April 2024

SiteVisor

Digital Twin Application for Buildings Monitoring and Asset Management

FINAL REPORT

NAME: Grzegorz Piotrowski

STUDENT ID: 20099926

SUPERVISOR: Caroline Cahill

MODULE: Project

COURSE: Higher Diploma in Computer Science

Acknowledgements

I would like to thank my wife Julita, for her support and encouragement throughout my work on this project and the whole journey through the course.

Sincere thanks to my supervisor, Caroline Cahill, for insightful suggestions, and valuable meetings which kept me focused and on track to reach this milestone.

Lastly, great appreciation to all the lecturers and staff involved in the Higher Diploma in Computer Science course. This was a truly amazing experience and the knowledge and skills I was able to gain are a testament to your commitment.

Preface

The report should be read together with the following:

Project Board: https://github.com/users/grzpiotrowski/projects/2/views/2

Project Roadmap: https://github.com/users/grzpiotrowski/projects/2/views/3

Repositories:

Main repository: https://github.com/grzpiotrowski/sitevisor-project

Frontend: https://github.com/grzpiotrowski/sitevisor

Backend: https://github.com/grzpiotrowski/sitevisor-backend

Sensor code: https://github.com/grzpiotrowski/sitevisor-sensors

Application Documentation: https://grzpiotrowski.github.io/sitevisor/

Declaration

I declare that the work which follows is my own, and that any quotations from any sources (e.g. books, journals, the internet) are clearly identified as such by the use of 'single quotation marks', for shorter excerpt and identified italics for longer quotations. All quotations and paraphrases are accompanied by (author, date) in the text and a fuller citation is the bibliography. I have not submitted the work represented in this report in any other course of study leading to an academic award.

Student ……Gregor P. Doody…….. Date 04/04/2024

Abstract

SiteVisor is a Digital Twin-based web application for environmental monitoring of buildings with elements of asset management. At the core of the user interface and experience is an interactive 3D viewer, rendering a representation of a building and the IoT sensors placed inside it.

The design philosophy is centred around simplicity, with that even users with minimal technical background can create and interact with the digital twin. With only a few clicks we can set up our 3D environment from scratch, by simply sketching out rooms and creating virtual representations of sensors and configuring the data connection from their physical counterparts. This spatial context can enhance the understanding of IoT data and subsequently we can bring out more information from this data.

The users can manage multiple projects, each representing different buildings or sites. Sensor data can be visualised on graphs or as a live heatmap directly in the model.

The application is deployed on Kubernetes and uses Strimzi operator to run Apache Kafka cluster, which is the heart and arteries of the whole system. This approach enables scalability and robustness.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1. Background

Considering my previous background in geomatics, I have been interested in ways of capturing and representing the physical environment around us for quite a long time now. Most commonly this is done through 2D maps and plans, or increasingly often in case of buildings and infrastructure, through complex 3D models. All these representations however have one thing in common – they capture the state of the modelled object at a specific point in time. Therefore, they are technically outdated almost immediately after being created.

In case of relatively static objects, like the structure of a small building, we can assume that the state has not changed significantly and ignore this issue. That said, almost any functioning building today is more than just a shell of the roof and walls and contains plenty of dynamic systems which all work to provide a comfortable environment for the people inside the building.

To gain some insight into the state of these systems and environment, we could perform periodical inspections and measurements. This seems like a costly and time-consuming process, giving us yet another snapshot approximating the real state of the system, which also quickly becomes outdated. For this reason, installing various sensors across the building comes to mind as a valid solution to this information deficit. The sensors can, for example, measure the basic environmental parameters like temperature, humidity, light levels, CO_2 concentration and many other factors including vibrations and motion of the structure.

Assuming we have now installed all the sensors across the building, we are getting a rich stream of data, that in the simplest solution would get written somewhere and provide no real meaning to the interested parties. That is, until it gets organised, processed, and put into context, which brings us from *data* to *information* (Galbraith, 2023).

Going further with this thought process, we should now connect the data stream to some form of a hub where it would be annotated, organised, and then presented to the user, perhaps in a table or graphed as a time series. This might work great for a few sensors, but we could easily get lost in a long list of devices when dealing with more complex buildings and systems.

This finally brings us to the vision for this project, which aims to utilise the benefits of 3D models and visualisation and bring all the information together in a form of digital representation of a building, its components, relevant content, and with sensor data incorporated into it. This is not a new concept conceived here by any means, and it is already existing and called a *digital twin*.

1.2. Problem statement

To recap the issues described in the previous section shortly:

- Limited insight into the status of building systems and environmental conditions.
- Dependency on outdated and static representations of buildings and infrastructure.
- Inefficient and costly periodic inspections for assessing system performance and condition.

1.3. Proposed solution

An application that provides:

- Continuous feed of environmental sensor data.
- Integration of sensor data into a digital twin representation of the building.
- Enhanced user experience through intuitive interfaces and interactive 3D visualisation tools.
- Sensor and asset management through issues/maintenance tasks.

2. Research and Analysis

2.1. Digital Twin

As hinted in the introduction, a digital twin is a virtual replica of a physical object, system, or process, which accurately represents its real-world counterpart. This model is updated dynamically by integrating data from various sources, such as physical sensors and predictions in a digital environment. This approach provides the best current approximation of the state, performance, and operation of the physical object. Digital twin technology enables users to gain better understanding of the real-world subject through analysis, monitoring, and simulation (Rajamurugu & Karthik, 2023).

A conceptual architecture of a digital twin in the context of manufacturing is shown below, created by (Parrott & Warshaw, 2017). They presented it as a sequence of steps:

1. **Create** – referring to the sensors around the physical asset and the process.
2. **Communicate** – real-time bidirectional communication, highlighting edge processing, interfaces, and edge security.
3. **Aggregate** – data ingestion into a database and preparation for analytics.
4. **Analyse** – utilising advanced analytics technologies to aid decision-making.
5. **Insight** – presenting the results through dashboards with visualisations.
6. **Act** – insights feed back into the physical asset, completing the loop.

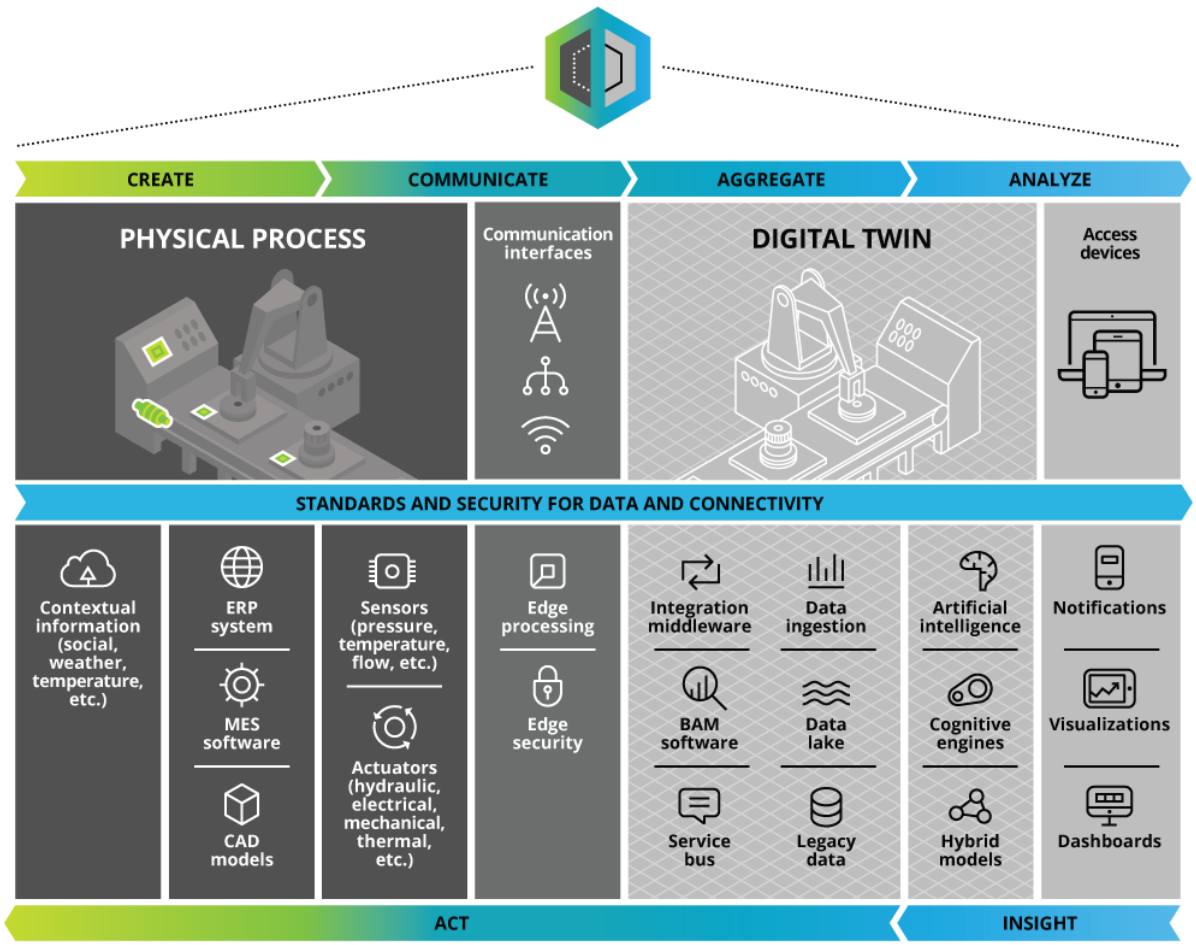

Figure 1. Digital twin conceptual architecture as per (Parrott & Warshaw, 2017).

