Rrroomsign

By:

Maarten Marcusse -s2584565
Bram Ouwerkerk - s2819503
Niels Rotmensen - s2291150
Mart Spil - s2618559

Supervisors:
Bernd Meijerink
Roland van Rijswijk-Deij

UNIVERSITY OF TWENTE.

Table of contents

Introduction

The DACS department has a problem where there are too few rooms available in Zilvering. Because of the lack of an intuitive system that can help them create organization in the chaos, they are currently using creative solutions to tackle the problem. For example they shift their availability sign between available and not available to indicate the room is free to use. They came up with the idea of digital room signs with an e-ink display and microcontroller that can dynamically show the room's occupants. It was up to us to implement and extend this idea by listing requirements and ideas, determining what is possible within the given timeframe and executing the idea and testing the result. In this design report we will explain our thought process, the design choices we made and how our system came to be.

Requirements

Before we could start designing and implementing the system we had to know what the stakeholders actually wanted. A meeting was planned and we listed the requirements that have been written in the MOSCOW model. The requirements were collected from the following stakeholders:

- Our project supervisors
- The administration team

We define the following roles:

- User: A person who has a room which has a name sign in the Zilverling.

The project must contain the following requirements:

- As a user I want that my name is displayed on the roomsign if I'm one of the assigned people to that room.
- As a user I want to be able to let the software know whether or not I'm present and if other people can use the room.
- As a user I want to be able to upload a CalDAV file to the software.
- As a user I want that the software can determine based on the uploaded CalDAV files whether or not I'm present.
- As a user I want that it is being displayed on the roomsign whether or not I'm present and if other people can use the room.
- As a user I want that the DACS and edge logo are displayed on the roomsign.
- As a user I want that data about me which can be retrieved from LDAP is done automatically.
- As a user I want to be able to use the software from home.
- As a user I want to authenticate in the software with my UT account.

The project should have the following requirements:

- As a user I want the font on the roomsign to be the official UT font.
- As a user I want to be able to label the CalDAV file that I upload as either education or meeting
- As a user I want that the end time of an event which makes me unavailable is displayed on the roomsign

The project could have the following requirements

- As a user I want to input a custom message in the software.
- As a user I want custom messages to be displayed on the roomsign.
- As a user I want the colors on the roomsign to be colorblind friendly.
- As a user I want the roomsign to be efficient with power consumption.
- As a user I want the software to be able to integrate with Mattermost.
- As a user I want the roomsign to display its battery percentage.
- As a user I want that a roomsign which belongs to a bookable room displays a QR code which directs you to a place where you can book it.
- As a user I want to be able to book a bookable room through the software.

- As a user I want that the education calendar has priority over the meeting calendar when being displayed.
- As a user I want the secretary to be able to see whether or not I’m teaching right now
- As a user I want a small picture being placed next to my name on the roomsign
- As a user I want to be able to simulate the roomsign on a smaller format.

With these requirements we could put together a project proposal that determines what is actually planned to be in the project and can be seen as some sort of contract between us and the project supervisors. However in consultation with each other things can still be changed.

Design

Frontend

The frontend is the main way of interacting with the system for the user. It allows users to change their availability status, change their display name, and more. For administrators, it features ways to change all roomsigns and users. More on that later.

The frontend was built using plain old HTML, CSS and JavaScript, with some of the HTML being generated server-side by Thymeleaf. We chose this structure due to the low maintenance that plain html/css/js requires, and due to the ease of adding new features in the future.

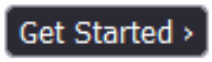
Most of the app is mobile friendly, making use of standard css technologies such as flexbox, css grid and media queries. The administration side of things is not made for mobile, except for one page. The page in question is the link you get from scanning a qr code to link, more on that later.

Styling

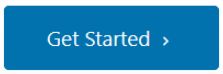
For our styling, we chose PicoCSS. This is a minimal and class-less library, which means that it overrides all regular browser styles. So for example, the following code:

```
<button>Get Started &rsaquo;</button>
```

Will normally produce a button that looks like this



but simply loading the PicoCSS stylesheets will transform the button into this:



This makes it incredibly easy to build nice and consistent looking pages, and makes it easy for future developers to add onto the frontend. Next to that, a classless library will force the developer into following semantic HTML. Using semantic html will give more information about certain elements, which helps with accessibility.

General structure

Most files are located in `src/main/resources`

HTML files are found in `templates`
Other static files (like css, js, fonts) are in `static/*`

These are all served by a single Java class/controller. This controller receives requests, fetches data needed to show on the page, and then renders the correct html page. This controller is located in `src/main/java/com/roomsign_backend/controller/TemplateController`

Every method is annotated with a `@GetMapping` which tells the framework which route it is. For example `@GetMapping("/admin/unlinked")` goes to the unlinked roomsign administration page.

Thymeleaf

Thymeleaf is a server-side template engine for Java that transforms a thymeleaf file into proper html. We mostly use this to fill out all data on a page before sending it to the client. For example, on the user administration page, there is a big table of users. These are filled in by the backend beforehand, to avoid unnecessary loading screens.

The structure for a request is as follows (to `/admin/unlinked`):

1. The request is received by spring boot and routed to the correct controller
2. This is received by `controller/TemplateController.java` to be processed
3. Here it calls the method `public ModelAndView unlinkedSigns()`
4. This method will then fetch all unlinked roomsigns from the `RoomsignService`
5. This is added to the model of the page
6. Thymeleaf will fetch the template from `resources/templates/admin/unlinked.html` and fill in all unlinked room signs
7. This finished page is sent as a response to the request.

Let us take a look at some of the interesting things behind `unlinked.html`. For this I would recommend opening the code next to it.

Lines 2-4

```
<html
        lang="en"
        th:attr="data-theme=${user.getThemeHTML()}"
        xmlns:th="http://www.thymeleaf.org">
```

This is present on all thymeleaf files, and specifies the current theme (light/dark) in use. This theme is set by our css library (PicoCSS)

Lines 10-14

```
<th:block th:insert="~{fragments :: head}"/>

<script type="text/javascript"
th:src="@{/js/modals.js}"></script>
<script type="text/javascript"
th:src="@{/js/admin.js}"></script>
<link rel="stylesheet" th:href="@{/css/admin/admin.css}"/>
```

This loads in all js/css into the page. Notice the th:block. That loads in a fragment from another file. Let us take a look into that fragment. (located in `resources/templates/fragments` Lines 4-20, shown 10-14)

```
<link rel="stylesheet" th:href="@{/css/basic.css}"/>
<link rel="stylesheet" th:href="@{/css/pico.fuchsia.min.css}">
<link rel="stylesheet" th:href="@{/css/style.css}"/>
<script type="text/javascript"
th:src="@{/js/gql.js}"></script>
<script type="text/javascript"
th:src="@{/js/base.js}"></script>
```

This ‘base’ fragment is loaded in on all pages, and contains the shared css (basic/style.css), css library (pico.), code for handling all GraphQL calls (gql.js) and shared javascript (base.js)

Now moving back to `unlinked.html`

Lines 17-19

```
<nav>
   <th:block th:insert="~{admin/fragments :: header}"/>
</nav>
```

These lines load in another fragment, this time it is responsible for the navbar

Line 22-47

```
<dialog id="modal--link">
```

This defines a dialog, or better known as a modal. This uses the modals.js as shown above. In this case this opens a modal which asks for a room number, to link to this sign. It then stores this in the database.

Line 51

```
<th:block th:insert="~{admin/fragments :: aside}"/>
```

Another fragment, this time the administration sidebar

Lastly, filling the tables in lines 81 until 87

```
<tr th:each="sign : ${signs}">
   <td th:text="${sign.getUuid()}"></td>
   <td>
       <button th:attr="onclick=|openLinkModal('${sign.getUuid()}')|">
        Link
      </button>
       <button
th:attr="onclick=|deleteUnlinkedRoomsign('${sign.getUuid()}')|">
        Delete
      </button>
   </td>
</tr>
```

This is all done server side, and uses the data which we got in `unlinkedSigns()` Every sign gets its own table row (tr), which has two buttons that call the correct javascript function with its UUID. Notice how we can use all java methods that are specified on the Roomsign object.

Backend

The backend handles all the requests for the frontend, it generates images for the frontend and microcontroller, handles the uploaded calendars, authentication and manipulation of the database. The process of handling a request is as follows: The request is handled by the server, this sends it to the correct controller, the controller sends it to a service and the service handles the request and does some parsing and sends it to the process where it stores it in the database (See Appendix C Fig. 8. for the classes).

Spring boot

We decided to use spring boot mainly because it is an industry standard Java backend framework and Maarten was already familiar with the framework and Bram was comfortable in Java as well. As well it was chosen as Java is a widely used language at the university and so in a possible future development it can be picked up more easily.

Security

Spring boot has a lot of security built in. It supports our desired OpenID flow. To implement this we need to add the required client id, client secret, a redirect uri and the scopes. We contacted LISA for this and they provided us with the right information to be able to set up our Microsoft authentication based on OpenID. On the provider site we need to make sure a client is set up with the previous mentioned info. We additionally need to create a security configuration so that we can modify the flow and define which endpoints require authentication. The flow needs to be edited so that we can link our own users with the provided open id users.

We also use role based authorization. This can also be easily done by just adding an annotation with the specified role to the endpoints. We need to add two roles called ROLE_USER and ROLE_ADMIN. All users except the first one automatically become a user. Any admin can make other users admin.

ORM

To be able to store user, roomsign and microcontroller data we need a database. To be able to talk efficiently to this database we will use the Hibernate ORM. An ORM is an object–relational mapping.

Calendars

One of the requirements was to link your outlook calendar and timeedit calendar. In the back-end these calendars are parsed and the events are stored in the database. The events are mapped to the user so that we can update their availability on the

roomsign. The events store the time locally and we did not consider time zones as for this project we expect every user to be in the same time zone. The system checks for updates on the calendar every 10 minutes to keep in account changes into someone's schedule.

Scheduling

When the events are parsed and checked for overlap we schedule jobs based on the end and start times of the events. When a job executes it sets the availability of the user to do not disturb if it is a meeting (outlook), unavailable if it is teaching (TimeEdit) or available if an event ends and they are available again.

Efficiency

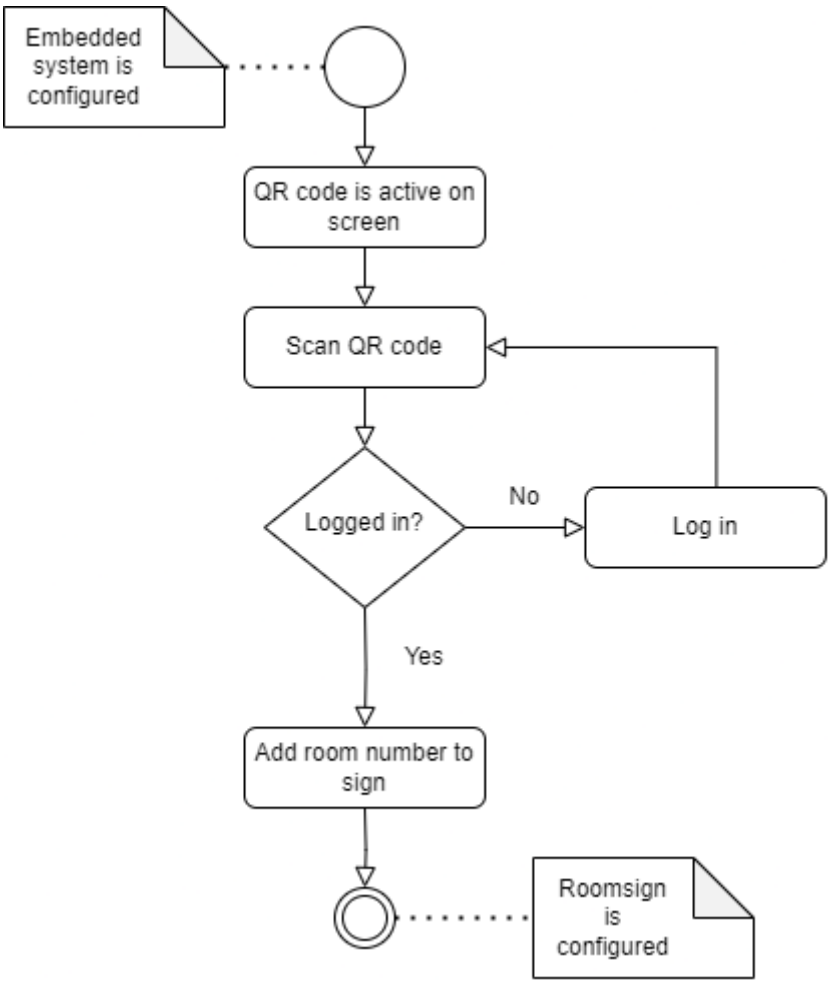
We also considered the amount of events that will be saved in the database when this project would run for multiple users for a longer period of time. To prevent unnecessary accumulation of calendar events we decided to delete events that happened in the past as those would serve no purpose anymore. With this approach we reduce energy consumption (less storage needed) and improve the privacy of our users as we only keep their events as long as necessary.

Image generation

For generating images we decided to use a SVG format. We have a template SVG file which is being edited by the back-end so that it displays the correct users in the correct order, it displays the correct room number and it displays the availability. We decided to choose an SVG template since it can easily be modified, due to the format of being text-like. We save the image as a base64 string in the database. This is done for optimization and easy decoding and encoding. When processing the image for the roomsign we use the same algorithm which is being provided by the original software of the e-paper. We made an implementation of this algorithm in java since the original program is in c++. This algorithm uses the euclidean algorithm to determine the closest color to the input color and writes the color code in 4 bits.

