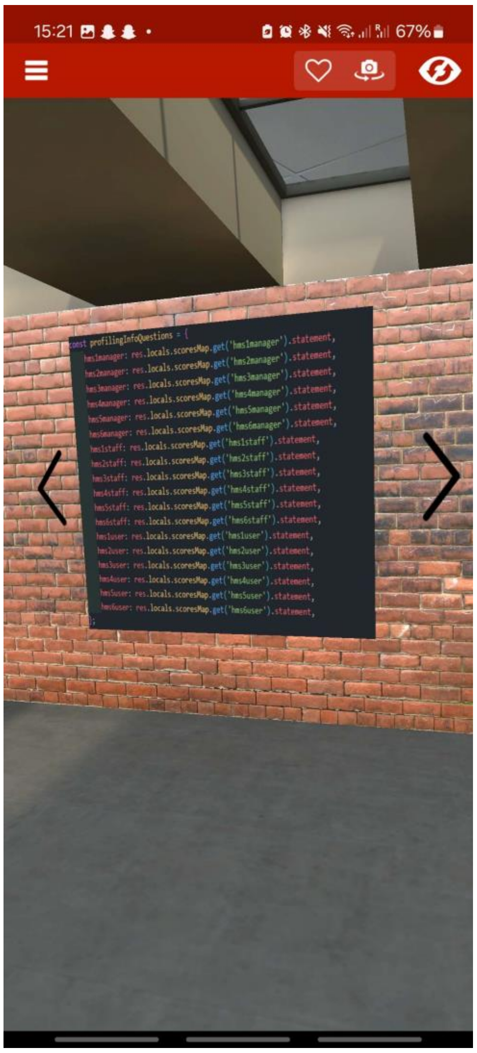


Figure 17 Project Hierarchy

A billboard object always faces the user, but this means when we rotate the camera sideways it also rotates with us. This is not the behavior we want. To prevent this, we can block the billboard from rotating over certain axes. We would have to block both the y- and x-axis, so that the billboard can only rotate over the z-axis. This did not work as expected, the plane was rotated 90 degrees over the x-axis and didn't rotate with the user, I had to find another solution. This solution was wrapping the billboard in a group that is rotated 90 degrees over the x-axis. In the billboard we place another group that is rotated -90 degrees over the x-axis. Now the content in the billboard always faces the user but doesn't follow the rotation of the camera, just as we needed. The hierarchy of the video object is shown in the figure on the left.

2.3 Presentations

Presentations in the form of powerpoints also must be added to the scene. Gabriele made the functionality of the powerpoints, and I had to place them in the scene. Gabriele sent me the .zzp file containing the logic for the presentations, I used this to create the presentation objects in my scene.
Users can use the arrows to go to the next or previous slide.



```
const profilingInfoQuestions = {
    hms1manager: res.locals.scoresMap.get('hms1manager').statement,
    hms2manager: res.locals.scoresMap.get('hms2manager').statement,
    hms3manager: res.locals.scoresMap.get('hms3manager').statement,
    hms4manager: res.locals.scoresMap.get('hms4manager').statement,
    hms5manager: res.locals.scoresMap.get('hms5manager').statement,
    hms6manager: res.locals.scoresMap.get('hms6manager').statement,
    hms1staff: res.locals.scoresMap.get('hms1staff').statement,
    hms2staff: res.locals.scoresMap.get('hms2staff').statement,
    hms3staff: res.locals.scoresMap.get('hms3staff').statement,
    hms4staff: res.locals.scoresMap.get('hms4staff').statement,
    hms5staff: res.locals.scoresMap.get('hms5staff').statement,
    hms6staff: res.locals.scoresMap.get('hms6staff').statement,
    hms1user: res.locals.scoresMap.get('hms1user').statement,
    hms2user: res.locals.scoresMap.get('hms2user').statement,
    hms3user: res.locals.scoresMap.get('hms3user').statement,
    hms4user: res.locals.scoresMap.get('hms4user').statement,
    hms5user: res.locals.scoresMap.get('hms5user').statement,
    hms6user: res.locals.scoresMap.get('hms6user').statement,
};
```

Figure 18 Presentation in room

2.4 The Room

The room itself went through an evolution. Before we had received a .glb file that we had to use as the room, I used a photosphere as the room. Once we received the .glb, I had to use that as the room. I inserted it in the scene and tried to place the content in the room.