In a sense, digital twin builds on top of the internet of things and provides a way of contextualising the IoT data. The relation between IoT and Digital Twin in an example system architecture is shown below:

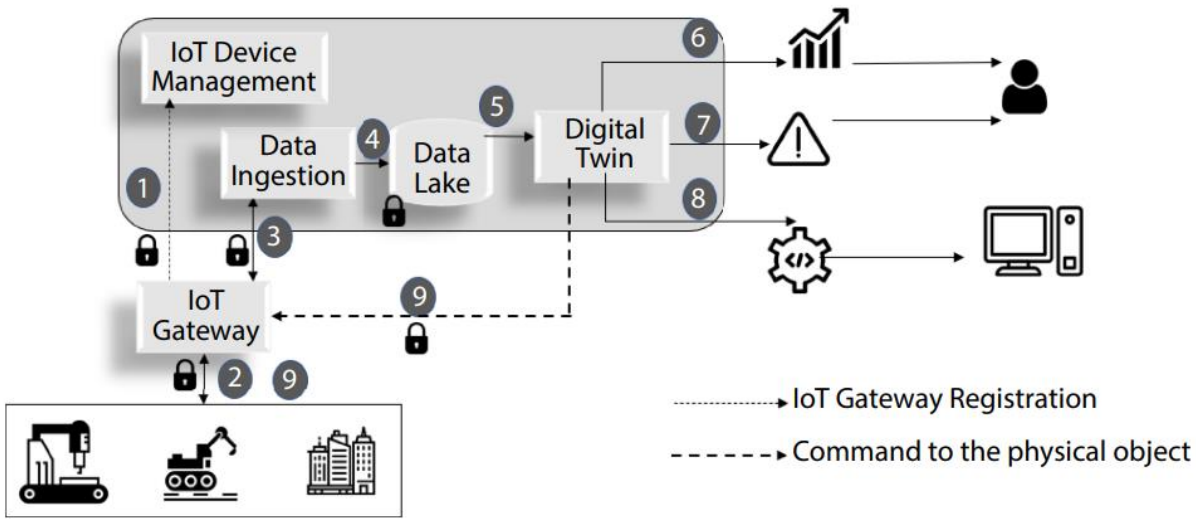

Figure 2. Internet of Things and digital twin end-to-end solution architecture as per (Joshi, 2023).

2.2. Existing solutions

One of the implementations showcasing a 3D model enhanced with IoT sensor data is the Newcastle Urban Observatory portal (UKCRIC, 2023) and specifically the representation of the Urban Sciences Building at Newcastle University. The building has more than 4000 sensors, monitoring various aspects of the environment like temperature, humidity, CO_2 concentration, air pollution and light brightness level (CIBSE, 2018).

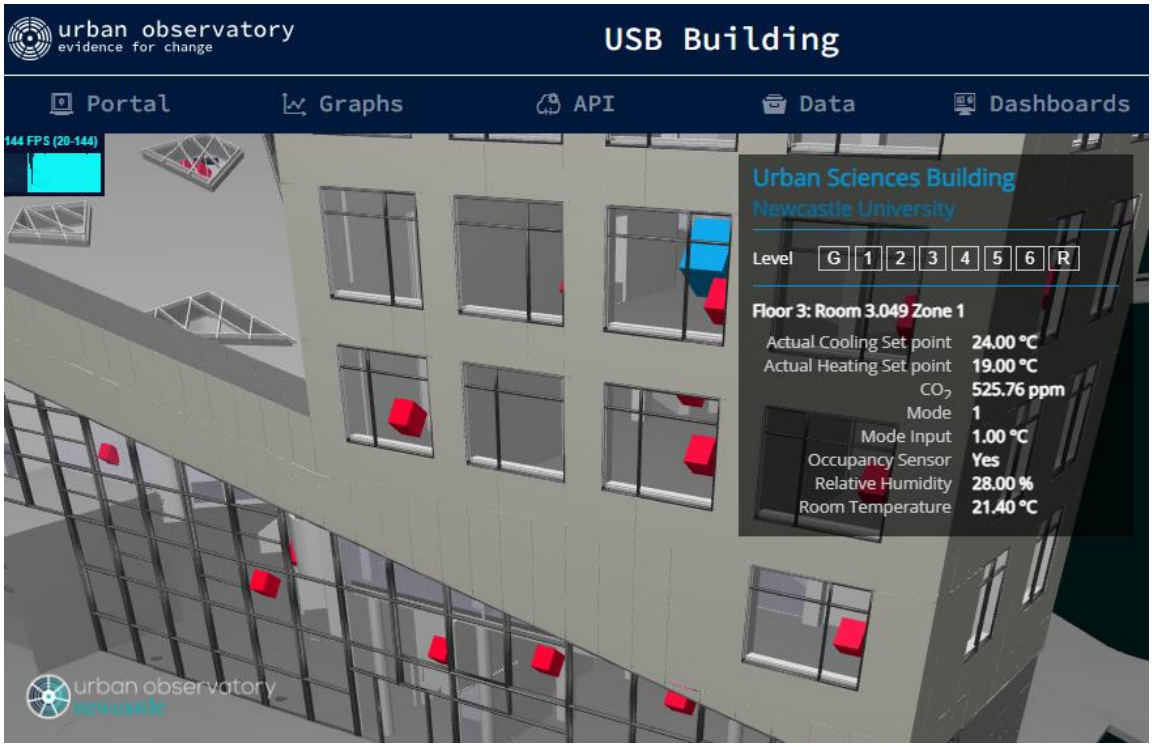

Figure 3. BIM model of Newcastle University's Urban Sciences Building accessible on Urban Observatory website.

As shown in the image above, the user can view the real-time data from various sensors around the building. The red cubes are representing rooms and zones in the building and after clicking on them, an overlay window shows readings from sensors associated with that zone. This is very intuitive in terms of navigating around the rendered environment. However, besides the monitoring capabilities the user cannot affect the system in any way. This is an area of potential improvement in terms of interactive features that SiteVisor can provide.

The next example is Forge Digital Twin Demo, created by (Autodesk, 2022) and showcasing the use of their Forge platform in the context of digital twins. It enables to view a detailed 3D model of an asset and monitor its real-time performance. There is also a feature allowing to view and create issues related to specific parts of the asset as shown below.

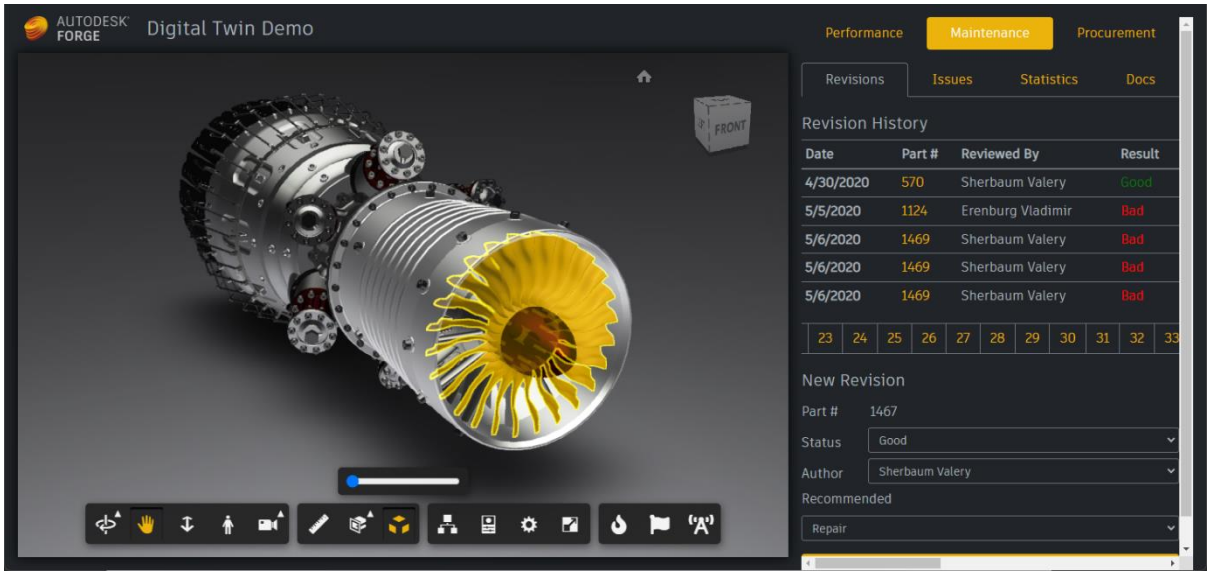

Figure 4. Autodesk Forge Digital Twin Demo.