Linking roomsigns

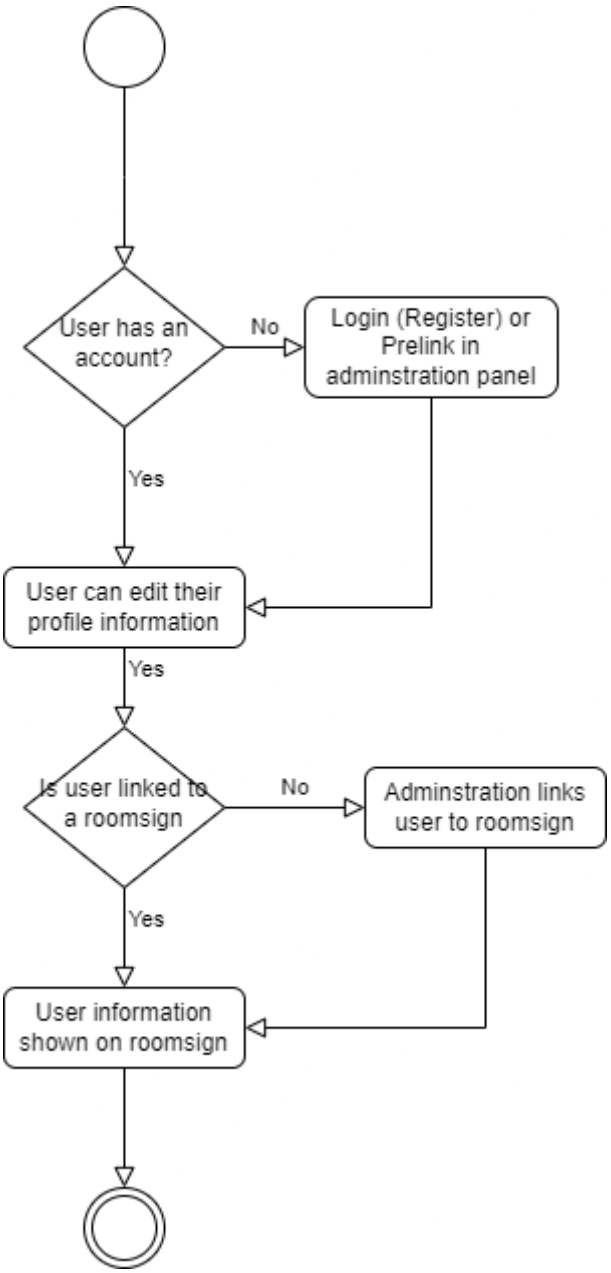
For linking the roomsigns to the server we decided on an approach where the roomsign sends its uuid to the server and the server stores it as an unlinked roomsign. The roomsign will display a QR with a link containing its uuid and this link is used for connecting the roomsign and setting it as active on the website and giving it a room number (See Fig. 1.). After this the administration can add users to the active sign.



(Fig. 1. Link a roomsign to the server)

Users

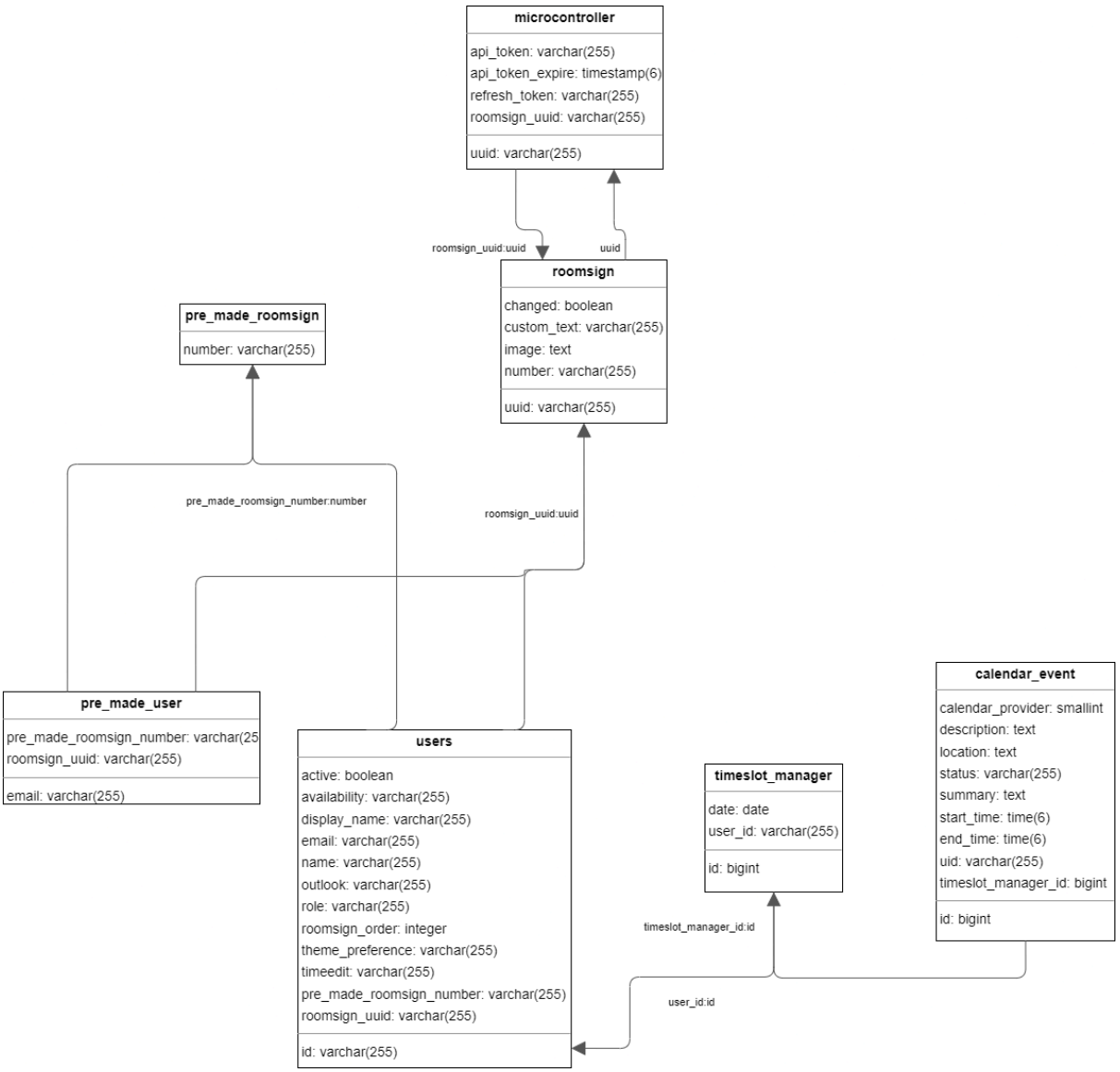
Users are authenticated using Microsoft Authentication with OpenID as according to the requirement so that users can log in with their Utwente account. Once they login we use their email address to create their profile and set their display name as the display name taken from the authentication system. A user can then add their calendar or change their display name in their account. Once the administration linked them to a room their information shows up on the respective roomsign (See Fig. 2.).



(Fig. 2. User login to user information on roomsign)

Database design

To store data of our users we needed to come up with a database design. The requirements are that it needs to store our user information like their email and their calendar links. Also we need to store our roomsigns with their generated image. We need to keep track of our connected microcontrollers so that it does not need to be re-configured every time it tries to connect to the server. At last we need to store the uploaded calendars of our users so that we do not need to parse the whole calendar every time something changes to the board and so that we can display what event is happening and schedule events for showing the availability of a user. In figure 3 we made an overview of the database.



(Fig. 3. Database design)

Error handling

We decided on adding custom error handling to define errors in our application. When they are thrown we can give the user some specific information for what went wrong and these are shown on the front-end (See Appendix C Fig. 7.).
We defined the following errors:

IMAGE001	Failed to load the template image
IMAGE002	Failed to convert the template image
USER001	Failed to load the authenticated user
USER002	Failed to find an user with the provided id
USER003	You tried to remove the last admin and

	there must be at least one admin in the system.
USER004	User is inactive.
USER005	There is no space for the user on the roomsign
USER006	Editing user gone wrong
USER007	No user or premade user can be found under this email
USER008	Another user already uses this calendar link
TIME001	Failed to find associated TimeSlotManager with provided user
ROOM001	The provided uuid is not registered
ROOM002	There is no room linked to this user
ROOM003	The provided uuid should be registered but is not
ROOM004	There is no user linked to this room
ROOM005	There is a duplicate order index for this room
ROOM006	The order must be between 1 and 6
CALDAV001	The provided caldav link is not properly formatted
CALDAV002	The provided caldav link is malformed or can not be opened
CALDAV003	Something went wrong with scheduling an appointment

GraphQL

For the connection between the front-end and back-end we use GraphQL. With GraphQL we could achieve a faster development process as the front-end can ask the specific information it needs from the back-end without specific APIs needed to be created for each specific case. Also with graphQL we can have clear

documentation to work without needing to write a lot ourselves (See Appendix A Fig. 5. and Fig. 6.).

Hardware

The hardware used for this project consisted of an esp32[1], an e-paper display[2] and the connectors for these devices. Our supervisor provided this hardware and we didn't choose to expand upon it.

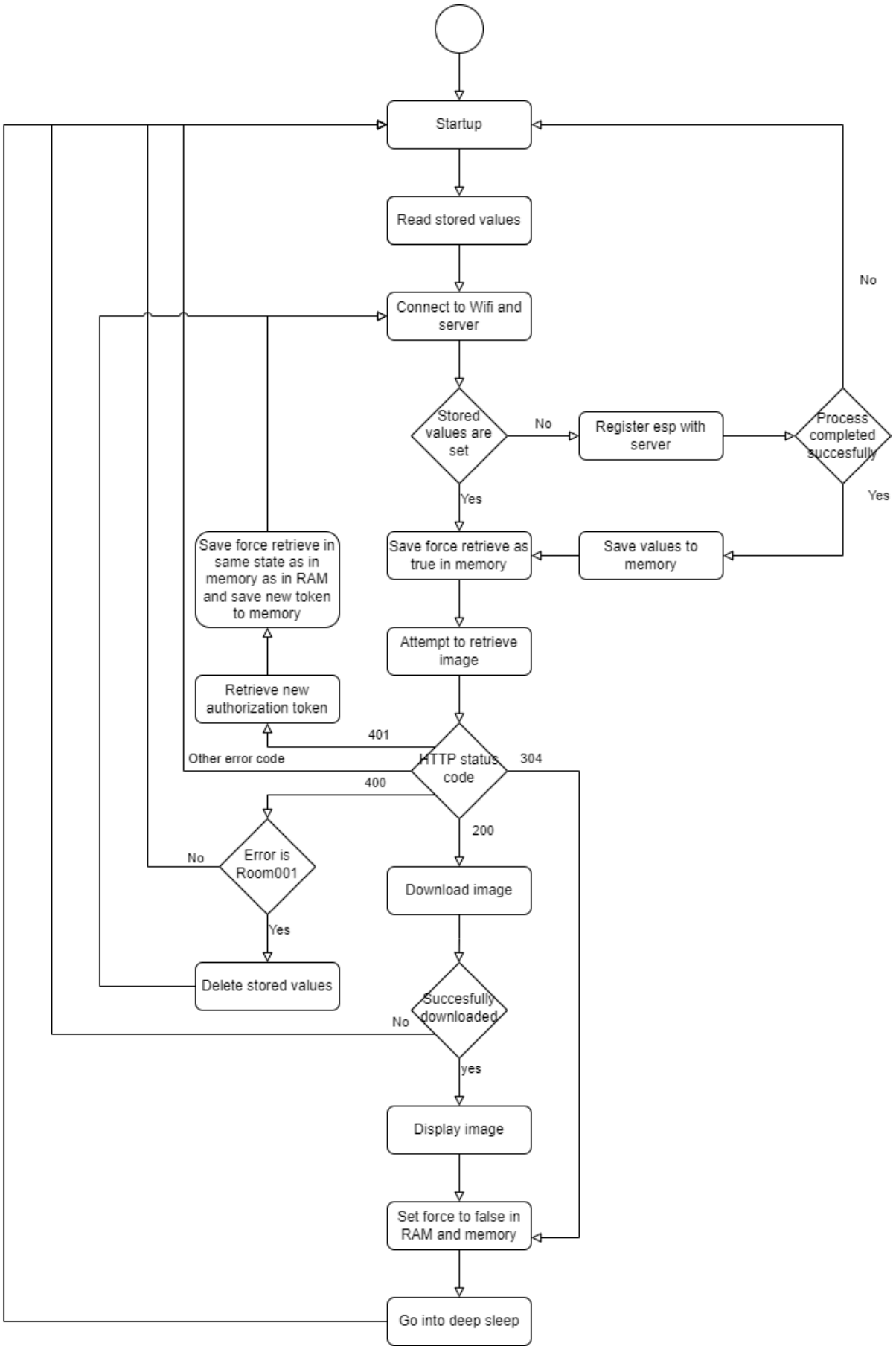
The esp32 is used to get the image from the backend and then display it on the e-paper. C++ was chosen as the language for three primary reasons.
First of all the display drivers were written in the language.
Second, the default language of the esp32 is C++.
Third is that the esp32 has limited ram and there was very little to spare with the wifi module and dealing with the image.

In figure 4 you can find the logic of the esp32.

The esp32 RAM can not be relied on to stay consistent while functioning as the deepsleep turns it off and when connecting to the wifi the module often needs to restart. For this reason the main variables get written to memory and read again at boot.

The esp32 should be able to work without any intervention for a long time. For this reason a design was chosen that if it failed anywhere it would go back to a safe state. This safe state is a reboot. Here it will read it's values from memory and try to continue with the failed step.

[1] https://www.waveshare.com/wiki/E-Paper_ESP32_Driver_Board
[2] https://www.waveshare.com/wiki/7.3inch_e-Paper_HAT_(F)_Manual



(Fig. 4. Microcontroller logic)

API

To communicate between the microcontroller and the server we host an api on the server. First off we have a function to register a new microcontroller, where the microcontroller provides a uuid and the server returns an authorization code and a refresh code and the microcontroller will be added to the database. Then the microcontroller can request the image from the server using the authorization code, where there server returns a byte array which can then be displayed on the e-paper. For security we want to keep the authorization code fresh and to do this we have the last api endpoint with which the code can be refreshed once it has expired.

Implementation trajectory

In our project proposal we made a planning for us and our supervisors (See Appendix D: Planning). In this we showed our trajectory for features to be implemented. We were planning on adding the features of importance 1 and 2 but only a subset of these were added. We did manage to implement custom text and calendar priority. We also looked into faster displaying but due to hardware limitations of our e-ink screen and the respective controller it was not possible to do partial rendering.

Risk analysis

We defined some risks that needed to be thought of that could impact the end result and that thus should be thought of in this project. These risks are divided into three categories: low, medium and high.

Low risk
Scope creep: It should be avoided to add unnecessary features that do not work towards the main goal and add to much complexity. Features should have an estimated time of required work so that in the end there won't be time stress because of too many added features.
Communication: communication should be clear and deadlines should be disclosed clearly.

Medium risk
Hardware failure: The provided hardware should be handled with care to avoid unnecessary defects.
Planning: An unclear planning might result in unfinished or rushed work.

High risk
Data security: user data needs to be stored safely and shouldn't be accessed by unauthorized users. This can be avoided by implementing security measures across all parts of the application (Front-end, back-end, embedded).

Responsibilities

Frontend	Niels Rotmensen
Backend	Maarten Marcusse (Focus on image generation) Bram Ouwerkerk (Focus on calendar integration)
Embedded (Microcontroller + e-ink display)	Mart Spil

We considered three main branches for the development part: front-end, back-end and the embedded part. As the back-end would require the most work we decided that Maarten and Bram would work together on that.
Besides this we all worked on the design report, presentations and Bram made the poster design.

Deployment

The application runs on a VM reachable on roomsign.student.dacs.utwente.nl

This virtual machine contains a few services, namely:

- Caddy
- Docker
    - Portainer
    - PostgreSQL
    - The roomsign application

Caddy

Caddy is a web server and application platform that we use as a reverse proxy. When you connect to the url of our application, you will not send the request to the application server directly, but to caddy. Caddy will then forward the request to the application server, and send back the response. Doing this has a few advantages.