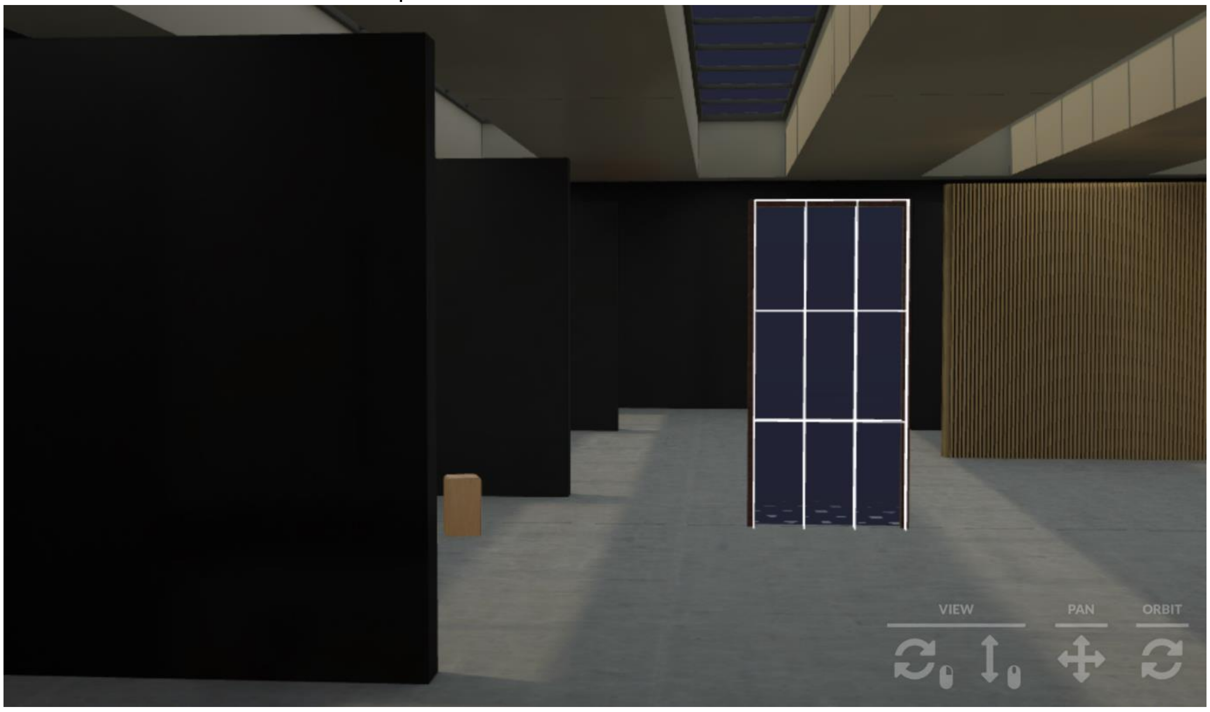


Figure 19 Back of the room