2.3. Digital Twin Platforms

Cloud services providers are offering their platforms for digital twins as a service (PaaS) enabling their customers to create digital models of entire environments. Two of the platforms that seem to stand out the most are Amazon's AWS IoT TwinMaker (Amazon, 2024) and Microsoft's Azure Digital Twins (Microsoft, 2024). It is important to note that project SiteVisor did not aim to recreate a general platform for creating digital twins, but rather to provide an easy-to-use application and code-free means for facility managers to create digital twins of buildings and manage the rooms and sensors inside them.

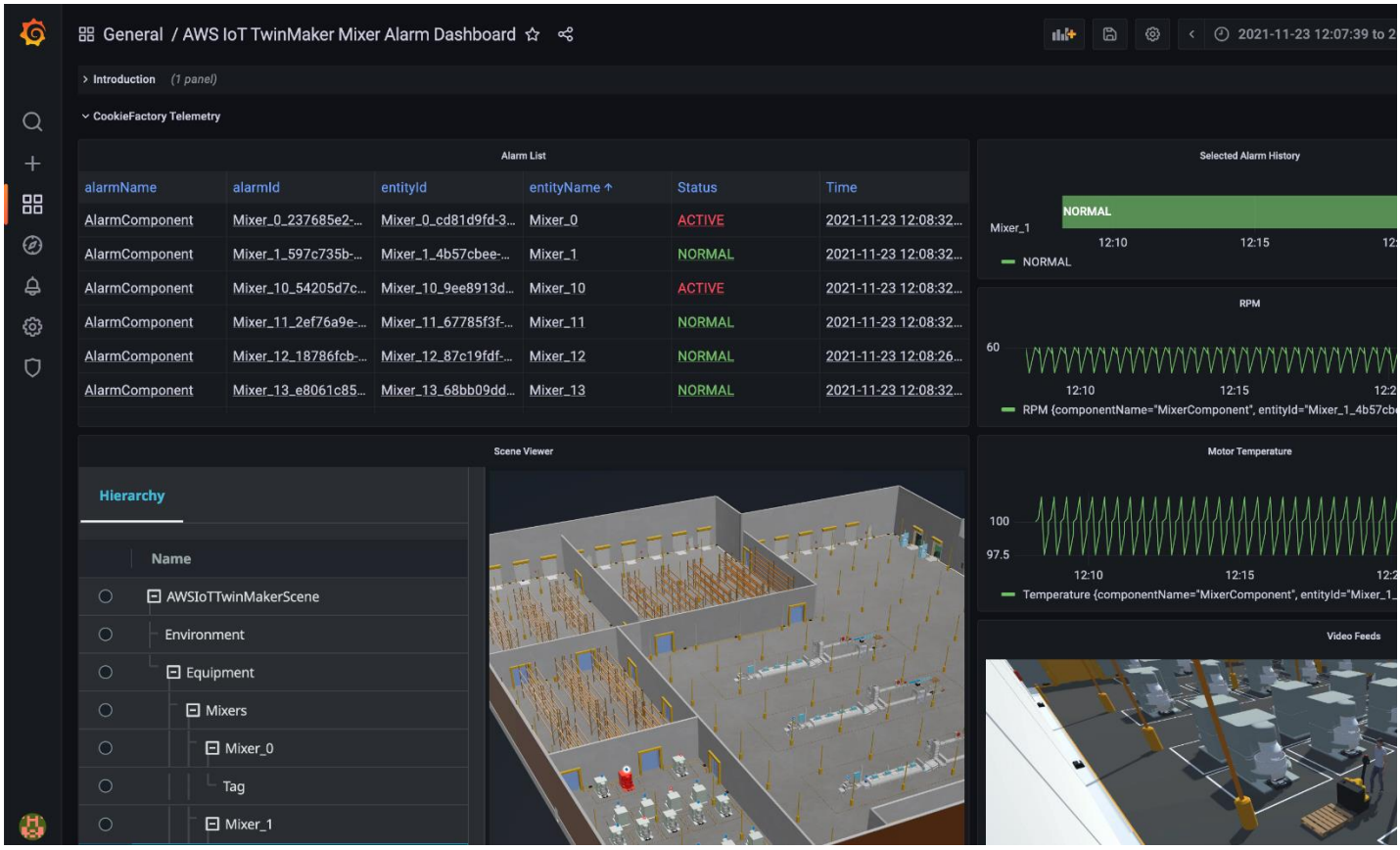

Figure 5. Example Cookie factory Digital Twin dashboard built with AWS IoT TwinMaker and Amazon Managed Grafana (Amazon, 2024).

Both platforms offer an interactive visual 3D environment (Figure 5, Figure 6), where users can monitor and investigate operational digital twin data. The offerings have an advantage of close integration with other cloud services from the providers and are designed to handle large number of models and devices with scalability as needed.

Figure 6. Azure Digital Twins 3D Scenes Studio (Microsoft, 2022).

Together with great capabilities of these systems, comes complexity in setup and management, requiring a steep learning curve. One of the disadvantages of these platforms could be the potential need for integration with non-Azure or non-AWS infrastructure, as the users might already have their IoT systems deployed with different cloud providers.

2.4. Digital Twin Architecture

Research literature is quite plentiful on the topic of digital twins. There are also many books which go into the subject in depth. The goal of researching the publications was to gain insight into the best practices, any established standards and fitting system architectures.

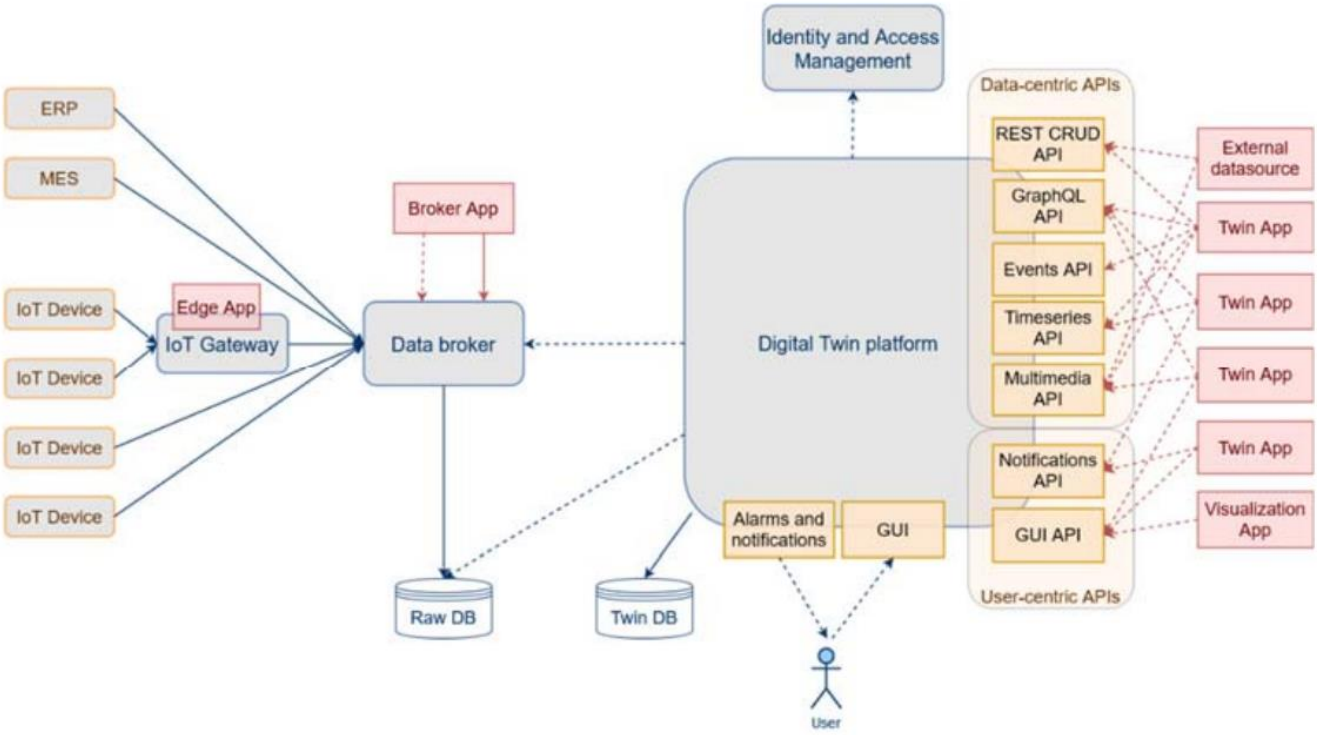

Figure 7. Overview of the IoT system architecture with a focus on the Digital Twin platform (Nölle, et al., 2022).

The diagram above shows a typical system architecture for a Digital Twin platform (Nölle, et al., 2022). As intuited in the initial phase of the project ideation, the usual architecture utilises a central data broker, which gathers data from the IoT devices. From there, the data can be streamed to the digital twin applications but also stored in a dedicated database. In the diagram, the dashed arrows originate from the platform and point towards the broker and the raw database, to indicate the data flow that is on-demand. In contrast, the stream of sensor data from the devices to the broker is continuous.