First of all, caddy manages SSL certificates and forces HTTPS. The application only needs to respond in HTTP. This lowers complexity. It will automatically refresh SSL certificates when they expire. For this they use LetsEncrypt. Next to that, Caddy also uses gzip or zstd as compression. This reduces the size of the response.

This is a sample Caddyfile, which is a simplified version of what is used in production. Note the `handle_path` declaration, which tells caddy to redirect all requests to */docker* to the portainer instance.

Sample Caddyfile

```
Unset
(common) {
    encode {
        gzip
        zstd
    }
}

roomsign.student.dacs.utwente.nl {
    import common

    # Otherwise portainer cant handle the request
    redir /docker /docker/

    # Portainer
    handle_path /docker/* {
        rewrite /docker/* /

        reverse_proxy http://localhost:9000
    }

    reverse_proxy http://localhost:8080
}
```

Portainer

To manage the docker container, we use portainer. This allows us to easily update the roomsign application to a new version. This is because portainer allows us to bump a version, without forgetting previous environment variables and other configuration.

Deploying a new version

For deploying a new version, refer to the *README.md* file in the repository.

Testing

For testing we decided to continuously test our new features through integration tests and reviewing each other’s work when they added something to the code.

Endpoint tests

Each endpoint got tested after a change got made in the related services and processes. For the API endpoints we used postman. We had 3 requests to test after changes had been made to the microcontroller services and processes. Some basic test requests are used to test if the behavior is still as intended.

For the graphql endpoint we used graphiql to test the endpoints. With this we could test all mutations and queries and see the result. Just as with the API whenever a change was made to a relevant service or process the mutations and queries got tested with some basic information.

Integration tests

The endpoint tests also allowed simulating functionality that was not yet implemented. This allowed the microcontroller to test functionality that would be implemented in the front end without the front end being made and the other way around. These factors made it so that when we were combining the different parts of our project there were mainly only problems with the image generation as this could not be tested without the full basic functionality implemented. In the end we also let the system work for multiple hours continuously and it did so without any significant problems.

Discussion

Planning

There were some things that could have gone better. The major one that impacted the project the most was a planning that was not held strictly enough and a lack of leadership. In the beginning we decided what parts everyone would work on for the coming weeks. We had a meeting every week with our supervisors and after these meetings we worked on the project and decided what everyone would work on towards the next meeting. This was positive and helped us in achieving our final result but during these times between meetings it was not fully clear to every group member what everyone was exactly working on. Something that could have helped was software like Trello or Github Projects. That way it would be more clear to everyone what was being done and this was only one step away as we did use Github issues to track features that needed to be built and bugs that needed to be resolved. In addition to this, assigning a leader at the start of the project could have helped in keeping better up to schedule with the initial planning as sometimes the direction of the project felt like a rudderless ship with no clear direction. However we do need to state that although there was a lack of leadership the teamwork was great and we helped each other out along the way.

Design

Another thing that could have benefited the project was an initial design of the front-end. We decided at the beginning not to do this as it would take some time to create and that time was better spent implementing the front-end. This way we could iterate faster through the designs and because we had a meeting every week we got feedback continuously from our stakeholders that we could use to improve the design. However, a first design could have been helpful as it gives a basic impression for the stakeholders on what to expect as well as that it gives us a clear view of the features that would be needed to be implemented for it to function as a whole.

Future

Not all requirements have been implemented, since they fell out of scope for this project, but if one wants to work on this project in the future, there are some things we would recommend.

First of all, the roomsign does not have any casing. This casing would ideally contain the sign, the microcontroller and some way to power the sign. This can be mains power, or ideally a battery.

That leads us to the second improvement, a battery. Deep sleep to improve battery saving is already implemented, but since the screen can only be completely changed and the image can not be stored in the microcontroller no battery level can be displayed.

Another feature that can be added to the system is a room booking system. Since there is already information about if a room is occupied, tacking on a room reservation system will not be much work.

Our product also doesn’t feature dithering which makes it harder to display exact colours, future versions could implement this to allow for more colour options and to display images.

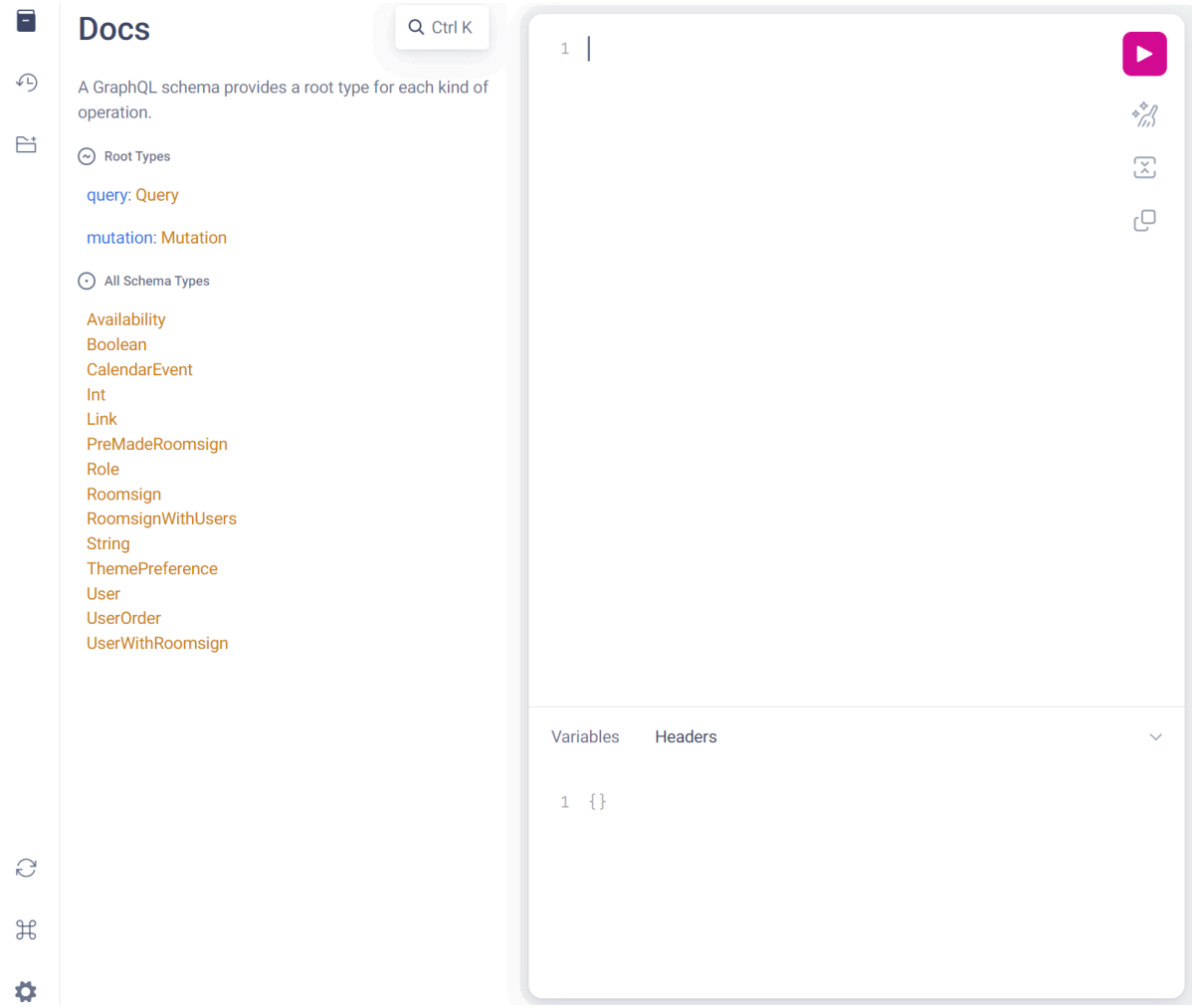
Conclusion

At the start of the project we listed all the requirements of our stakeholders and we wrote a project proposal to our supervisors. We divided the different responsibilities and knew what we expected from each other. As we had meetings every week with our supervisors it was clear to both sides where the project was heading and what needed to be changed. We iterated with front-end designs and added the features that were needed in the back-end. We tested our product continuously throughout our process to make sure our features were working and bugs were found and resolved. Because of this in the end we achieved a satisfactory result and the requirements were met. The project is a good example of what is possible with a system like ours and we listed how it can be improved and added upon in the future. We are pleased with our end result and how we worked together within our group and with our client. Their enthusiasm really motivated us to put a lot of effort into the project and make it a success.

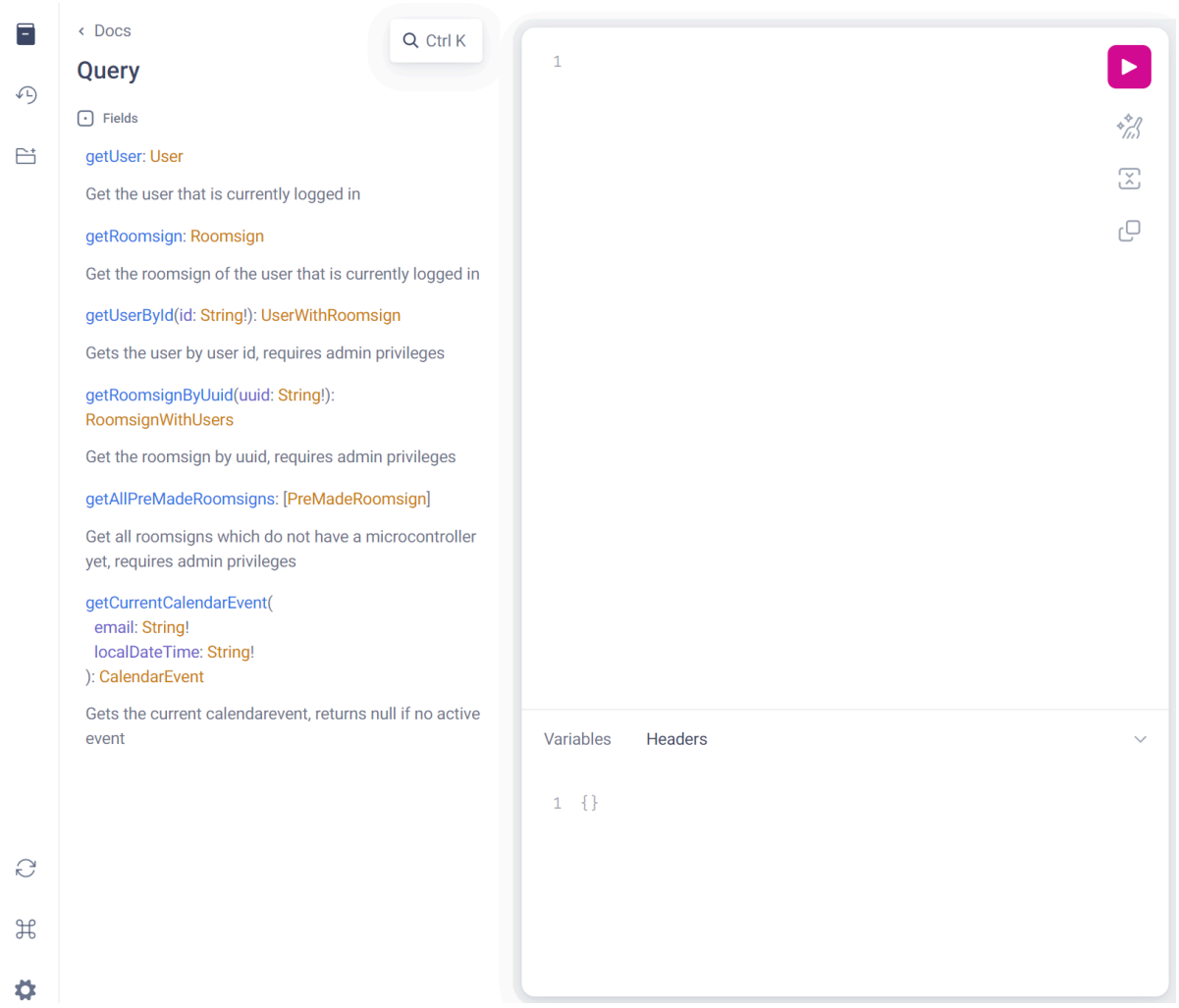
The project website can be found at https://roomsign.student.dacs.utwente.nl/ (As long as the vm is still hosted). And one can login with a UT test account (Authentication is valid until 06-01-2025) ending on .net .

Appendix

Appendix A: GraphQL

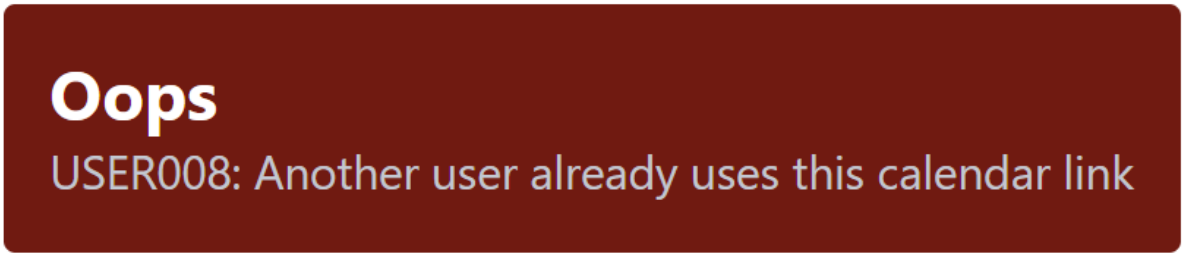


(Fig. 5. GraphQL Documentation)



(Fig. 6. GraphQL Query Documentation)

Appendix B: Error handling



(Fig. 7. An error that gets shown on the front-end if the same calendar link is used)

Appendix C: Back-end

(Fig. 8. Class diagram of the back-end)

Appendix D: Planning

End of week (sunday)	**What to do**
Week 1 (02-09 , 08-09)	Start back-end MVP
Week 2 (09-09 , 15-09)	Start front-end MVP
Week 3 (16-09 , 22-09)	Back-end endpoints finished, Project Proposal + back-end design
Week 4 (23-09 , 29-09)	MVP implemented
Week 5 (30-09 , 06-10)	Demo of MVP during meeting with the supervisors Integration testing of MVP Starting on next version MVP (importance 1)
Week 6 (07-10 , 13-10)	Next version MVP (importance 1)
Week 7 (14-10 , 20-10)	Start on next version MVP (importance 2)
Week 8 (21-10 , 27-10)	MVP (importance 2) done and integration testing
Week 9 (28-10 , 03-11)	Adding last unit tests and integration tests, finishing report
Week 10 (04-11 , 10-11)	Presentation and finishing report

Could have features

Feature	**Importance (Lower is more important)**
Faster displaying	2
Power efficiency	4
Power microcontroller and display by battery	3
Link to mattermost	5
Display battery percentage	5
QR code to book a room	2
Book rooms in the front-end	2
Calendar priority (For links and uploaded files)	1

The secretary to be able to see whether or not the user is teaching right now	1
Picture of user on the display	1
Simulate room sign on a smaller format	6
Custom messages	1

Appendix E: User manual

Regular user

To access the service you can head to https://roomsign.student.dacs.utwente.nl

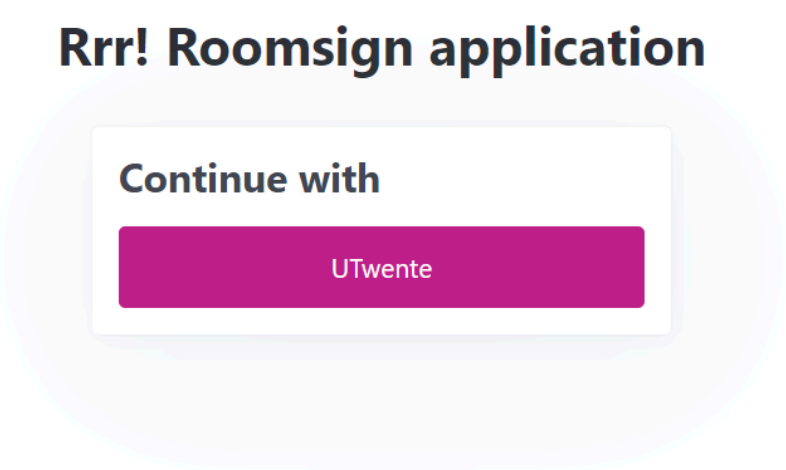


Figure 1: Login screen

After your first time logging in, you will be sent to the account edit page. This is because no roomsign has been assigned to you yet. On this page you can add calendars to synchronize, and change your display name. If set, this display name is used for generating the roomsign and will be used instead of your full name.