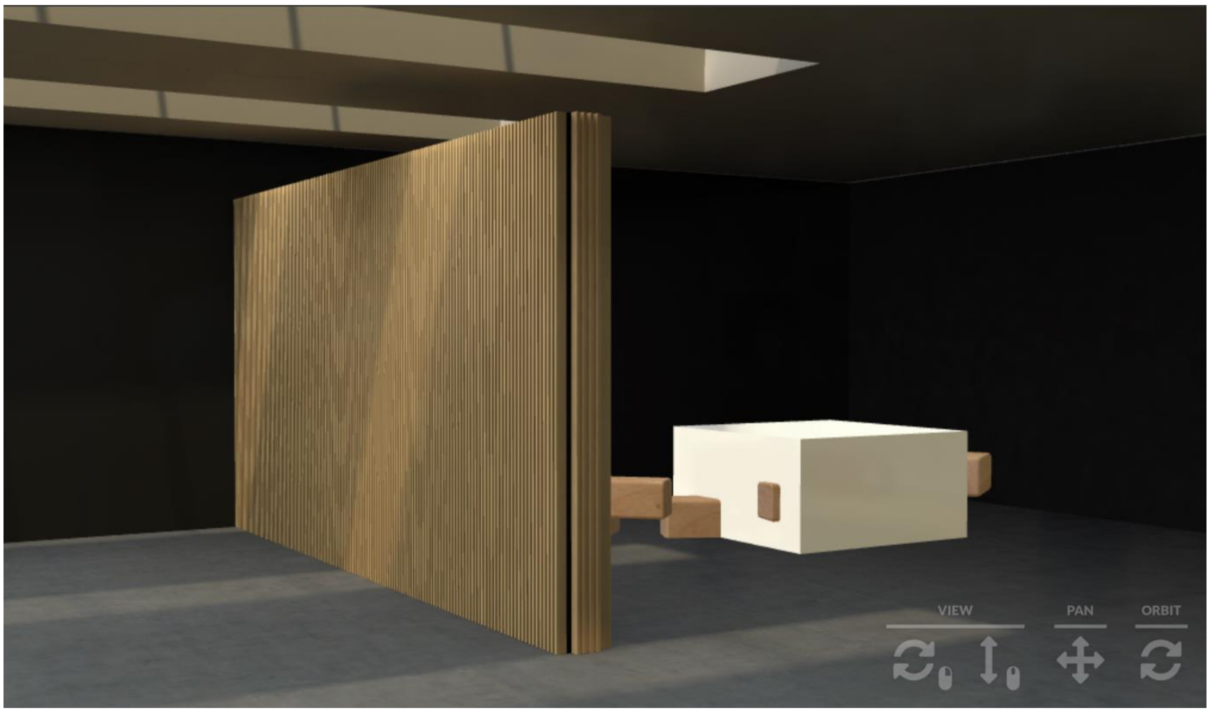
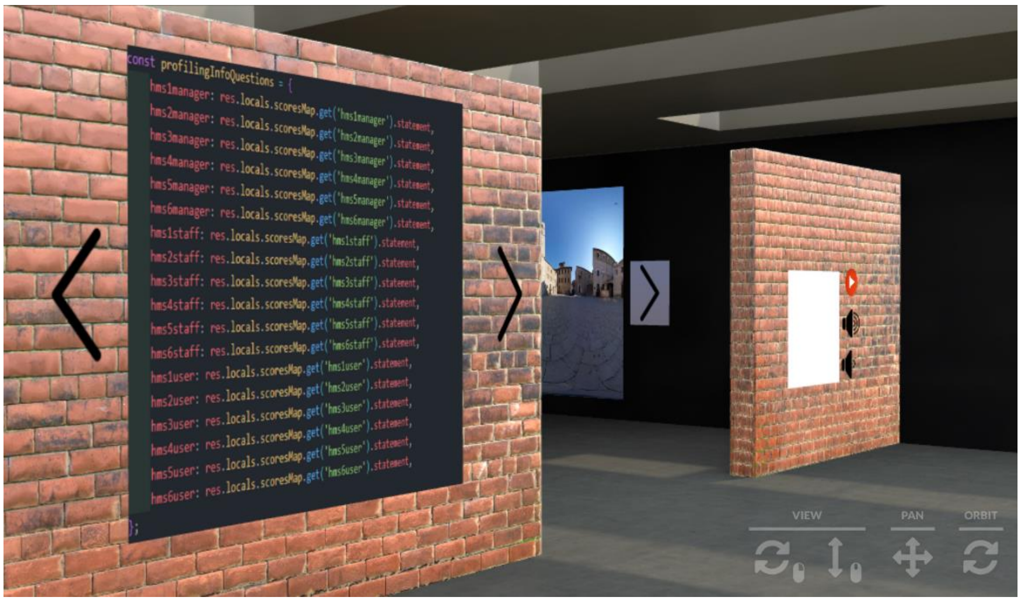