2.5. Technologies

This section provides an overview of the main technologies researched and selected for the SiteVisor project.

2.5.1. Kubernetes

Kubernetes is an open-source platform for running and scheduling containerised workloads on multiple hosts (Rosso, et al., 2021). In short, it can be described as a container orchestrator. It was picked as a platform for this project due to its flexibility, networking capabilities, automated scaling and load balancing which can facilitate a growing digital twin system.

2.5.2. Apache Kafka

Apache Kafka is a distributed streaming platform that functions as a publish-subscribe messaging system designed to handle high volumes of data in real-time. It provides storage and allows for continuous processing of data streams. Kafka is commonly used in real-time streaming data pipelines and applications (Scott, et al., 2022). Kafka might be too powerful for this project and heavily underutilised, but this choice was made mostly based on my desire to learn this technology.

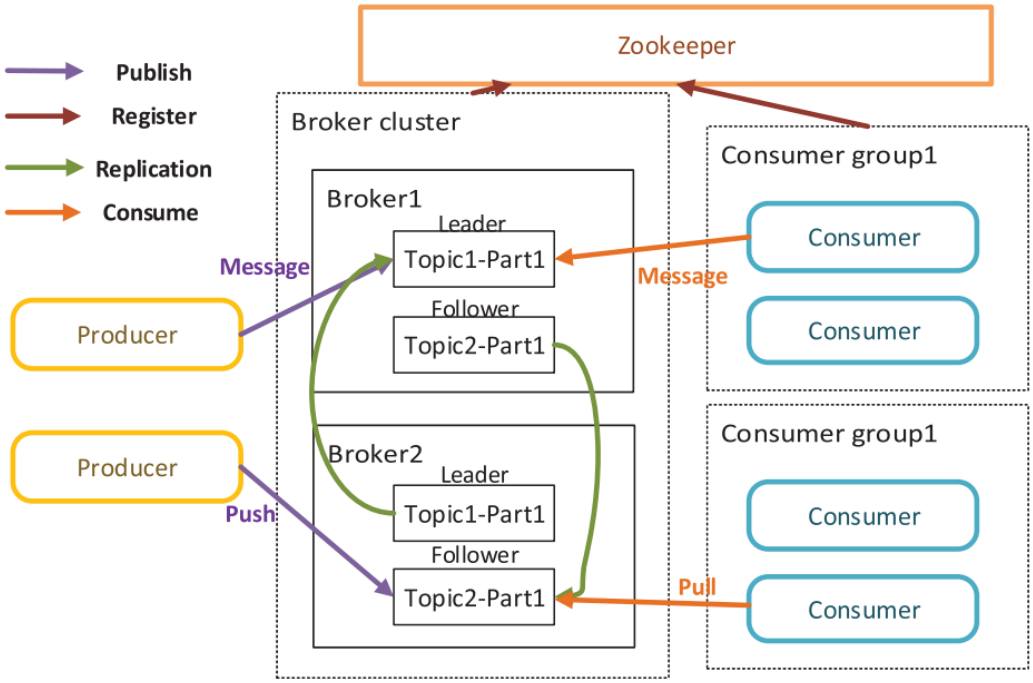

Figure 8. Kafka architecture (Fu, et al., 2021).

Key terminology describing Kafka components:

- **Broker** – server, receives messages from Producers and allow Consumers to fetch them.
- **Cluster** – multiple Kafka Brokers as a distributed set to increase availability.
- **Topic** – a category for streams of data. Every message received by a broker is written to a topic log.
- **Partition** – a subset of a given topic, used to spread the load and facilitate data sharding.
- **Producer** – external application which sends messages to a topic.
- **Consumer** – external application that receives messages from a topic.
- **Zookeeper** – coordinates all the brokers in a cluster.

2.5.3. Strimzi

Strimzi is an open-source project which simplifies the process of running Apache Kafka on Kubernetes. Operators provided by Strimzi make the administration tasks easier and reduce the manual intervention required to manage Kafka cluster. For this reason, Apache Kafka deployed by Strimzi Operator was picked as the backbone of the project architecture.

2.5.4. Three.js

Three.js is a powerful 3D JavaScript library which enables to create and animate 3D scenes rendered in real time in the browser. It uses WebGL which is supported by modern web browsers but makes the development of 3D scenes a lot easier (Dirksen, 2023). Some of the features possible with Three.js are:

- Creating 3D geometries and rendering them in the browser.
- Animating/moving objects in a 3D scene.
- Texturing and applying materials to objects.
- Adding post-processing effects to the rendered scene.
- Loading assets from 3D modelling software and exporting models into files.
- Creating virtual reality (VR) and augmented reality (AR) experiences with WebXR (Immersive Web, 2024).

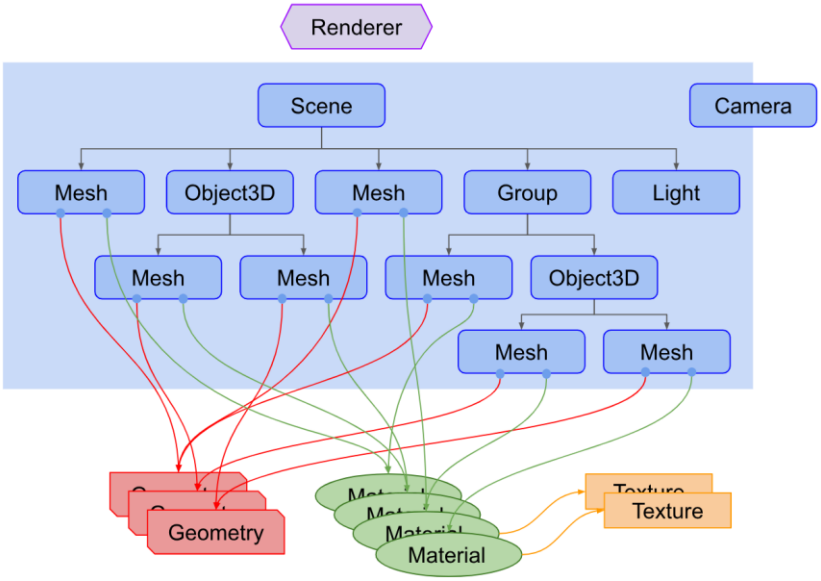

Figure 9. Diagram representing objects in a small three.js application (Three.js, 2024).

The image below shows a very basic animated 3D scene divided into two views, the one on the right highlights the current camera orientation and its field of view (orange) and the one on the left is the view from that camera looking at orbiting wireframe spheres.

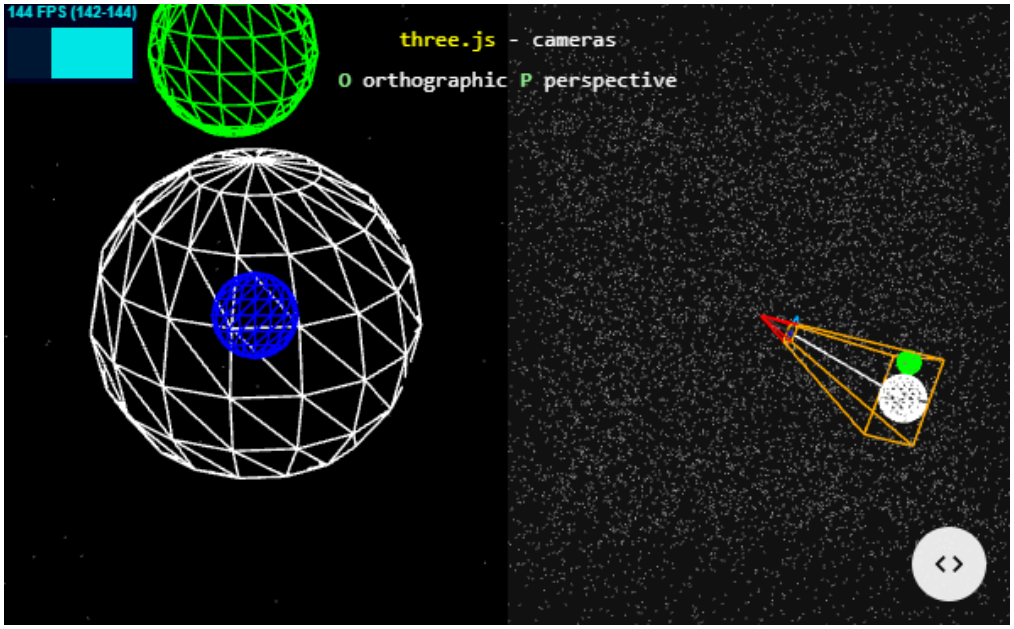

Figure 10. Example Three.js scene showcasing the camera object and wireframe rendering.

The next example (Figure 11) showcases a 3D model loaded from a file and rendered with more impressive details achieved with advanced reflective and transmissive materials.

Figure 11. Example Three.js scene showcasing advanced materials rendering.

The possible integration of 3D models from external software is a huge benefit for a digital twin project. This means that users who already have an existing model of their building could import that model into the 3D scene, which would provide an excellent context for the data. Three.js supports many file formats, like OBJ, PCD (for Point Cloud Data), STL (commonly used for 3D printing) and GLTF, which became a standard for exchanging models and textures (Dirksen, 2023).