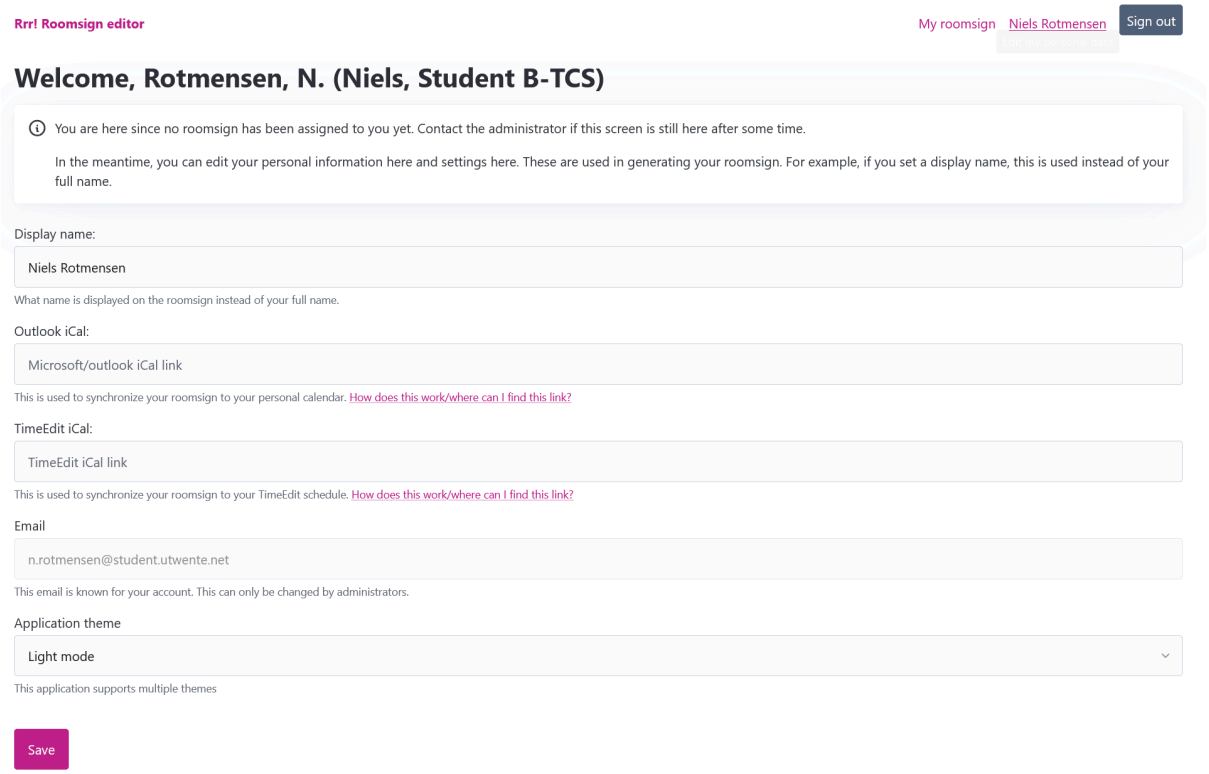


Figure 2: Edit your account details

After an administrator has added you to a room, you can edit your roomsign and availability. This is also the main screen when you login in the future.

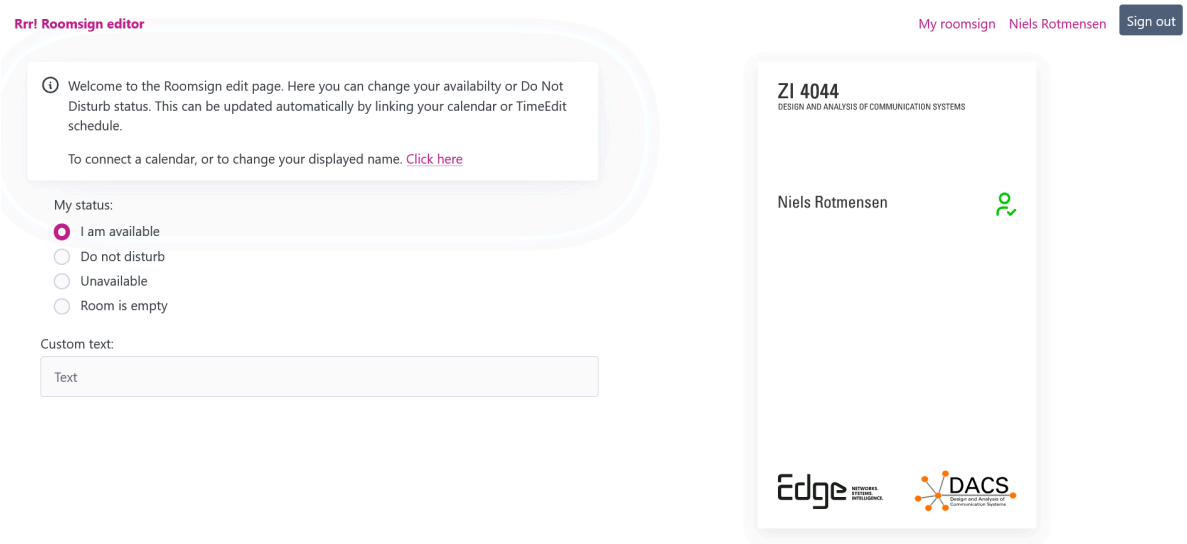


Figure 3: Main screen for editing your roomsign.

On this page you can edit your availability with the radio buttons on the left side. When you edit your status, this will also trigger a re-render of the roomsign itself.

There is also the *Custom Text* feature, which allows you to put some extra text on the sign.

Synchronizing your calendar

A feature of our system is the ability to connect an Outlook and/or TimeEdit calendar. To do this, go to the account management page, as shown in *figure 2* and fill in a proper iCal link.

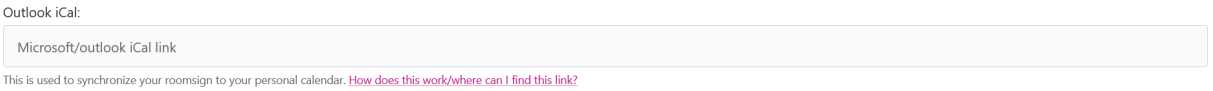


Figure 4: iCal edit field

To find this iCal link, you can use the pink link shown in *figure 4*. That will lead you to the help page (shown below) for the calendar you want to add. On this page you will find a step-by-step guide for adding the calendar.

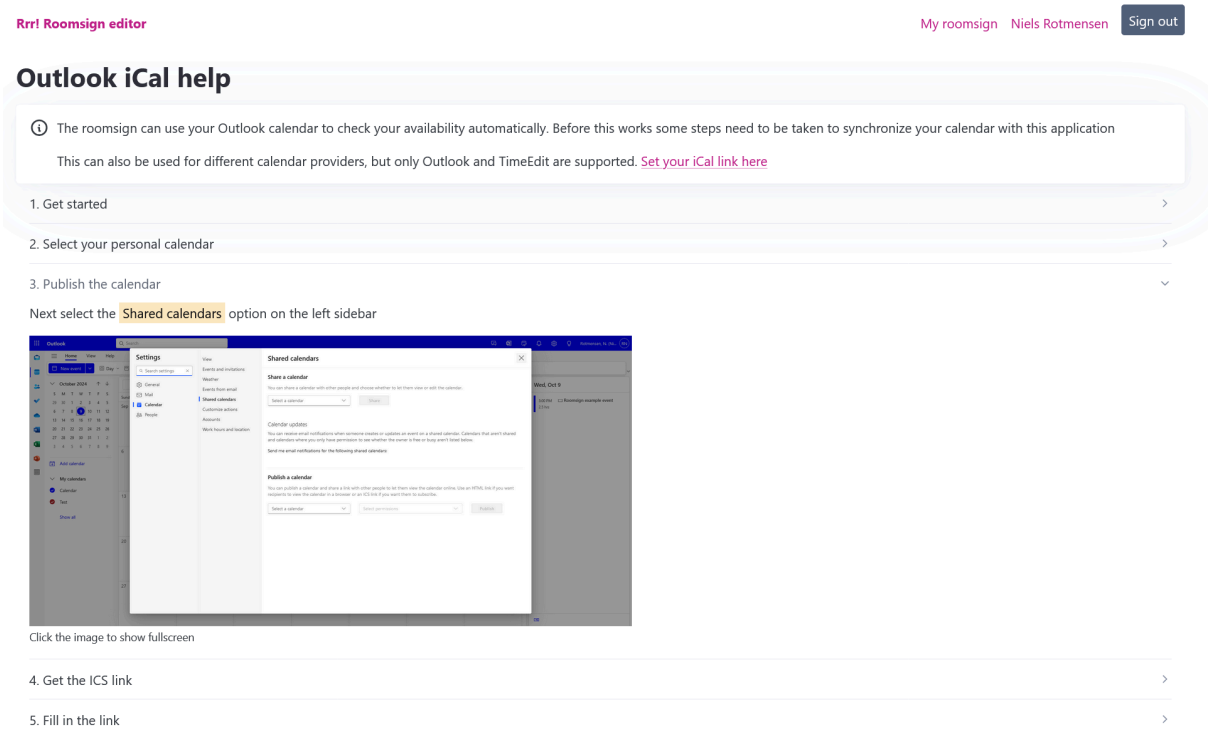


Figure 5: Outlook help page (you can click on an image to enlarge it)

Administration

If you are an administrator your base profile will be the same as a user, but you have an additional administration tab.
All actions described here can be found under the administration tab.

Figure 6: Navigation bar for administrators

Connecting a roomsign to the system

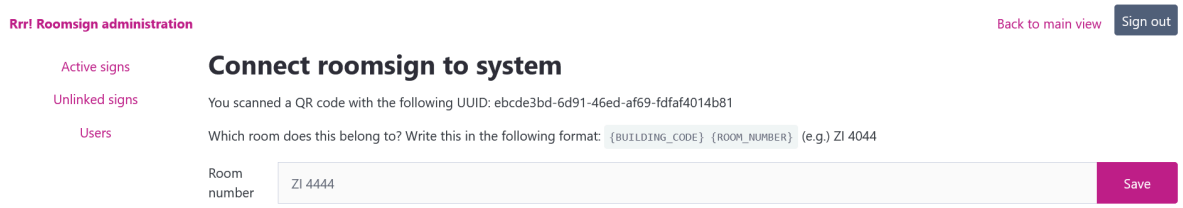


Figure 7: Screen shown when scanning a QR code

When you connect a roomsign microcontroller to power, it will automatically connect to wifi and will display a QR code. This QR code will lead you to the roomsign linking page. On this page there is only one input, to give it a room number. For example if the roomsign hangs at office *ZI 4044,* you input *ZI 4044*. Clicking save will activate the microcontroller, and after a few minutes the roomsign will show the correct room number instead of a QR code.

Alternatively you can search for the board in unlinked signs (the ID matches the identifier in the url of the qr).

Another way of activating microcontrollers is to use the unlinked page. You can find a link to this page on the left side of the page.

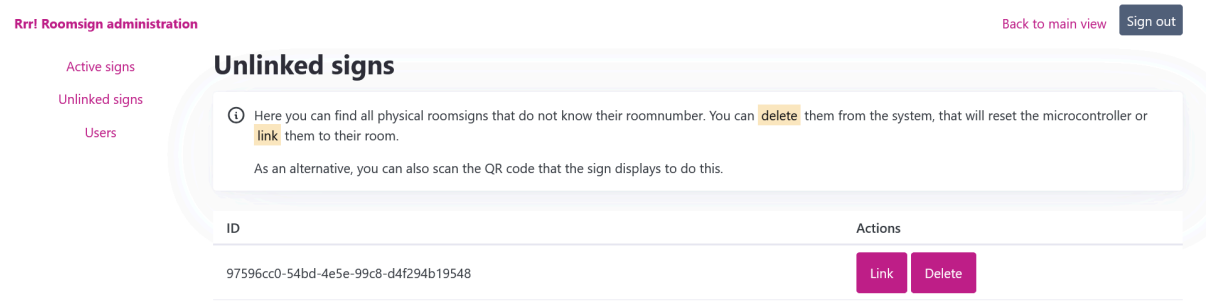


Figure 8: Unlinked signs page

Clicking on the *Link* button will bring up a dialog to fill in a room number.

