Project Report R&D Team-21

Gregor Bujda - s1135939
Björn Smith - s1135336
Robin Strik - s1034818
Chris-Robin Talts - s1150750

1. Introduction

This section is a detailed description of the overview of our system: Its purpose, the intended users, and our motivation to choose and develop this specific project.

Goals of the system

In our current school environment here at Radboud Universiteit, there are no truly effective ways to find your way around campus. Sure, the university does provide a PDF campus map, but that map is non-interactive, and also non-optimized for mobile devices. This means that figuring out where a specific building is, and by extension finding a specific room in a building for a class can be quite challenging for newer students or people visiting campus. To tackle this problem head-on, our group set out to develop a mobile friendly, interactive web application that allows users to:

- Quickly locate any building or any room decided upon important buildings on the Radboud campus via a searchable map interface.
- View detailed room-level and building-level information, including bathroom accessibility and dining facilities for some important buildings.
- Accurate room codes and locations that match pretty well with most students' schedules.

These goals align with our objective of creating an accessible way for new students or visitors to navigate the campus intuitively, reducing the frustration of getting lost or not knowing where to go for a class.

Target users

The primary group of users we decided to target the most consists of students who are new to the Radboud campus, specifically those who are joining the science faculty. For instance, incoming first years often struggle to navigate between classrooms and lecture halls, making the first few weeks of attending class challenging and stressful. Despite our target audience, our tool is to be also useful for:

- Visiting academics or conference attendees who may be unfamiliar with building locations.
- University staff or maintenance workers need quick access to some room-level details.
- Possible incoming students or parents touring the campus for the first time.

Our group believed that by focusing on individuals who stand to benefit most from real-time, location-based search, we could ensure the feature set remains focused, practical, and, importantly, feasible to develop within a short timeframe.

Motivation & Value

At present, Radboud is missing a truly portable way to navigate around the campus. The current options consist of Radboud's official campus map which is distributed as a PDF online (non-responsive, non-searchable), or in building fire escape maps that display room numbers and locations. Though useful, these fire escape maps are sparse and hard to read. Plus, they are stationary, which can also be frustrating. This lack of interactivity can lead to wasted time searching for rooms or buildings, especially when changing classes between relatively far away campus locations. Our web app directly addresses these issues by providing a single, cohesive interface that:

1. Combines building footprints with room-level data (from our JSON data files)
2. Offers an intuitive search bar that accepts building names, abbreviations, or room numbers
3. Displays food and drink, bathroom, and lecture hall locations in a color-coded format through our menu functionality.

By doing these things, we believe that daily campus life for new students and even staff can be improved and streamlined, if even a little bit.

2. Description

This section outlines the core features of our campus-map app, explains why we chose a web-based solution over existing alternatives, and details the technical specifications, such as data formats and rendering behavior, that guided our implementation.

2.1 Focus on properties

This project is a web-based interactive map application specifically designed for Radboud University. It allows users, primarily students, to search for, view, and navigate to campus buildings and individual rooms as the university campus is very extensive, we focused on mapping each building, but only implemented the interior floorplans of the buildings most important for computing science students (Huygens Building, Mercator I, Transitorium, Linnaeus Building and EOS) as those are the buildings our curriculum takes place in. The key properties of the product include:

- An interactive map built using the Leaflet library, offering high zoom resolution and room-level granularity.
- Room visualizations that render dynamically based on zoom level, with room-level data becoming visible at zoom level 16 or higher (Figure 1).
- Clickable building polygons that, upon selection, initiate a smooth zoom and highlight effect while displaying associated metadata in a floating information panel (Figure 2).
- A search function supporting Radboud's internal naming conventions (e.g., HG 00.304), allowing quick look up and location of specific rooms.
- A built-in directory that highlights frequently accessed campus amenities such as lecture halls, restrooms, and dining areas.
- Progressive Web App (PWA) support, enabling mobile installation and offline caching.

The user interface is composed of a full-screen map canvas, a floating search panel, and an information panel that updates contextually. Visual interactions such as zooming and highlighting are animated using Leaflet's flyTo and setStyle methods to enhance usability.

2.2 Product justification

We as first-years students have often found ourselves in a situation where we couldn’t find the room our class was in, sometimes even the overall building, as the naming and code system in our university can be quite overwhelming.

To add to that, the existing method for locating rooms based on their code is:

1. Open the “Building name abbreviations Radboud University” document and find which building does the building code reference
2. Open the “Campus map” document to find where on the campus that building is.
3. After locating the building, go to the “Building name abbreviations Radboud University” document again and understand the room numbering logic (which is not provided for all buildings)
4. Go inside and try to navigate to the room based on the signs.