2.5.5. Technologies choice

Below is a short summary of the technologies chosen for the project:

- **IDE:** Visual Studio Code
- **Version Control**: Git, GitHub
- **Project Management**: GitHub Projects
- **Containerisation**: Docker, Kubernetes, kind
- **Backend**: Django
- **Database**: PostgreSQL (Changed from MongoDB initially), CloudNativePG
- **Communication:** Kafka, WebSocket
- **Continuous Integration**: GitHub Actions
- **Frontend**: TypeScript, Three.js, Vite, Svelte/SvelteKit
- **Hardware**: Raspberry Pi 4B and DHT11 sensor

3. Design and Modelling

3.1. User Personas

To focus the design of the application, it was necessary to identify key personas and their main possible use cases for the project. Different personas in a building environment, such as facility managers, technicians or environmental engineers work with different components of the digital twin and within different scope. The following is an example of what different persona would require from the application and the digital twin:

Project owner - user which created the project:

- Edit project details,
- Delete project,
- Change user roles within the project,
- Add/Edit/Delete all elements of the Digital Twin.

Facility Manager:

- View project overview,
- Create maintenance tasks,
- Assign technicians to the maintenance tasks,
- Add/Edit/Delete assets,
- Generate reports on assets and sensors.

Technician:

- View the digital twin,
- Is assigned to maintenance tasks,
- View maintenance tasks list,
- Update/close maintenance tasks.

Environmental engineer:

- Add/Update/Delete sensors,
- Create maintenance tasks,
- View and analyse real-time and historical sensor data.

3.2. Planned Features

The list below shows features initially planned for the application:

- 3D visualisation of a building.
- Navigate freely around the scene with an orbit camera.
- Interact with the objects in the scene directly.
- Add rooms to create a simple building model from scratch.
- Upload a 3D model as GLTF file, serving as a more accurate reference for the user.
- Multiple levels in the digital twin of a building:
    - User able to switch between levels/floors,
    - Reference plane and the camera would move up or down and rooms and sensors belonging to the level would be shown and the rest hidden.
- View/Add/Update/Delete sensors in the model:
    - Each sensor would get automatically assigned to a room and a level,
    - The user will select the type of a sensor from the sidebar menu.
- View/Add/Update/Delete assets in the model:
    - An asset would be a general representation of any object that user wishes to track and monitor in the Digital Twin. This could be any equipment, but some examples would be heating system elements, electrical panels, fire extinguishers or smoke detectors.
    - Maintenance task could be associated with an asset.
- View/Add/Update/Delete maintenance tasks in the model:
    - Maintenance tasks can be associated with a sensor or an asset object.
    - Technician (user) can be assigned to a task.
- Sensor data dashboard:
    - Display charts with historical sensor data.
    - Display real-time data.
- Automated notifications and alerts:
    - Users able to set up thresholds for sensor readings.
    - Alerts for abnormal sensor readings.
    - Notifications about maintenance task deadlines.

- Sensor simulation
    - Users able to add a simulated sensor to the scene and analyse how it affects the overall system.
    - Notifications and alerts based on simulated data (to test the system).

Summary and reflection on the features implemented in the project can be found in section 6.2 (Achievements) of this document.

3.3. Considered Architectures

The application was designed with Apache Kafka at its core. Kafka Cluster is deployed and managed by Strimzi operator in Kubernetes cluster. Since the system was required to provide real-time data for the users, messages from Kafka cluster needed to be forwarded to the browser continuously. Long polling the Kafka Cluster was briefly considered, but Websockets are more efficient for this task. However, the latter approach introduces more complexity and poses a challenge – Websocket connection cannot be established directly to the Kafka Cluster out of the box and there is a need for a component which would bridge this gap.

One solution was to find an existing tool for this task, or if there is no right project available, to build a bespoke Websocket bridge for Kafka, which would likely turn into a separate project itself.

This rejected architecture variant is shown in the diagram below. This approach could be perhaps more tailored for the specific use in the SiteVisor project and could possibly be easier to manage but, as mentioned, it would require additional resources to implement, and it was decided not to go that route.

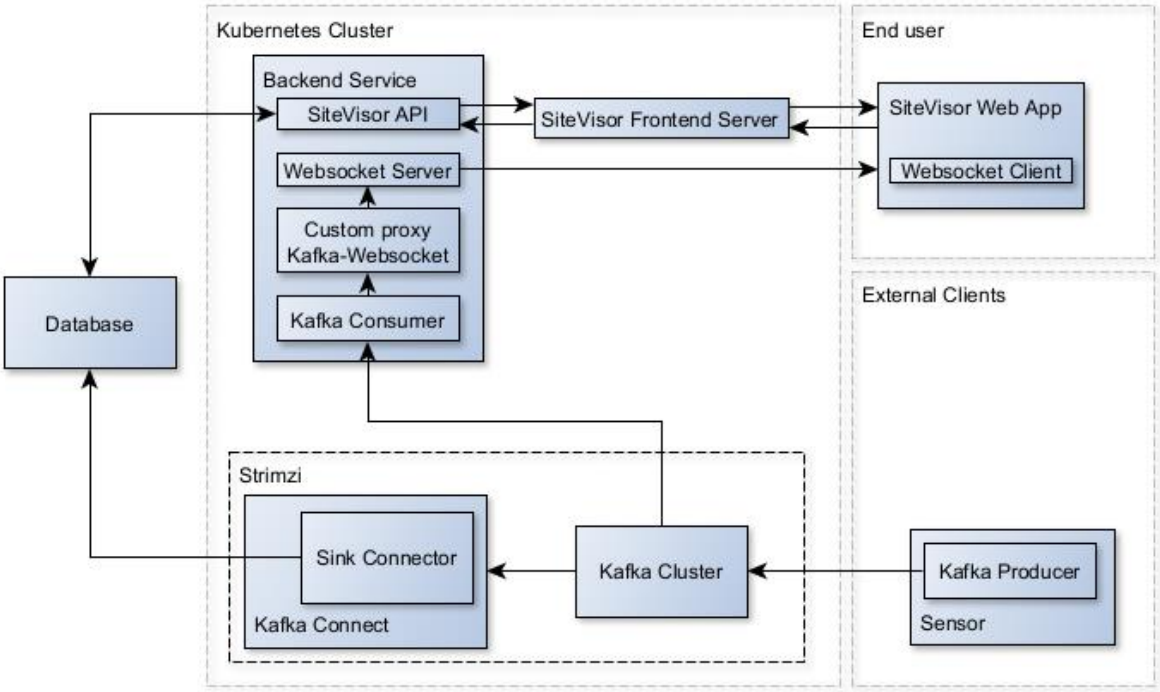

Figure 12. Application architecture overview - minimal variant 2.

Out of a few options available, I have settled on an existing solution named Kafka WebSocket Proxy (kpmeen, 2023). The project does not seem to be very popular with only 4 Stars on GitLab and 10 Stars on its mirrored repository on GitHub. However, it has pretty good documentation and is not abandoned like some of the other similar projects and seemed like a good fit for SiteVisor overall.

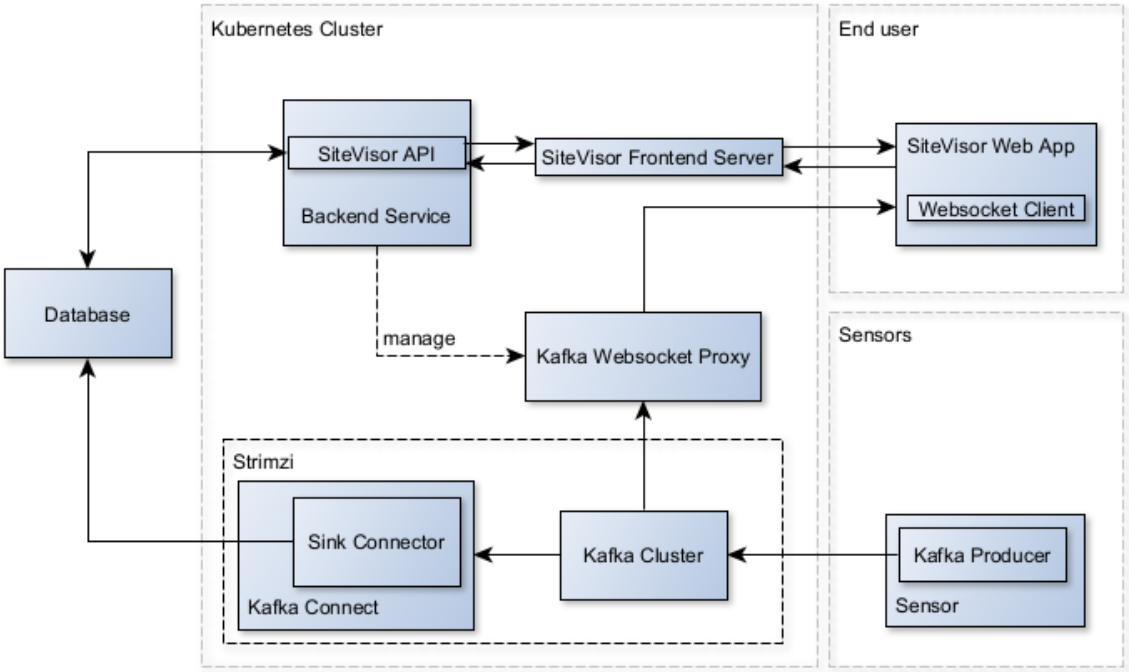

Figure 13. Application architecture overview - minimal variant 1.