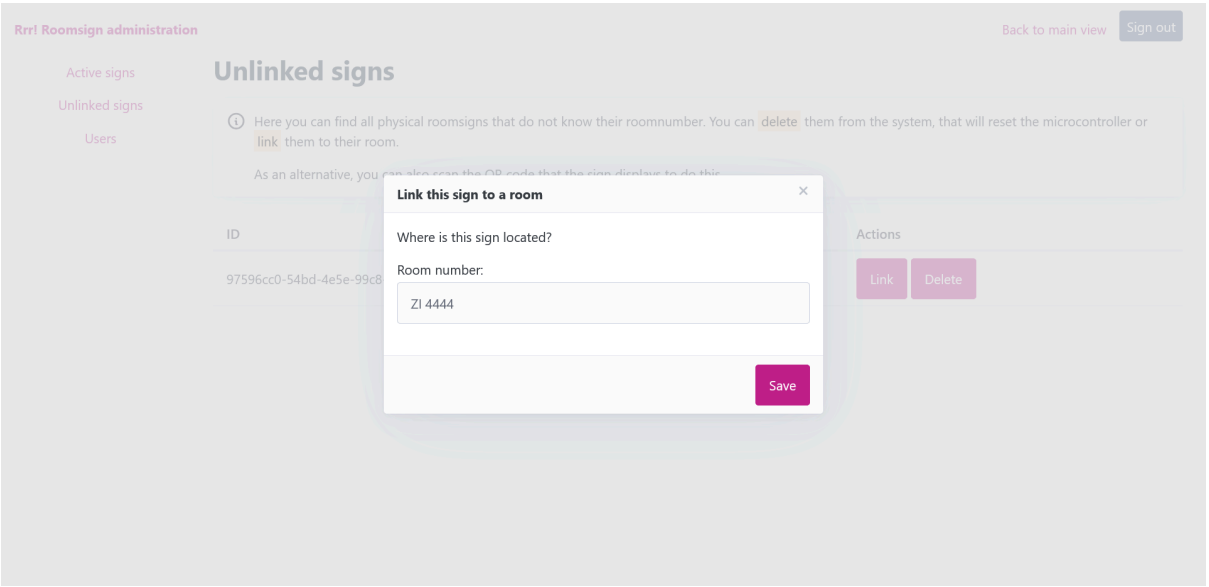


Figure 9: Link roomsign dialog

You can also click delete, this will delete all mentions of that microcontroller from the system, and will reset the microcontroller.

Managing existing roomsigns

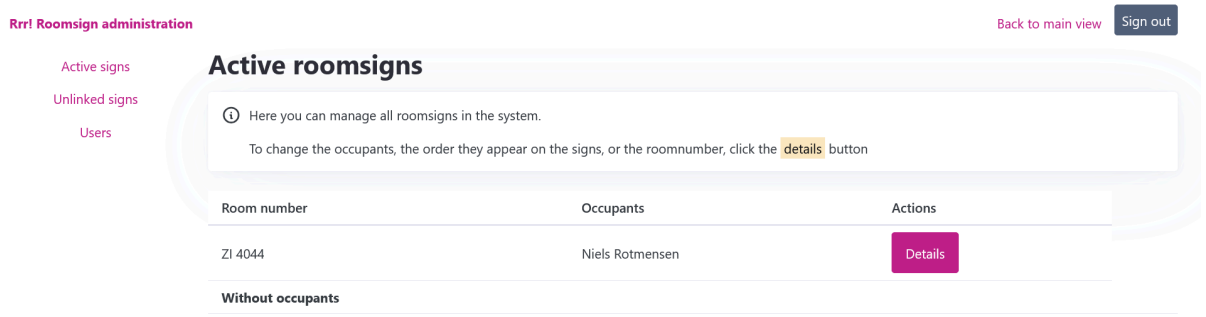


Figure 10: Active roomsigns

On */admin/overview* you can find an overview of all existing roomsigns. Clicking on the details button will bring you to the roomsign detail page.

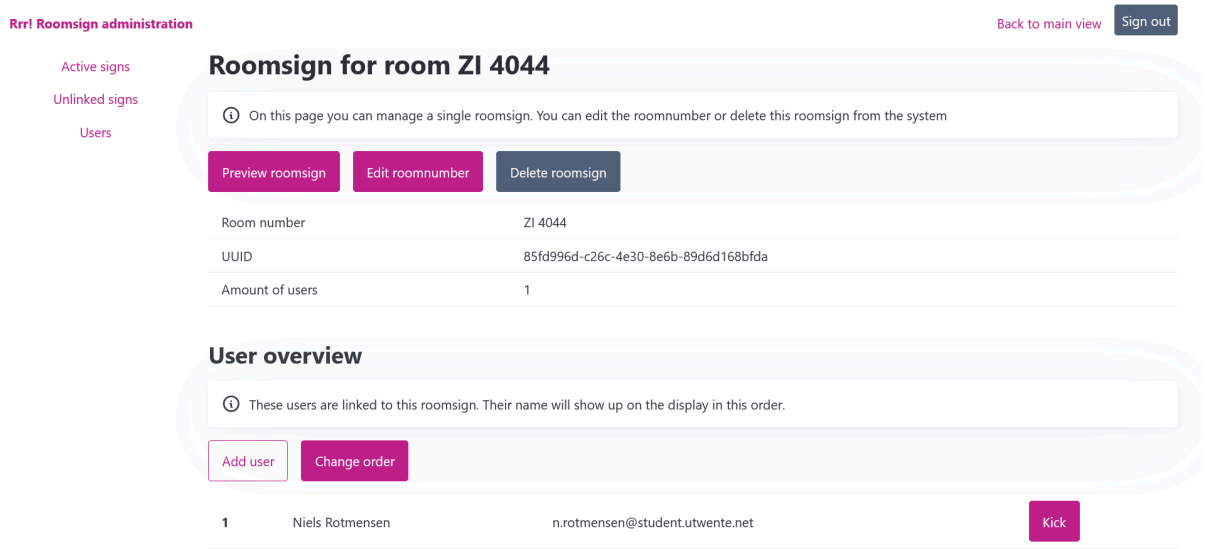


Figure 11: Roomsign detail page

This page features a lot of options for managing your roomsign. Such as viewing a preview (figure 12), editing its room number (figure 13), adding an user (figure 14), or changing the order the users are shown (figure 15)

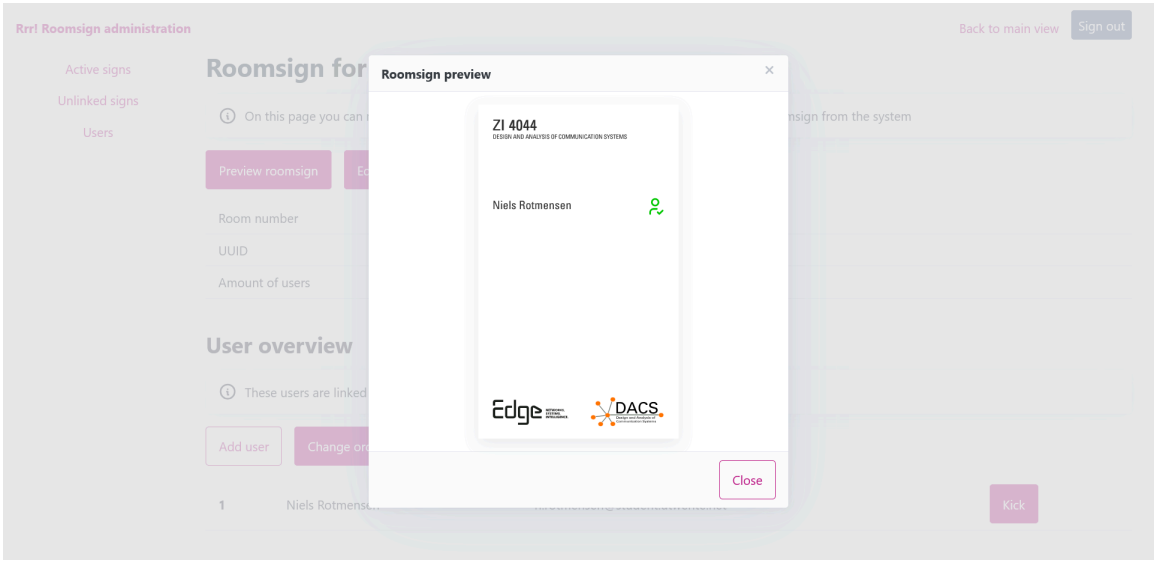


Figure 12: Roomsign preview

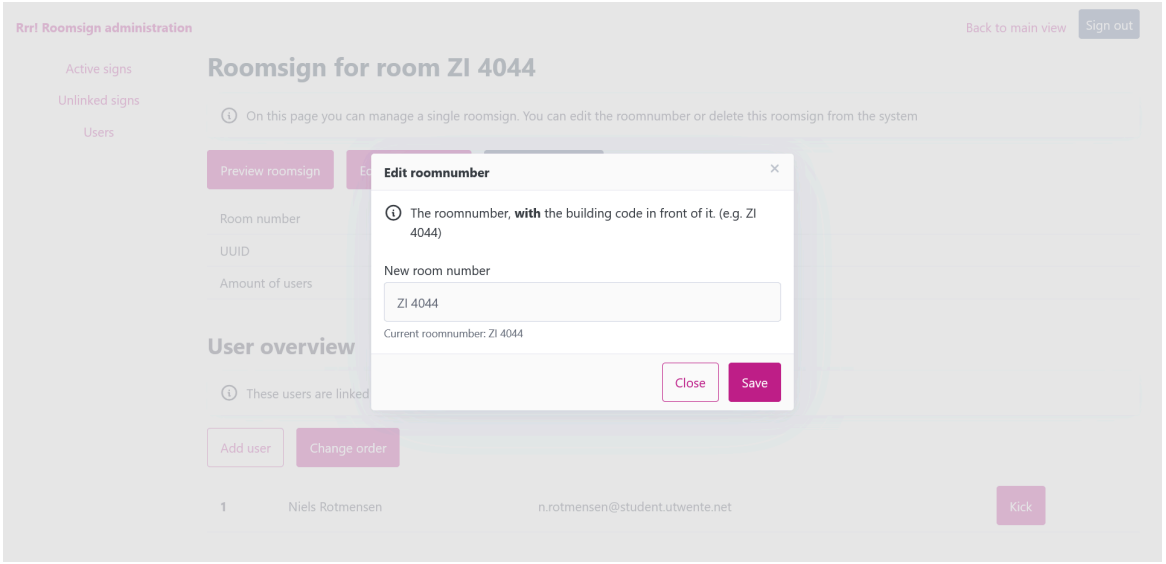


Figure 13: Editing room number

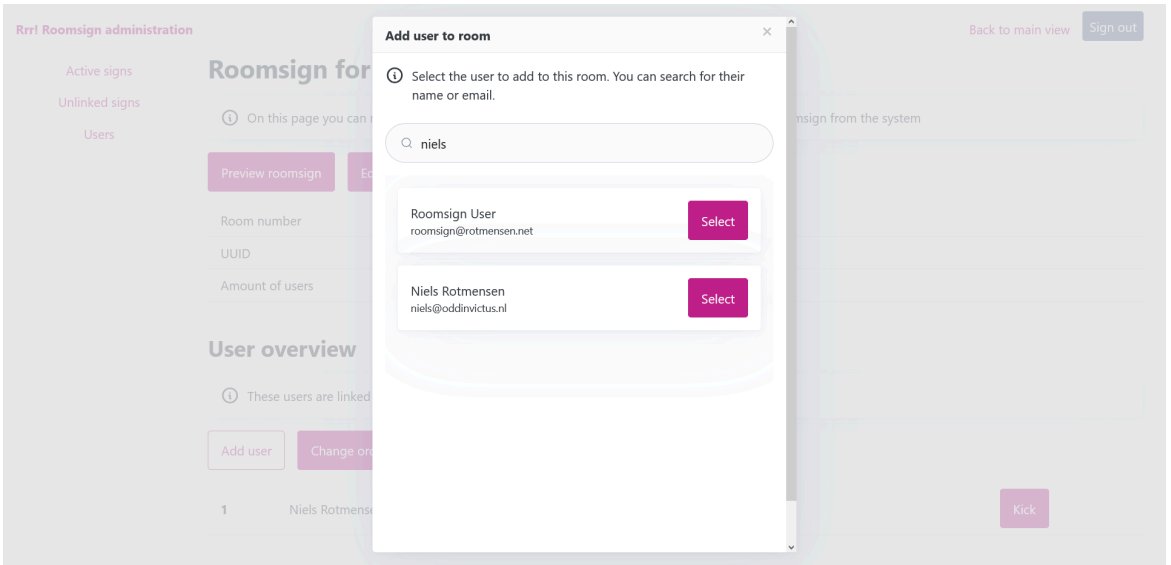


Figure 14: Adding an user by searching

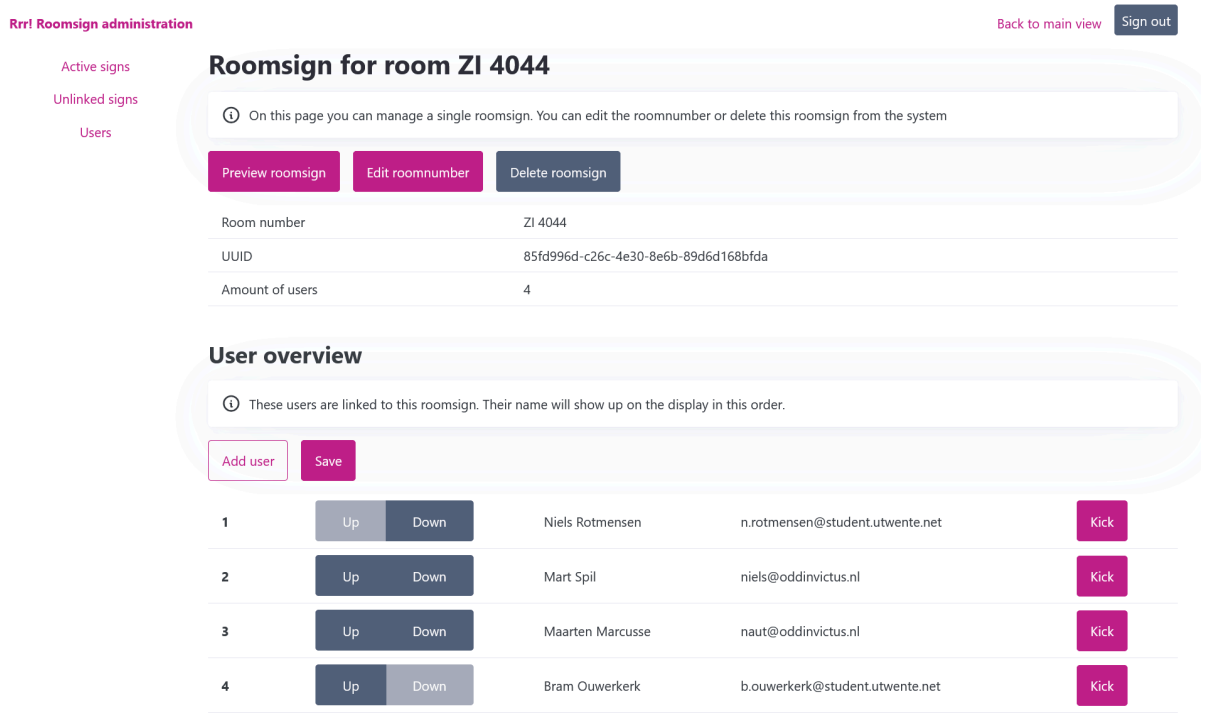


Figure 15: Clicking the change order button will bring up these buttons

When you click the change order button, it will bring up Up/Down buttons to move users. After you are done, you can click save to save your changes.