This approach is inefficient and cumbersome, particularly for new students. The proposed application consolidates all these steps into a single, streamlined interface. By leveraging direct room lookup, interactive maps, and metadata panels, users can navigate with confidence and precision. While the design was inspired by digital maps from other institutions, the final implementation is unique, as each campus is designed differently and we chose the functions that, knowing from experience, a student at Radboud would need the most.

2.3 Specifications

The map uses GeoJSON files to store the shapes and information of all buildings and rooms on campus. We created these files by tracing the evacuation plans found in the university buildings (Figure 3). We did this manually using a program called QGIS, which allowed us to place the floorplans accurately on the map.

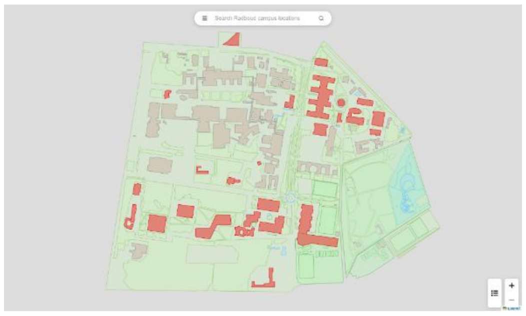
Each building is shown as a shape (polygon) on the map (Figure 4), with extra information like its name, code, and number of floors. Inside each building, the rooms are also shown in smaller shapes. Each room has information such as its room number, building code, floor number, and room name. At this moment, the Huygens Building, Mercator I, the Linnaeus Building, Transitorium, and the Elinor Ostrom building are all fully implemented. The rest of the buildings do not contain any rooms yet.

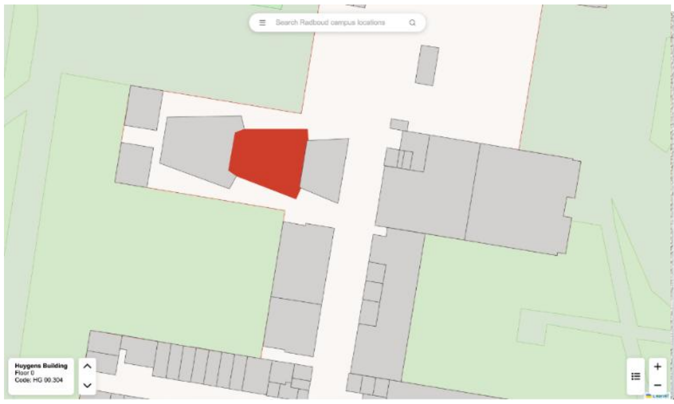
The rooms only appear when you zoom in closely (zoom level 16 or higher), so the map doesn't get too crowded. The map only lets you move around within the campus area to keep the focus on Radboud University.

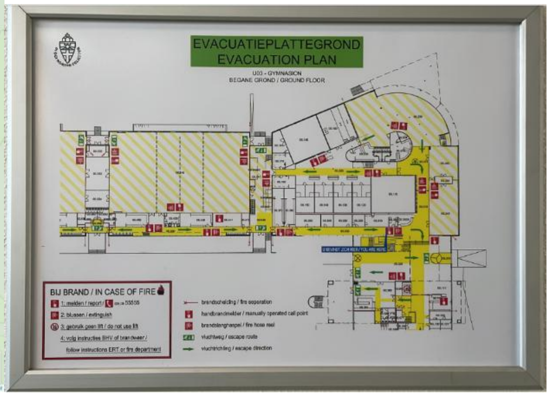
When you click on a building or search for a room, the map zooms smoothly and highlights the location. This is done using special functions from the Leaflet library called flyTo() and setStyle().

Because all the data comes from GeoJSON files, it's easy to update the map if buildings or rooms change in the future. This makes the tool both accurate and easy to keep up to date, helping students find their way around campus more easily.

Our web app functions as a progressive web app (PWA). This means that it can be installed on smart devices directly from the browser to provide a more app-like experience. This includes theme colors and an app icon. Once installed, the PWA updates automatically once an update is pushed to the web app.






EVACUATIEPLATTEGROND
EVACUATION PLAN
U03 - GYMNASION
BEGANE GROND / GROUND FLOOR
U BEVINDT ZICH HIER / YOU ARE HERE
BIJ BRAND / IN CASE OF FIRE
1: melden / report / 024 36 55555
2: blussen / extinguish
3: gebruik geen lift / do not use lift
4: volg instructies BHV of brandweer / follow instructions ERT or fire department
brandscheiding / fire separation
handbrandmelder / manually operated call point
brandslanghaspel / fire hose reel
vluchtweg / escape route
vluchtrichting / escape direction

In order (left-right, top-bottom): figure 1, figure 4, figure 2, figure 3

3. Design

In this section, we describe the system's modular architecture and dive into the key data structures, algorithms, and component interactions, then briefly explain why each design choice was made.