The variant shown above is the one chosen to be implemented and it uses the Kafka WebSocket Proxy project mentioned earlier to establish a data stream between the Kafka Cluster and a WebSocket client in the browser.

Regardless of the chosen approach, both architectures can be additionally extended by enabling incoming data streams from sensors using various protocols like HTTP or MQTT. This can be achieved by incorporating Kafka Bridge for HTTP communication and MQTT Bridge for MQTT clients.

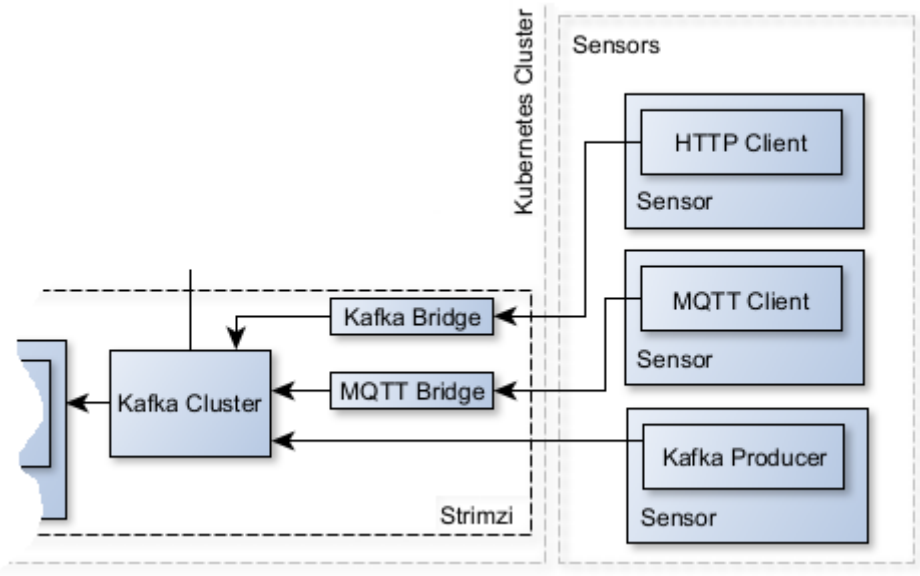

Figure 14. Sensor connection to Kafka Cluster – extended variant.

3.4. User Interfaces design

Besides the Digital Twin 3D View, the application was designed to have additional views/pages listed below:

- **Signup page** – allows users to create new accounts.
- **Login page** – enables registered users to log in and gain access to the application features and projects.
- **Add Project page** – users can create new projects, which will represent sites or buildings that the user wants to monitor and manage.
- **Projects list page** – displays the list of all projects accessible to the user and their role/access level.
- **Project manager page** – designed as a central hub for a specific project, where users can monitor and manage assets, sensors, and maintenance tasks.
- **3D Viewer/Digital Twin page** – focal point of the application where majority of user interaction will take place. Renders a 3D digital twin of the building or site, with sensors, assets, and real-time data heatmap overlays. Users can navigate through the virtual environment and interact with the objects in the scene to view more detailed information and add new objects.
- **Historical data page** - presents an overview of the archived sensor data, where users can view trends and analyse patterns over time to aid with decision making.
- **Room page** - focuses on a specific room within the building, displaying details and sensor data related to that specific space. Also lists all assets located in the room.
- **Sensor list page** - displays a list of all sensors associated with the project, allowing users to view and manage sensor configurations and data.
- **Sensor details page** - provides detailed information about a specific sensor, including real-time data readings, historical trends, and configuration settings.
- **Asset list page** - displays a list of all assets within the project, providing a quick overview of all objects of interest to the user.
- **Asset details page** – presents detailed information about a specific asset, such as equipment specification, maintenance history and current state.

- **Maintenance tasks list page** - displays a list of tasks relevant to the user, grouped in categories.
- **Maintenance task creation page** - allows users to create maintenance tasks, specifying details like task type, priority, and assign technicians.
- **Maintenance task details page** - detailed view of a specific maintenance task, including status, assigned technician and comments on the task.

Each of the detail view pages, like individual sensor, maintenance or asset page should also link the user to the specific location in the 3D digital twin.

Wireframes in the following images were created using Figma design tool and show concepts of the key pages for the project.

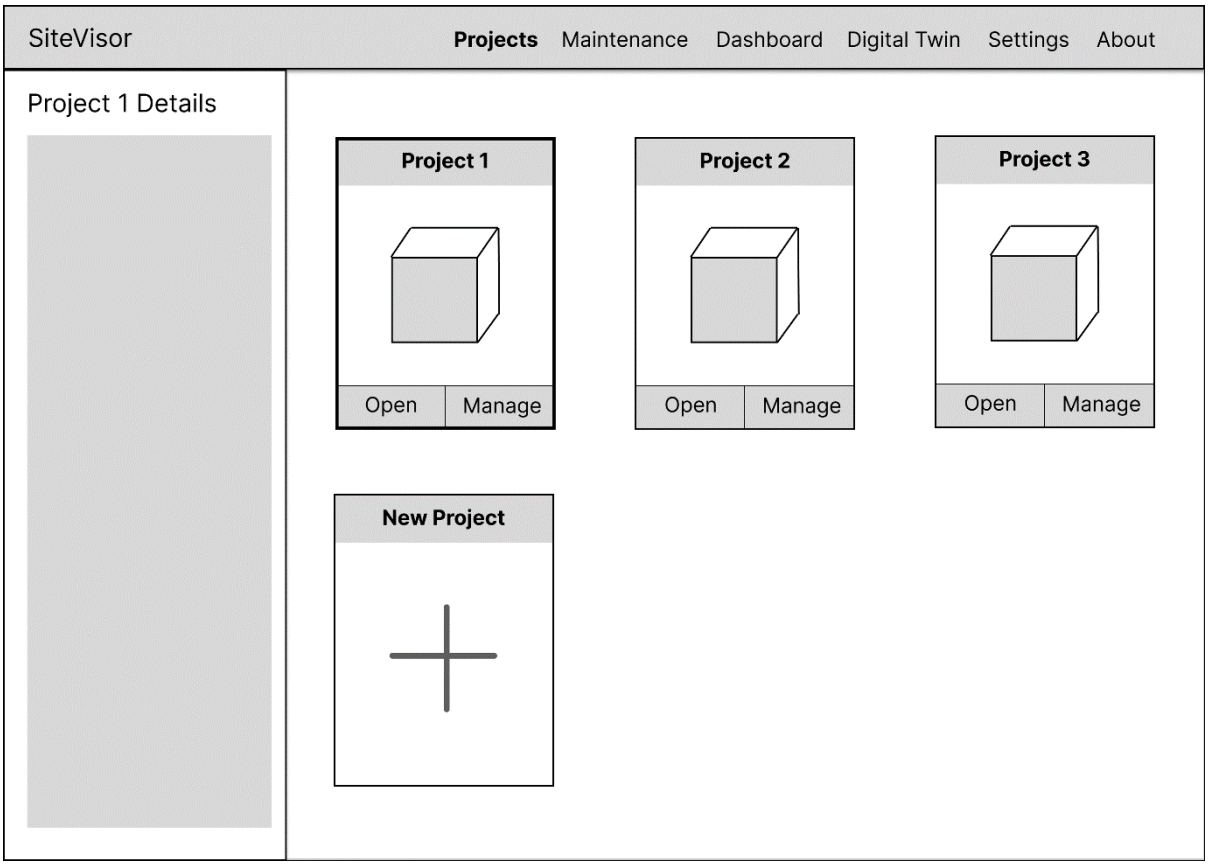

Figure 15. Projects View wireframe.