As shown in figure 14, you can also add users. After clicking on the add user button, it will open a dialog with a search box. In this search box you can search by email, full name or display name. Clicking *Add* will add this user to the roomsign. There is a maximum of 6 people on a roomsign.

Managing users

Navigating to */admin/users* will bring you to the user overview page

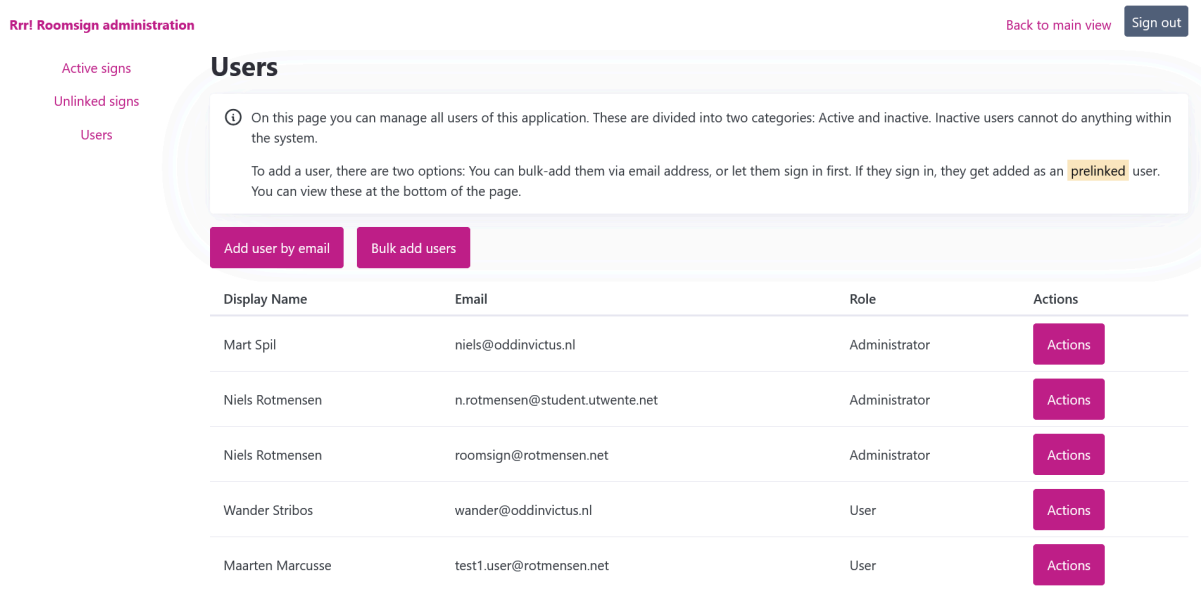


Figure 16: User overview page

On this page you can use the *Add user by email* or *Bulk add users* buttons to link a person to a roomsign, before they have even signed in to the application. This makes it easy to already initialize all data before everyone has signed in. (Note: this does not create an account, it only links an email to a room number, the person still needs to sign in to link their name and such)

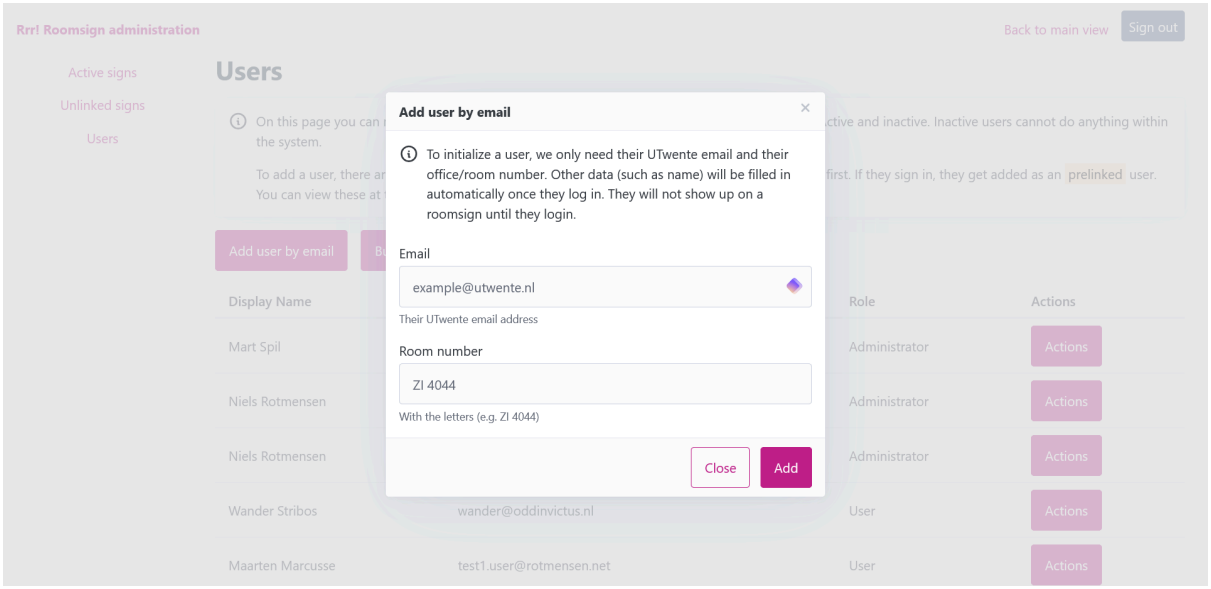


Figure 17: Linking an user to a roomsign before they have signed in

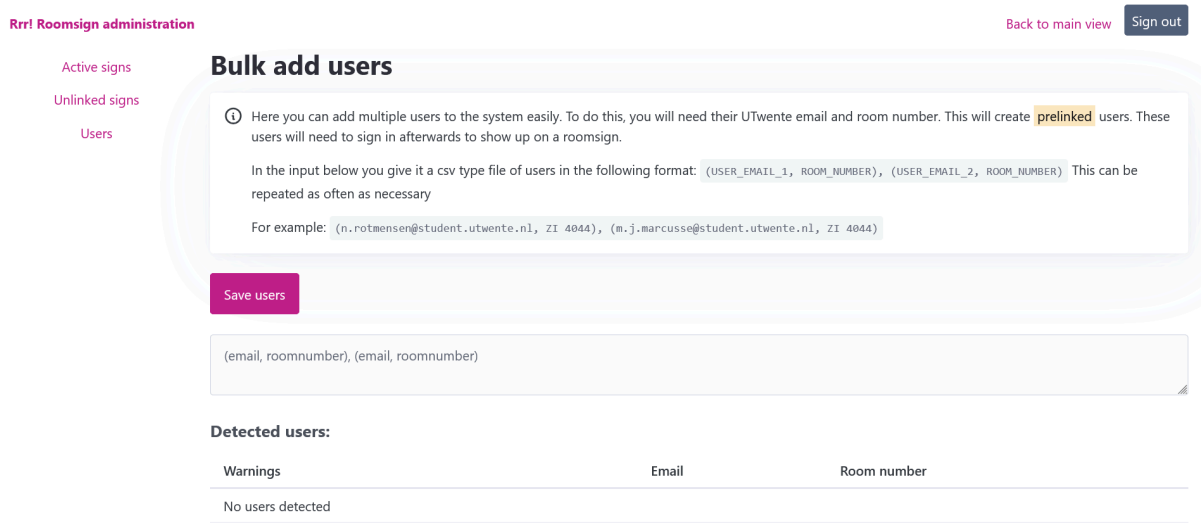


Figure 18: Pressing the bulk add users will bring you to this page.

On the bulk add page, you can use a csv formatted string to link a whole bunch of users to a roomsign. As shown in figure 20, if you input an incorrect value, it will produce a warning.

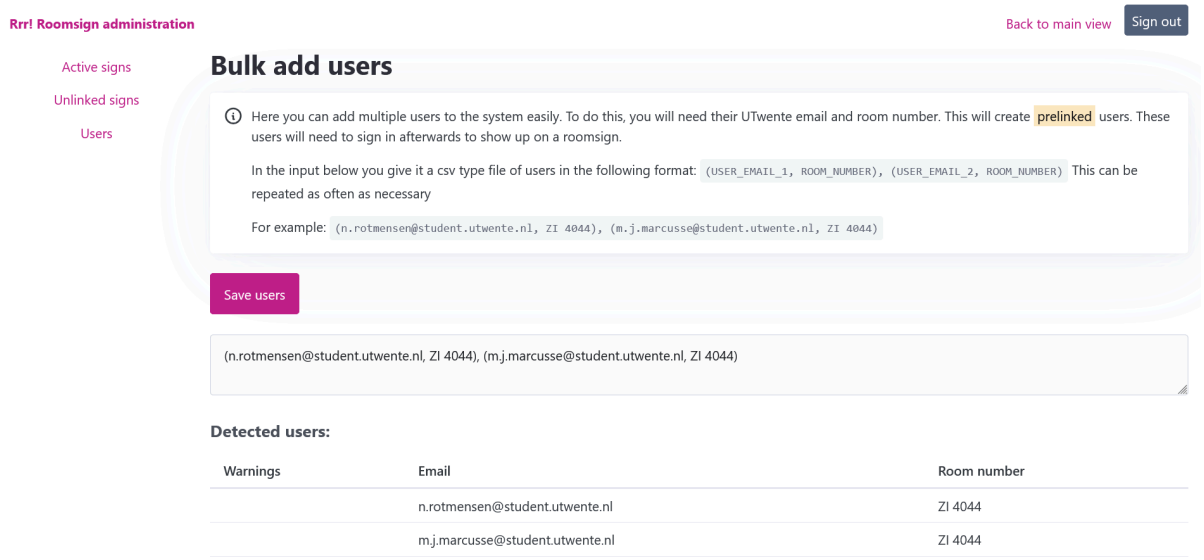


Figure 19: Filled in the form with a correct csv

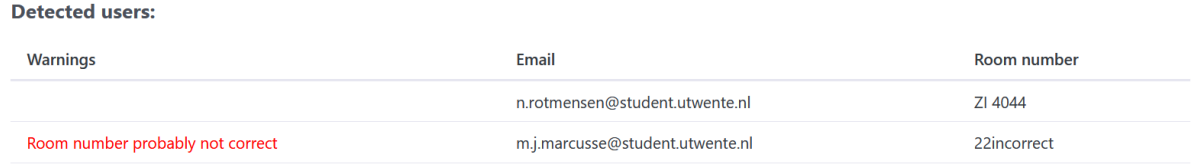


Figure 20: Filled in an incorrect room number, this will trigger a warning

Going back to the user overview page as shown in figure 16, there is another button. The *Actions* button. Clicking this button will open the dialog shown in figure 21.

This dialog will give you a few options to do. Firstly you can toggle someone's administrator status with the gray button. The system will not allow you to remove the last administrator.

If you deactivate the user, it will still be in the system, but will be kicked out of their roomsign and will be unable to do anything within the system.

The change name button will allow you to change their display name.

Lastly, the link roomsign button will bring you to the page shown in figure 22. This page shows all roomsigns and clicking the select button will add the user to that roomsign.

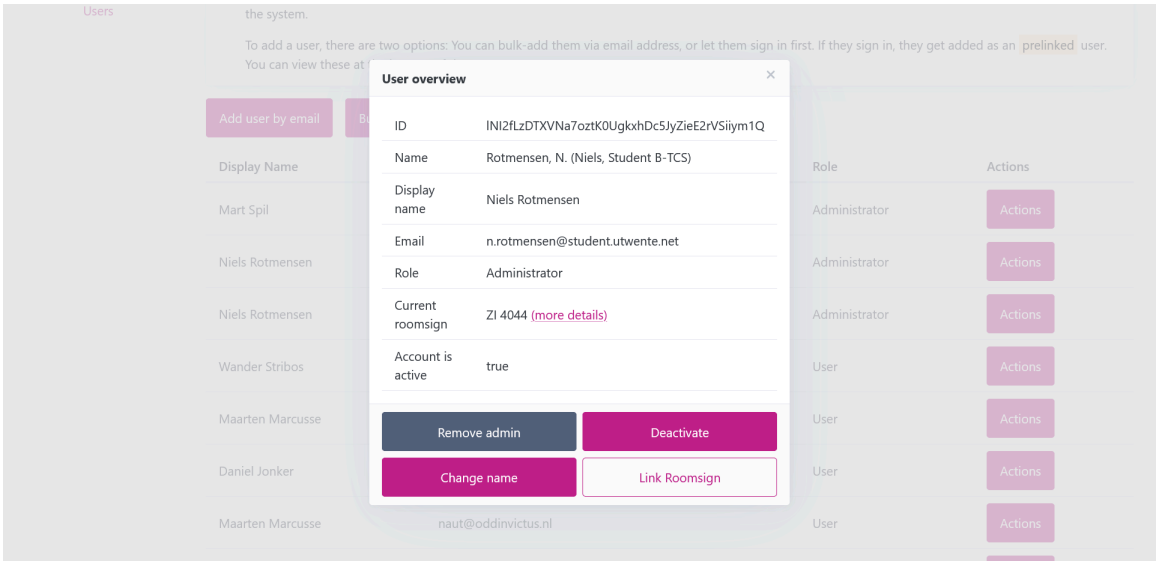


Figure 21: User overview dialog

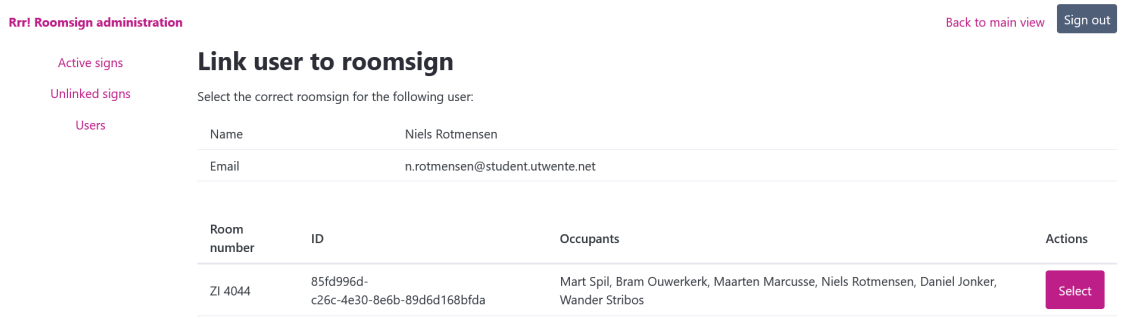
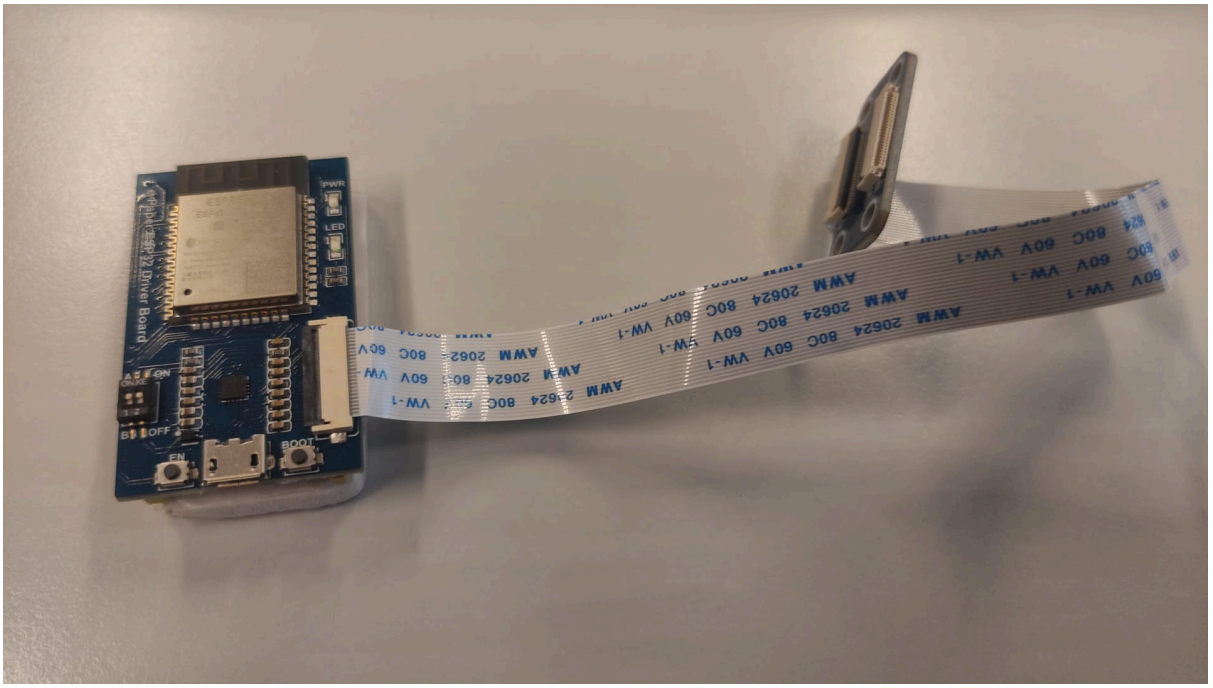