3.1 Global design

Our campus map application is divided into modules, each with a specific task. This modular setup makes the system easy to maintain, extend, and it looks very clean. Below is a summary of each module's purpose:

Main HTML file – index.html

This file contains the layout of the entire web page.

Offline HTML file – offline.html

This file serves as a fallback file for when the service worker fails to start due to connectivity issues.

Stylesheet – style.css

This file contains the style rules used for our web app.

Service Worker – sw.jw

This file contains the service worker responsible for starting and running the service required for the web app to be a PWA.

Web app manifest – manifest.json

Specifies how the PWA should behave once installed.

Building shapes – buildings.geojson

This file contains the locations, properties, and vector shapes of every building on our map in a human-readable format.

Background map – map.geojson

This file contains the vector shapes of the background map in a human-readable format.

Configuration Module – config.js

Stores all fixed constants like map bounds, zoom levels, UI element IDs, and color schemes. Centralizing these values ensures consistent usage and easy updates.

Map Initialization Module – mapInit.js

Initializes the Leaflet map with appropriate boundaries, zoom settings, and a canvas renderer. Other modules use this to access a fully prepared map instance.

Layer Management Module – layers.js

Responsible for loading and displaying GeoJSON data (buildings, rooms, and background features). Controls which elements are shown at different zoom levels and user interactions.

Information Panel Module – infoPanel.js

Controls the information panel that displays building or room details when clicked. Manages visibility and content updates based on user input.

Directory Module – directory.js

Builds and manages the sidebar containing categories like restrooms, lecture halls, and cafés. Handles filtering, highlighting, and user navigation within building floors.

User Interface Module – ui.js

Connects interactive controls such as zoom buttons, floor navigation, search bar, and sidebar toggles and ensures enjoyable user interaction.

App Module – app.js

Wires all JavaScript methods together to create a coherent whole.

These components form a system where each one is responsible for its own part of the functionality. That way we are keeping our code organized and scalable.

3.2 Detailed design

The design of our application has a clearly structured JavaScript modules, data structures, and algorithms to ensure maintainability and extendibility. Below are the main design details:

Data Structures:

- GeoJSON Files: Store building footprints, room shapes, and associated metadata like room numbers, floor levels, and facility types.
- Leaflet Layers: Each GeoJSON feature (building or room) is converted into a Leaflet layer for interaction and visual rendering.
- Caches: Utilize JavaScript Map objects to efficiently store and retrieve building names and room-type classifications, speeding up repeated queries. Without caches the application was super laggy.

Main features/methods in each module:

config.js

- Provides constant values for the entire app.

mapInit.js

- initMap(): Initializes a Leaflet map instance with predefined settings.

layers.js

- createBackgroundPane(): Creates the pane that will hold the background map.
- styleFeature(): Holds the logic for styling the background vectors.
- loadBackgroundGeoJSON(): Places polygons extracted from map.geojson on the background pane, which are then added to the map where they can be displayed using a specified renderer.
- addGroundFloorBuildings(): Goes over buildings.geojson to find ground floor building features and adds these to a specified layer.
- createBuildingFeatureLayer(): Goes over specified feature(s) to turn them into, and return, a polygon layer.
- createRoomLayer(): Goes over buildings.geojson to find rooms belonging to the specified building and floor and returns a feature layer with the matching rooms.

infoPanel.js

- updateInfoPanel(content): Dynamically updates or hides the information panel content based on user interaction.

directory.js

- populateDropdowns(): Classifies rooms by type and populates sidebar dropdowns.
- selectBuilding(): Handles user selections, highlighting relevant rooms and navigating to selected buildings.
- Efficient caching of frequently accessed information.

ui.js

- setupSearch(): Handles the search setup by initializing the required search method when a user inputs text into the search field. It matches lower case-insensitive search terms to the buildings and rooms in buildings.geojson
- Sets event listeners for other UI controls (zoom, floor navigation, the side menu, and the directory).
- Provides responsive interaction handling ensuring intuitive usage.

App.js

- handleBuildingClick(): Handles the updating of the info panel, highlighting of buildings and rooms, and flying to buildings when a feature is clicked on the map.
- updateFloorButtonStates(): Updates the state of the floor navigation buttons by searching in buildings.geojson for the presence of a lower or higher floor of the currently selected building.
- floorChange(): Changes the floor by searching in buildings.geojson for the next or previous floor of the currently selected building.
- classifyRoom(): Returns the class of the specified feature when applicable by matching its code.
- getBuildingName(): Returns the name of a specified building code by searching through buildings.geojson and matching the feature to the code.
- The search functionality is fully set up here by firing clicks on the map when clicking a building or room name in the search results.