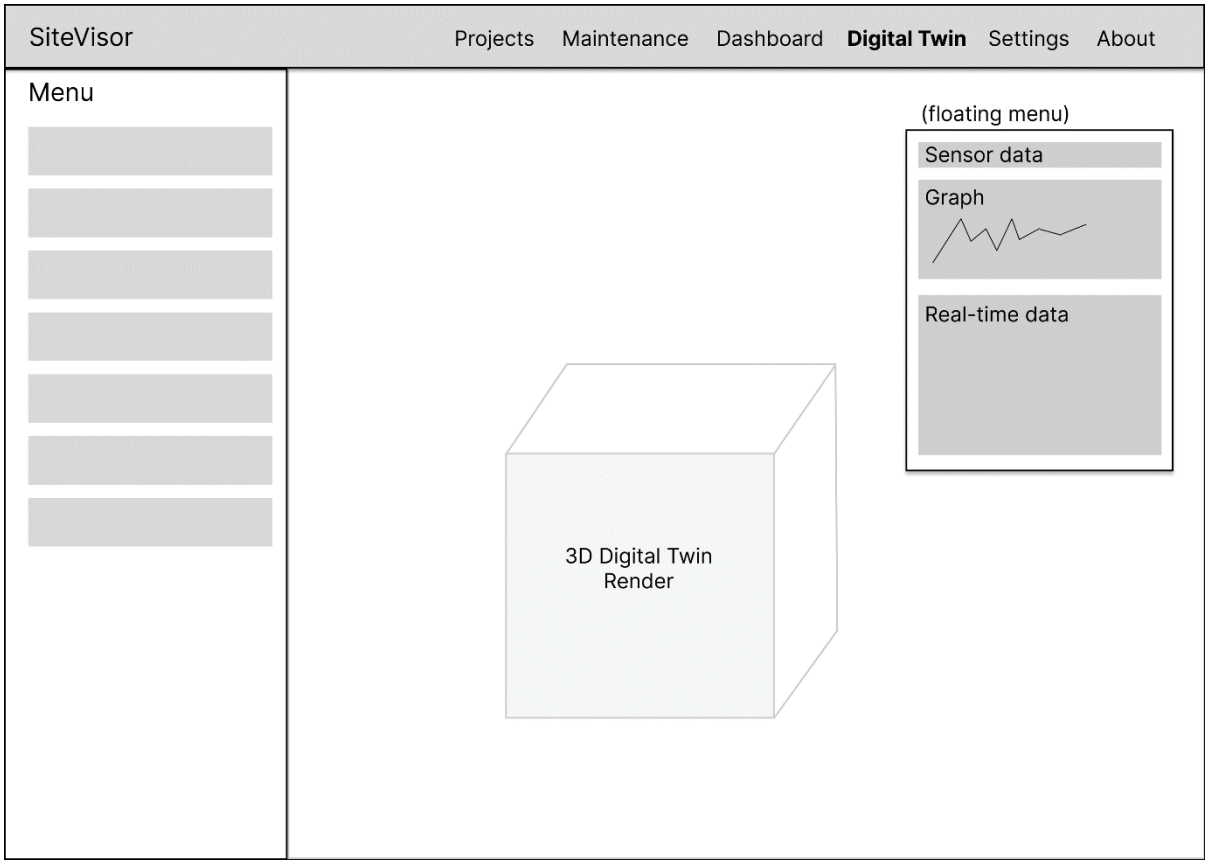

Figure 16. 3D Digital Twin View wireframe.

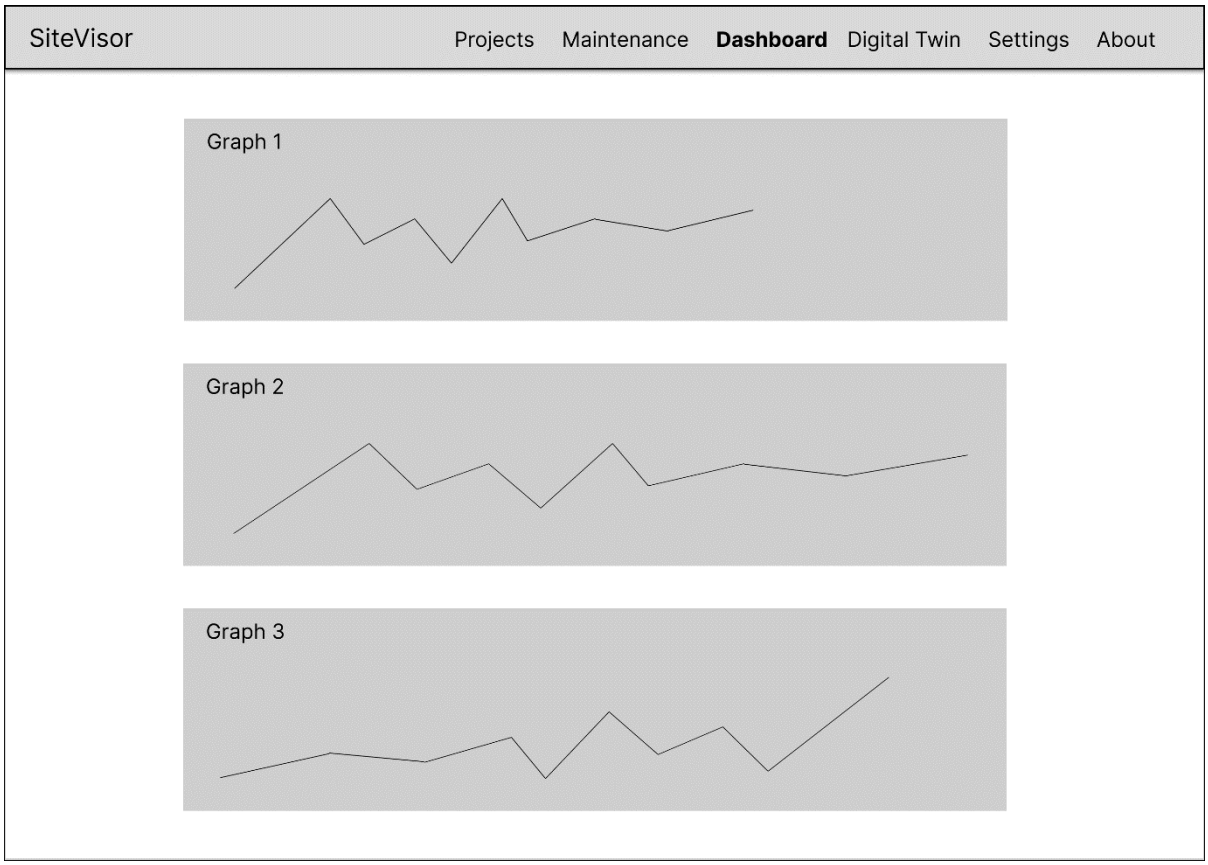

Figure 17. Dashboard View wireframe.

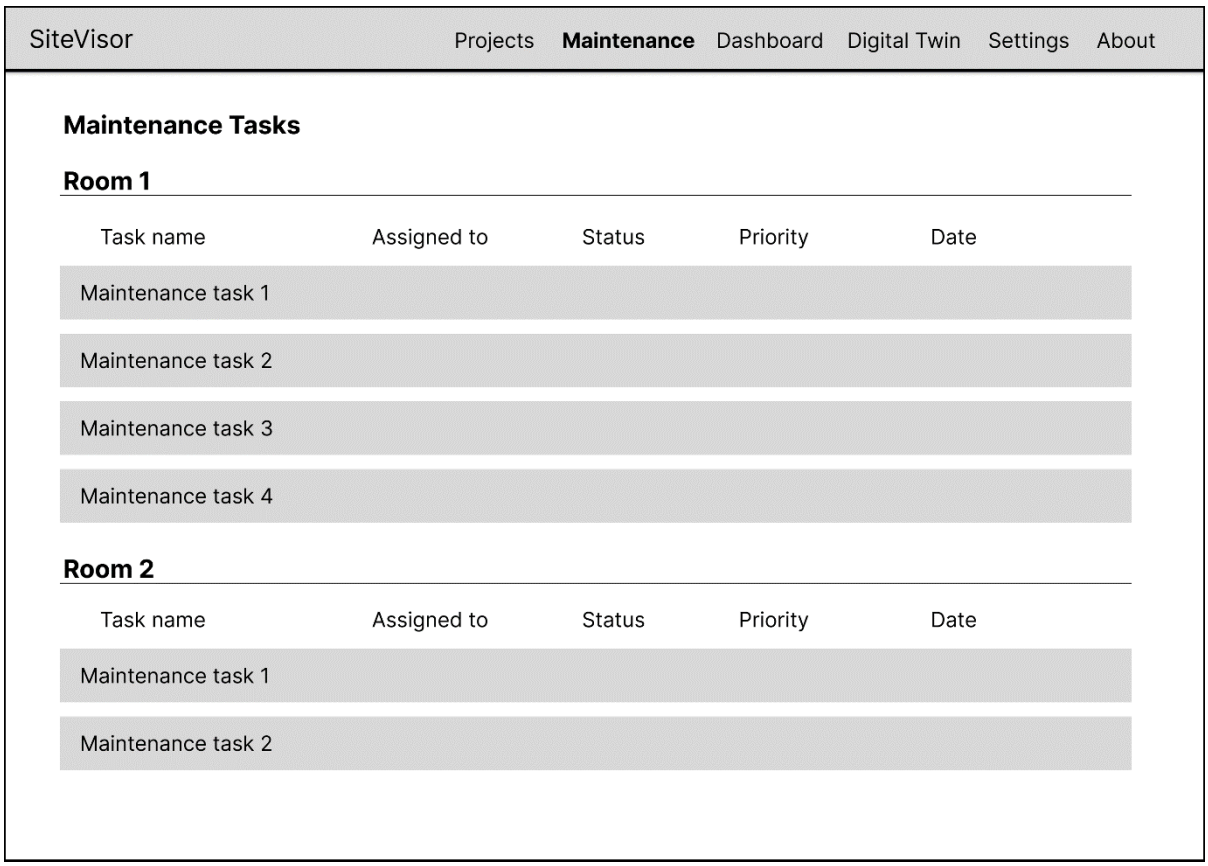

Figure 18. Maintenance Tasks View wireframe.