Figure 22: Link user to roomsign page

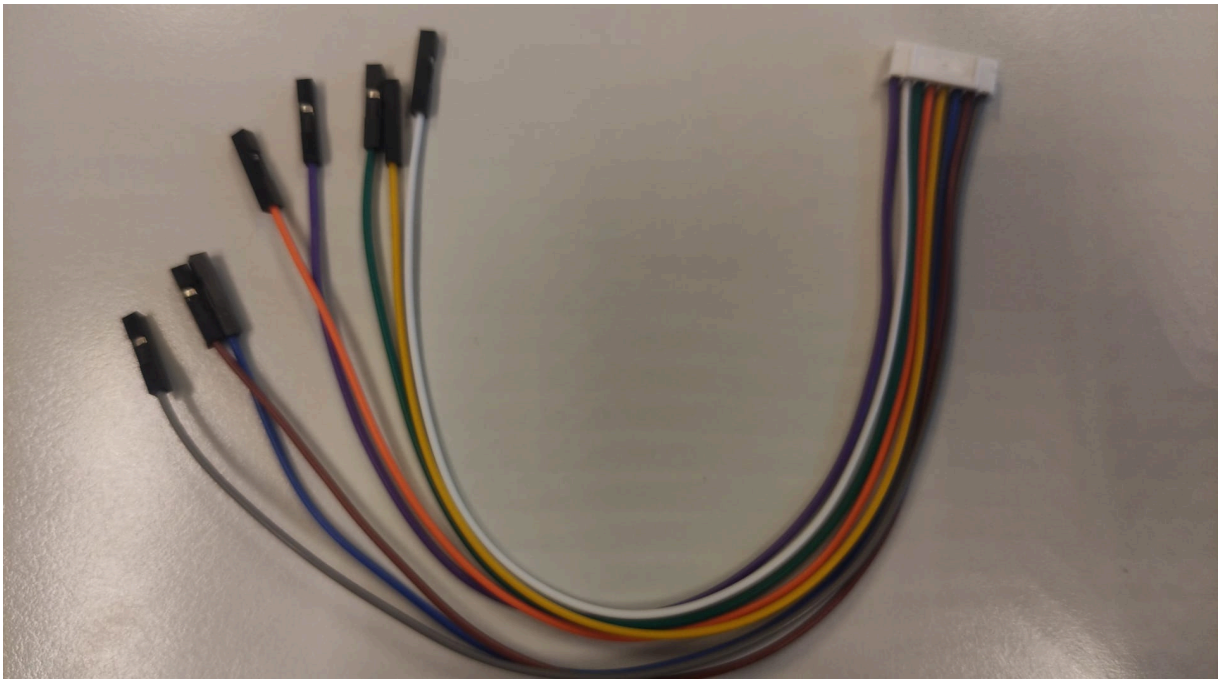
Microcontroller

Setting up the hardware

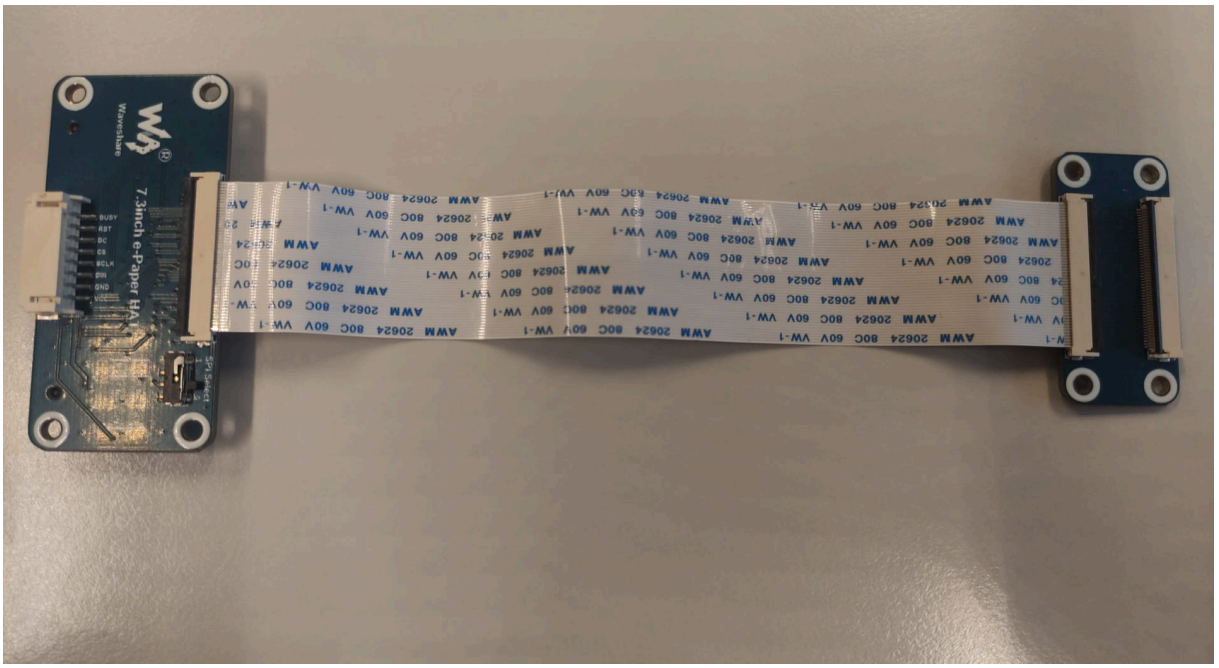
You should have the following items to set up the hardware:
Microcontroller and tiny FCC extension cable:



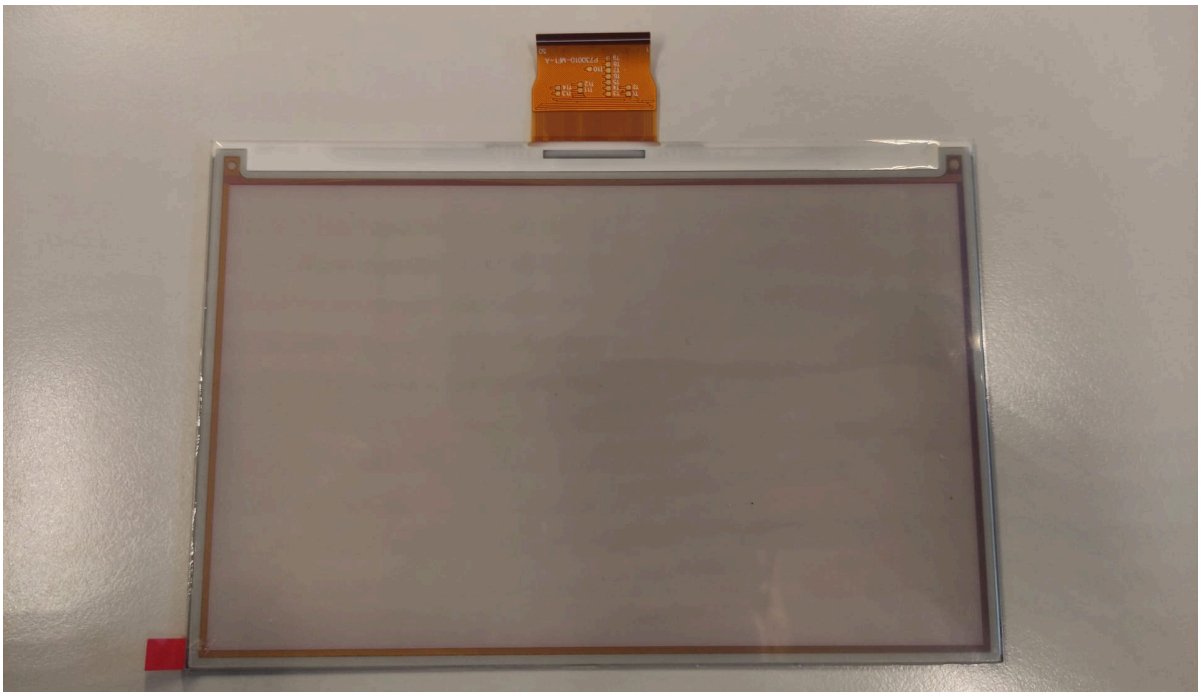
8pin connector:



FCC connector:

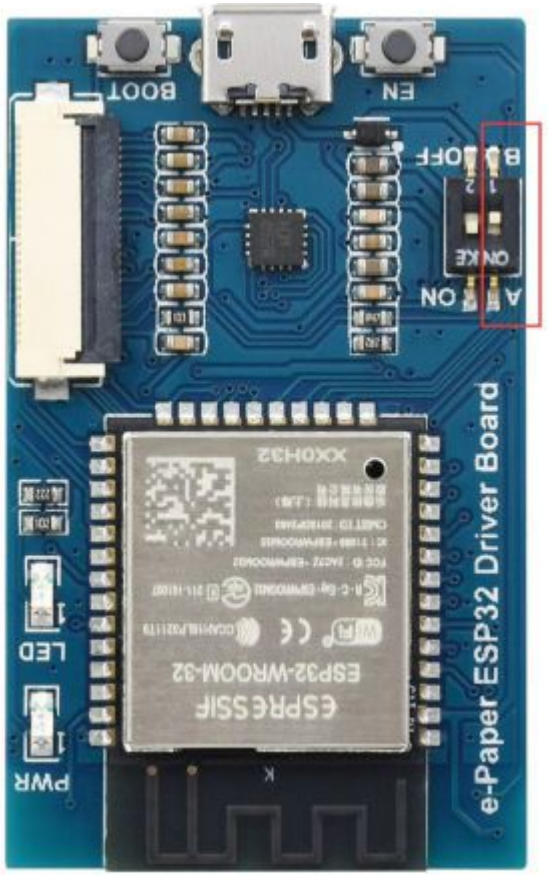


E-paper board:



Step 1:
remove the tiny fcc extension cable from the microcontroller, this can be done by lifting up the black side of the connector.