Figure 20 Table and chairs wrongly placed in the room

When I placed the content, I found that the .glb was not working correctly. Some parts of the .glb would overlay the content and every other parts of the room itself. Benches and tables that were placed in the room as part of the .glb file, were not placed where they were supposed to be. Because of these problems, we decided to remove those components from the .glb file. The wooden walls were also causing problems, depending on the layering mode they would either overlay the content and other parts of the wall, or the entire room would cause problems for the png's used. To solve this, I used 3D models from sketchfab to wrap these walls so that they don't overlay the content in the room itself. I used the same layering mode on All the content and the 3D model walls



```
const profilingInfoQuestions = {
    hms1manager: res.locals.scoresMap.get('hms1manager').statement,
    hms2manager: res.locals.scoresMap.get('hms2manager').statement,
    hms3manager: res.locals.scoresMap.get('hms3manager').statement,
    hms4manager: res.locals.scoresMap.get('hms4manager').statement,
    hms5manager: res.locals.scoresMap.get('hms5manager').statement,
    hms6manager: res.locals.scoresMap.get('hms6manager').statement,
    hms1staff: res.locals.scoresMap.get('hms1staff').statement,
    hms2staff: res.locals.scoresMap.get('hms2staff').statement,
    hms3staff: res.locals.scoresMap.get('hms3staff').statement,
    hms4staff: res.locals.scoresMap.get('hms4staff').statement,
    hms5staff: res.locals.scoresMap.get('hms5staff').statement,
    hms6staff: res.locals.scoresMap.get('hms6staff').statement,
    hms1user: res.locals.scoresMap.get('hms1user').statement,
    hms2user: res.locals.scoresMap.get('hms2user').statement,
    hms3user: res.locals.scoresMap.get('hms3user').statement,
    hms4user: res.locals.scoresMap.get('hms4user').statement,
    hms5user: res.locals.scoresMap.get('hms5user').statement,
    hms6user: res.locals.scoresMap.get('hms6user').statement,
};
```

Figure 21 Room with content and wrapped walls

If you look in the back, you can see that the png of the forward arrow is not working correctly. This is fixed when previewing or publishing the application. The layering mode used in this example can only be applied by using a script, this is only executed when the scene is loaded.

Before we fixed the .glb, We thought it might be best to use a different method for the room. This by creating the room ourselves, this way, we would only need one layering mode for the whole scene. I created a room using planes as walls. In this example I used colors like pink and blue to have a clear distinction between the walls and the floor. If this were to be used in the field, you can make use of models or a picture to use as a material to make it more appealing.

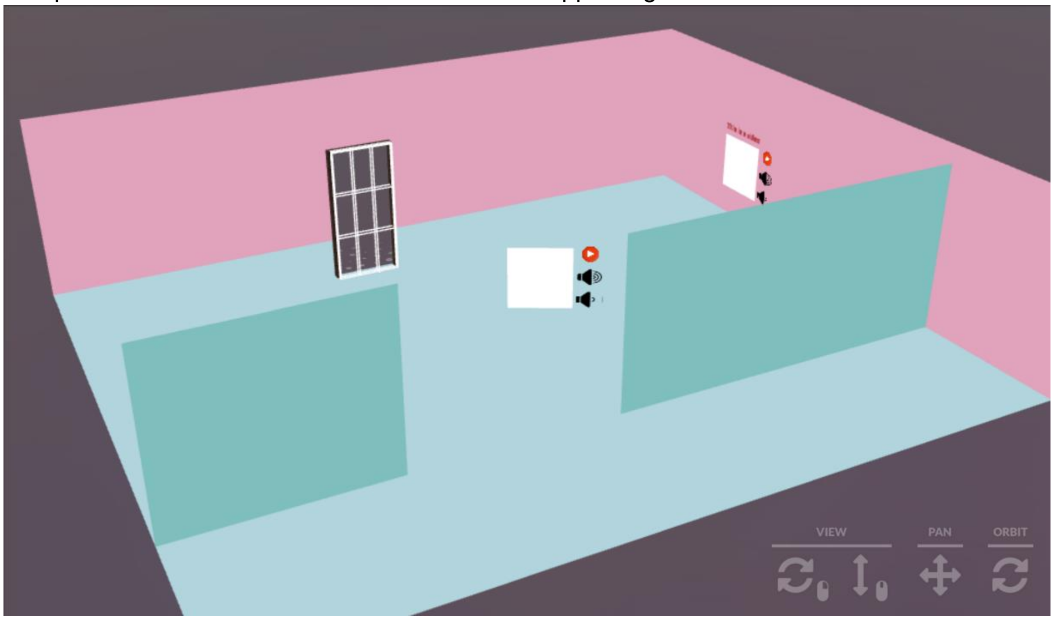


Figure 22 Self made room

3. Conclusion

My international internship has taught me a lot. I got the opportunity to work with technologies I had not worked with before. I completed two assignments, the Protocol Daemon as a whole project, and I completed my part of the portals and content for TOCC. These projects not only taught me technical skills but also soft skills, like professional communication. During the internship I made good friends and experienced new cultures. I want to thank Thomas More for giving me the opportunity of an international internship, and also Egina and ChrackLab F4D for having me as their intern.

Thank you for reading my bachelor thesis.