By structuring classes, methods, and data handling in a clear way, the design ensures easy future maintenance, enhancements, and scalability.

3.3 Design justification

Our design is implemented with a clear focus on user experience. Each component of the system was built to fulfill a single responsibility, which surely simplifies debugging, future extensions, and has a clean code structure.

Our Design Choices:

- We chose Leaflet as our map rendering library because of its ease of use, documentation, and plugin support. Also, it allows high performance even on mobile devices.
- GeoJSON was selected as the core format for buildings and rooms due to its compatibility with Leaflet and flexibility in editing map data.
- The architecture of components keeps responsibilities separated and makes the codebase considerably easier to work with in teams.
- The rooms in each building only become visible after selecting the building rather than all being preloaded because this improves performance, especially on mobile.

Alternatives Considered:

- We briefly considered MapLibre and OpenLayers as alternatives to Leaflet but found them unnecessarily complex for our use case.
- We also considered implementing a backend using Python but since the app does not require server-side functionality, we chose to keep it fully frontend-based for simplicity.

Resulting Benefits:

- Clean separation of logic across modules.
- Fast performance through selective rendering.
- Easy to update or extend (e.g. adding a new building floor just requires one new GeoJSON feature).
- Accessible UI with intuitive controls and a consistent experience across devices.

In summary, our design balances technical simplicity with rich user interaction, allowing a new student or visitor to easily find their way around the campus without being overwhelmed.

4. Collaboration platform and repository manager

This section describes how our group used GitLab as our main repository management system. Here, we explain our workflow, how we tracked tasks, and any documentation practices that supported our development process.

Repository structure

We maintained a single GitLab repository containing all source code, assets, and documentation for the campus-map web app. Within the repository, we used the following top-level folders:

- `/src/` for JavaScript files
- `/plugins/` for leaflet integration
- `/data/` for our GeoJSON files (which include all our buildings and the map)
- `/css/` for our UI configuration file
- `/assets/` for our assets and icons implementation

We adopted a branched workflow:

1. Main Branch (campus-map-init) – always contains deployable, stable code.
2. Development Branch (dev-setup) – served as the integration branch for completed features before merging into main.
3. When we decided to move towards working with GeoJSON for the map we created the branch: (map-as-geojson) which slowly turned into our main branch.
4. Finally, along the way, we've used another branch (tidy-up) to implement further code in a non-formatting form that needs to be properly formatted before being implemented into our main branch.

Progress tracking and coordination

The main method of progression was tracked through our commitments to each branch. Every group member was encouraged to write a short description after each commit detailing a change they made or a fix they implemented. GitLab was not the main method that we used for communication though. We found it much more convenient to have our main form of communication and collaboration be through a WhatsApp group and weekly meetings in reserved rooms on campus where we coordinated development tasks for each member, talked about changes and additions made, etc. GitLab, though not the main driving collaboration force using in our project, served as our hub for the main programming and development of our project.

5. Evaluation

We consider the outcome of our first Research & Development project a success. Our main goal was achieved: creating an interactive version of the Radboud campus map that makes it easy to find specific buildings and rooms on the campus. Even though we did not manage to implement all the features that we originally planned to do, namely wheelchair accessibility and the calendar feature, we are still satisfied with the outcome.

Our app does contain one main aspect that can still be optimized. The biggest challenge we faced was settling on a rendering method for the background map. Our first implementation used raster map tiles from Stadia Maps. While this provided a generally smooth experience for all browsers, both on mobile devices and on desktops, it sometimes caused iOS WebKit-based browsers to crash when zooming in and out due to an unresolved WebKit bug (https://bugs.webkit.org/show_bug.cgi?id=172206). We then gave vector map tiles a try, but this resulted in visible lines between the tiles in Firefox. In the end, we ended up settling for a vector layer directly generated from a GeoJSON file. This implementation works well on most browsers and devices but has some performance issues on lower-end PCs and the Android Brave browser. Other than this, our web app does not contain any unresolved issues that we know of.

The development process of our web app went well. By using GitLab, every one of us was able to work on our assigned tasks at any time. Because none of us had experience with R&D, we were not able to fully follow our weekly planning, but we managed to finish on time without any major problems.

When we started with the development of our web app, none of us had any experience with web development yet. This meant that we had to study HTML, CSS, and JavaScript before we could actually start programming. Since we were working with geospatial data, we also had to learn how to work with QGIS, a geographic information system software. Together with the background rendering issue, these were the main factors that influenced our development process, as they took quite some time.

As a result, we are now able to make better decisions in the future directly from the start when it comes to web development projects. Additionally, working with geospatial data has taught us a lot about how map rendering works in a web setting. In future projects, we intend to operate in a similar manner to what we did this time, namely via good communication and a clear division of tasks.