Step 2:
Set the 2 tiny switches on the microcontroller to On and A

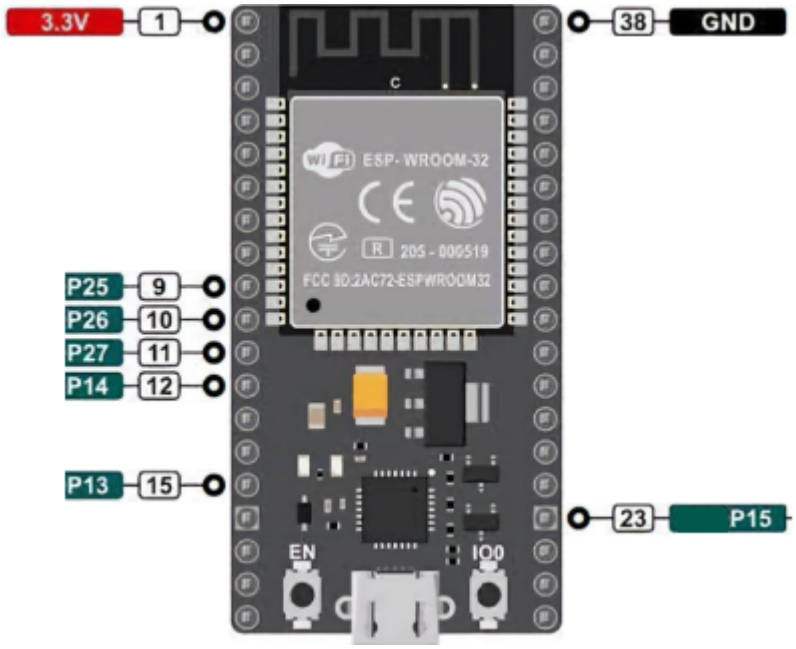


Step 3:
Connect the pins to the 8pin connecter according to the following table

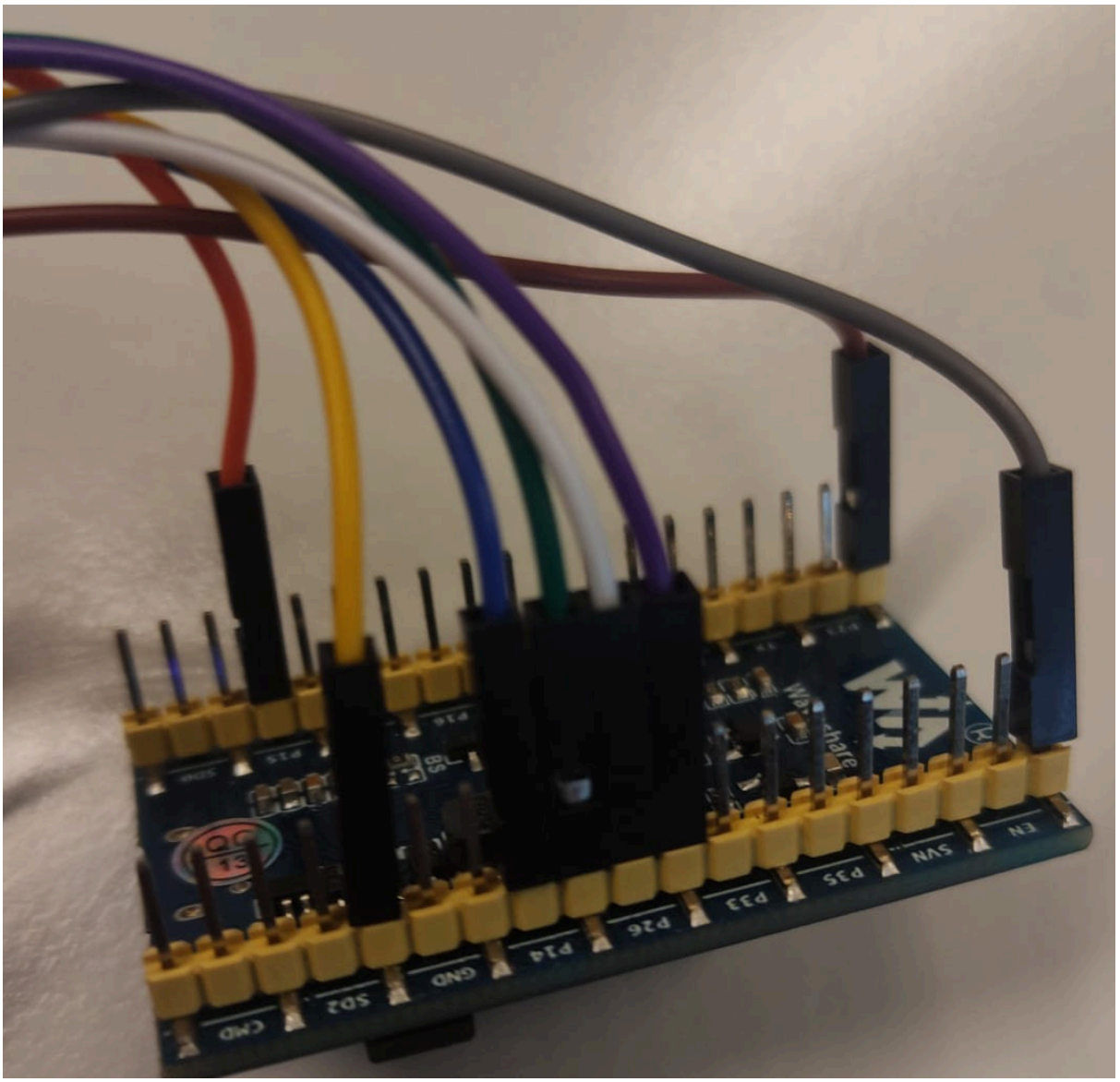
e-paper (these values can be seen on the FCC connector)	ESP32 (the values are next to the pins on the back)	Colour	Description
VCC	3V3	Grey	Power input (3.3V)
GND	GND	Brown	Ground
DIN	P14	Blue	SPI MOSI pin, data input
SCLK	P13	Yellow	SPI CLK pin, clock signal input
CS	P15	Orange	Chip selection, low active
DC	P27	Green	Data/command, low for commands, high for data

RST	P26	White	Reset, low active
BUSY	P25	Purple	Busy status output pin (means busy)

Needed pin layout (from top perspective):

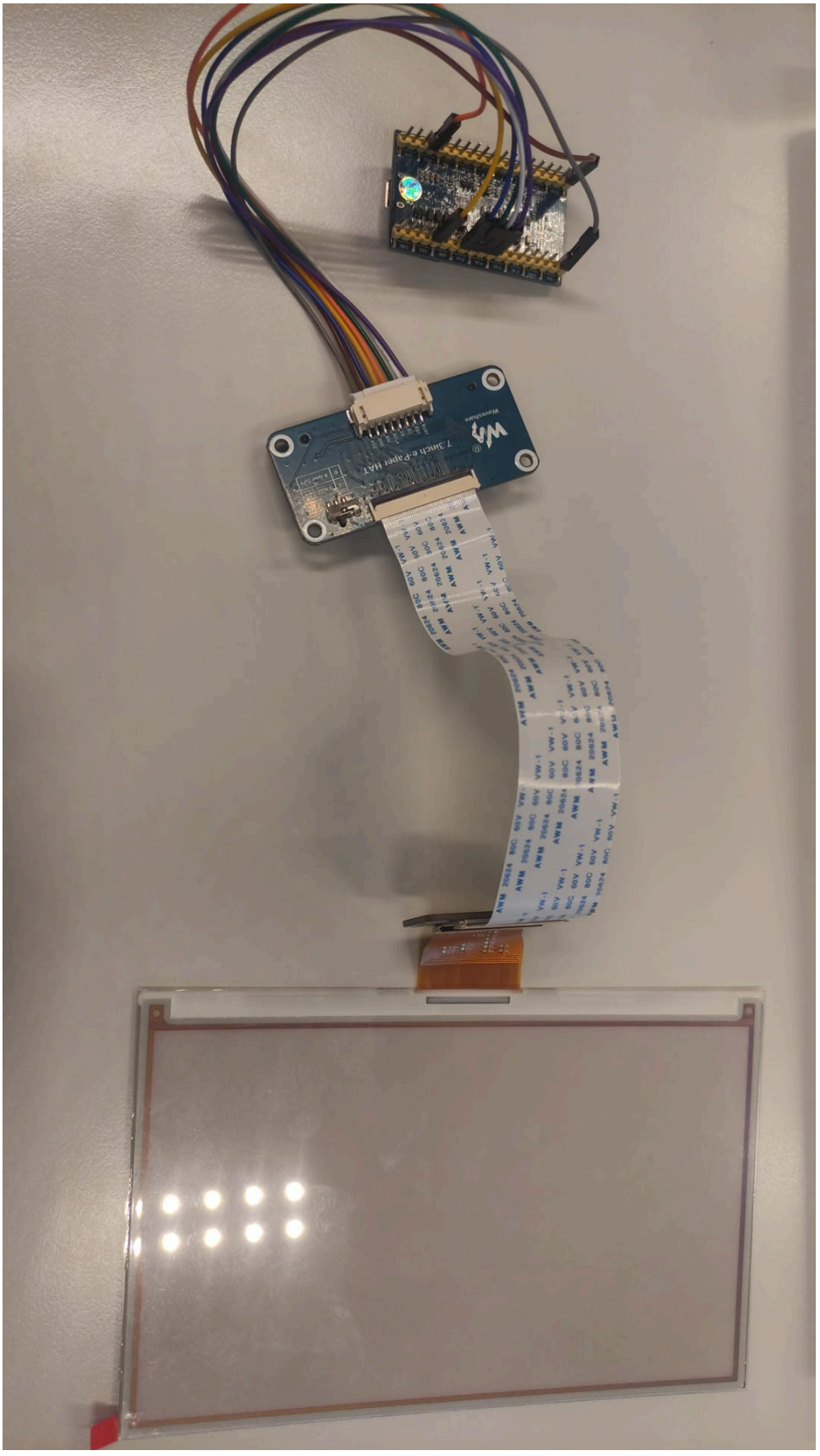


If done correctly it will look like this:



Step 4:
connect the 8Pin connector to the FCC connector
Step 5:
connect the other side of the FCC connector to the board (the connector opens by lifting the black side)

If all steps are done successfully it should result in the following setup:


7.3inch e-Paper HAT
Waveshare

First Time Install

To install Arduino IDE correctly follow the linked guide for esp32
When downloading the esp32 library keep in mind that the product was designed for version 3.1.0-RC2 and might not function properly on other versions.
https://www.waveshare.com/wiki/Arduino_ESP32/8266_Online_Installation

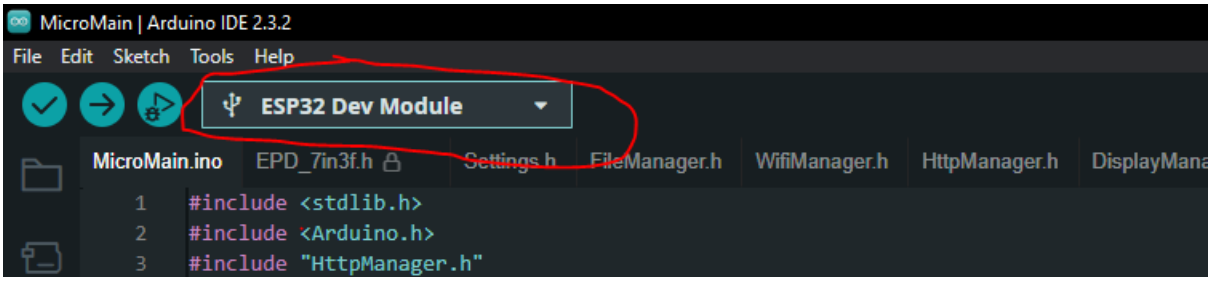
After installing Arduino IDE and the esp32 library you can open /MicroController/MicroMain/MicroMain.ino, after opening the file the project will load in Arduino. Here settings.h will contain a couple of variables that must be set for proper functioning. If the settings ever need to be changed the board will not be reset after flashing the new settings to the system.

Name	What to fill in	Description
ssid	Wifi name	Fill the wifi name in which you want the esp32 to connect to
EAP_PASSWORD	Wifi password	Fill in the password for the wifi you want the esp 32 to connect to
peap	true/false	Fill in false if you want to connect to a wifi that has only a name and a password (like uthings) Or true if a Username is needed (like eduroam)
EAP_USERNAME	Wifi username	Fill in the username for the wifi if this is needed to connect to the network (otherwise can be left the same)
EAP_IDENTITY	Wifi identity	This is in most cases (including eduroam) the same as EAP_USERNAME
refreshdelay	Number	The amount of seconds it takes for the microcontroller to attempt to get a new image, putting it higher decreases energy consumption at the cost of slower updating.

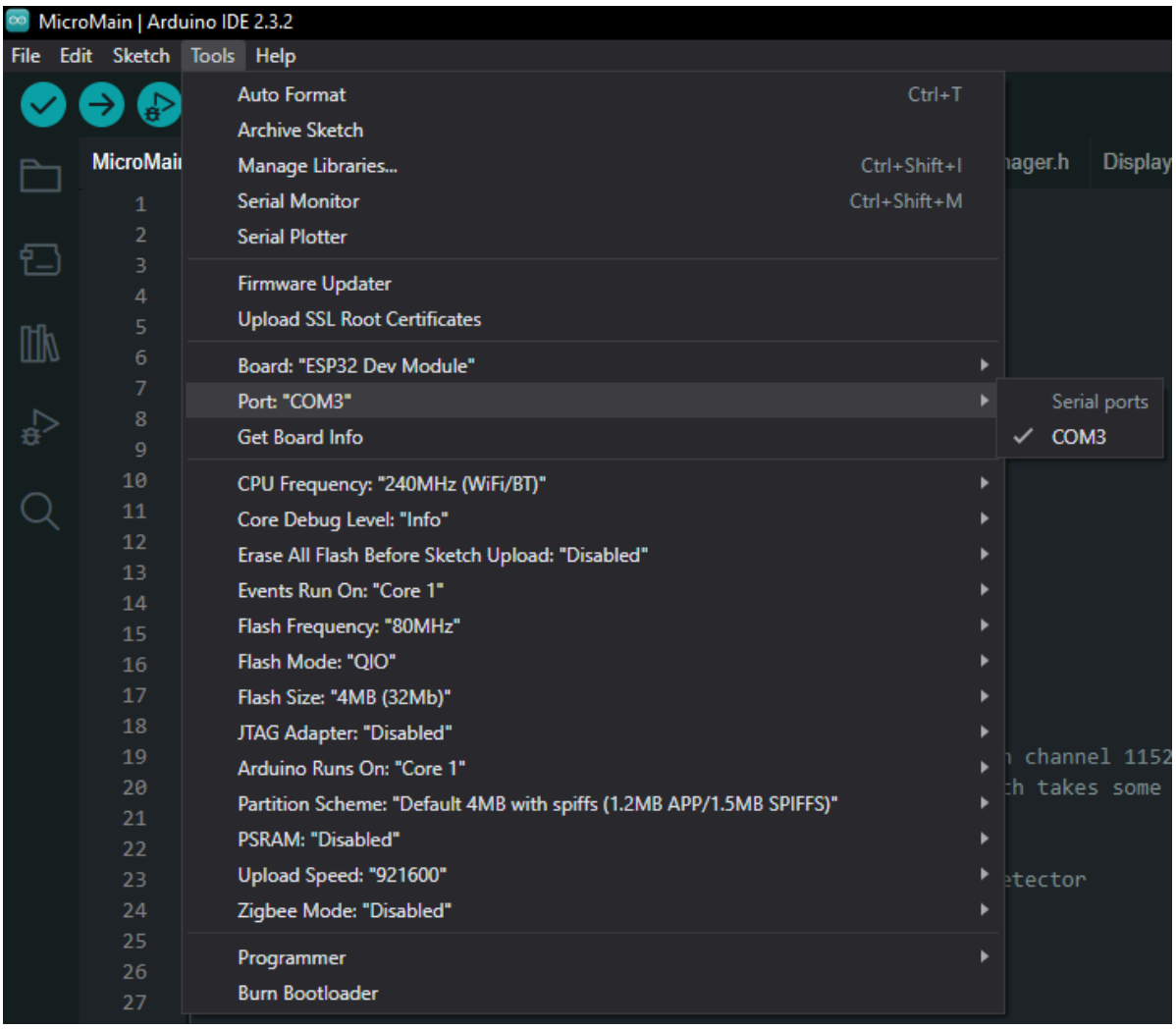
test_root_ca	Certificate	This contains a certificate to enable security, the current one won't need to be changed until 2035

Flashing the Microcontroller

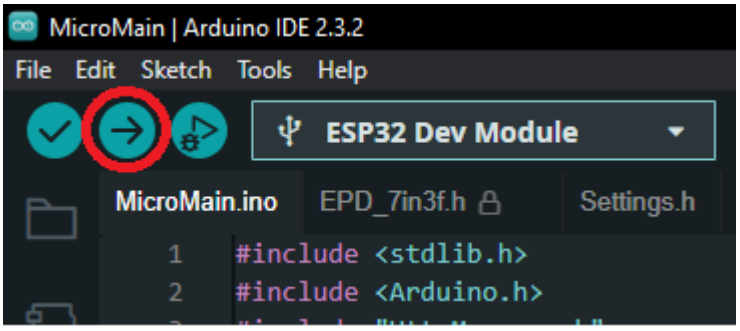
To give the microcontroller the right instructions open /MicroController/MicroMain/MicroMain.ino in Arduino IDE and connect the microcontroller to your computer with a micro usb connector. And ensure the board is set to ESP32 Dev Module.



After doing so the device should show up in the port under tools as COMx



When the board is detected you can upload the file by pressing the upload button (warning this may take a while)



After uploading the file a qr code should show up on the board after around a minute.