4. Planning

4.1. GitHub Projects

GitHub Projects was used as the project management tool. A GitHub Project consists of a spreadsheet, task-board (Figure 19) and road map (Figure 20) which integrates with GitHub issues and pull requests automatically (GitHub, 2023). Since GitHub is the platform of choice for the SiteVisor project version control, GitHub Projects was a fitting solution to manage and keep track of the development progress.

Issues and Pull requests from all repositories were added to the SiteVisor project, which can be seen below.

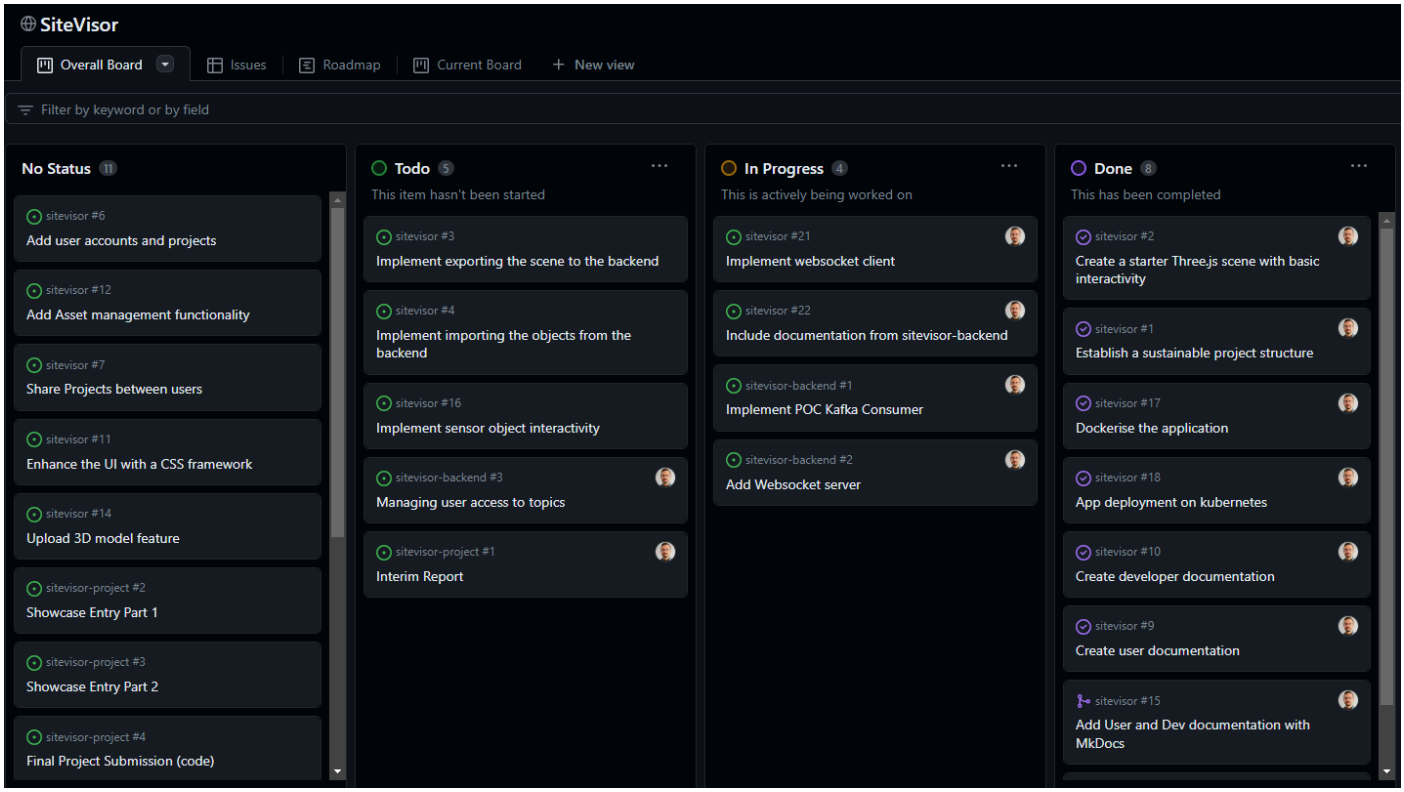

Figure 19. Project SiteVisor agile board.

Figure 20 shows a fragment of the Project Roadmap which was continuously updated while planning and progressing on the project. An export of the issues from the roadmap is included at the end of this report as Appendix A – Export from GitHub Project Roadmap (formatted).

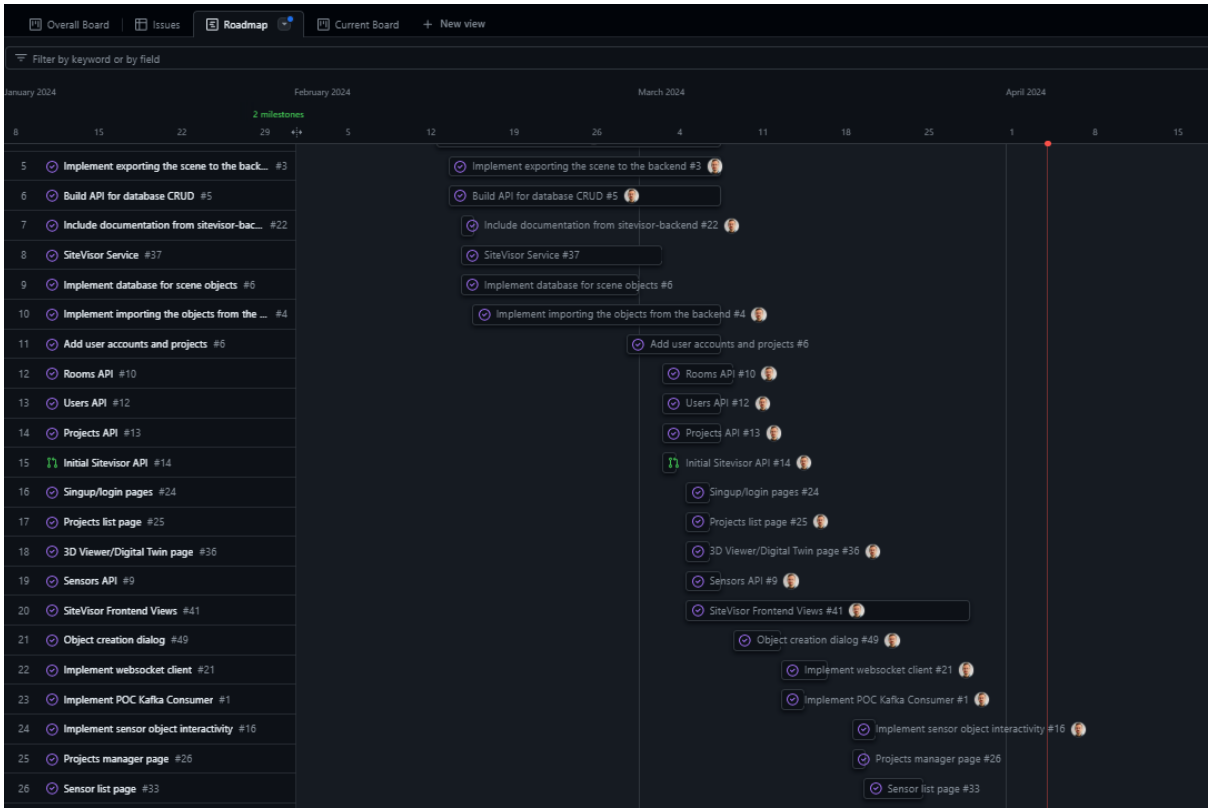

Figure 20. Project SiteVisor roadmap in GitHub Projects.

4.2. Development Process

During the development of the project, I adapted a certain rhythm to the process. This is nothing remarkable perhaps and based around Git/GitHub Flow process, but it was crucial to keeping the progression at a steady pace, especially in the very busy final phase of the implementation. Below is a brief outline of the usual steps:

1. Check the roadmap in GitHub Project to see what is planned for the day.
2. Make any changes to the schedule if necessary.
3. Pick an issue to work on based on the priorities set as labels.
4. Create a branch for the feature.
5. Work on the implementation and commit changes locally.
6. After some progress was made, push changes to the feature branch on remote.
7. Open a Draft Pull Request on GitHub and write a short description – this allowed me to step back and think again on what is already done and what needs to be done.

8. Continue working on the implementation, pushing commits to the remote branch regularly.
9. When stopping work for a longer time, leave a comment stating what still needs to be done. This helps when coming back to work on the issue.
10. After the feature is ready, change the Pull Request PR status to *ready for review.*
11. Review the files changed, ticking the *Viewed* checkbox after checking each file.
12. Leave a Review comment for yourself to mark the completion of the task.
13. Merge the feature into the *main* branch.
14. Closing the issues is handled automatically through the keywords (“Closes”, “Fixes” etc.) in the pull request.

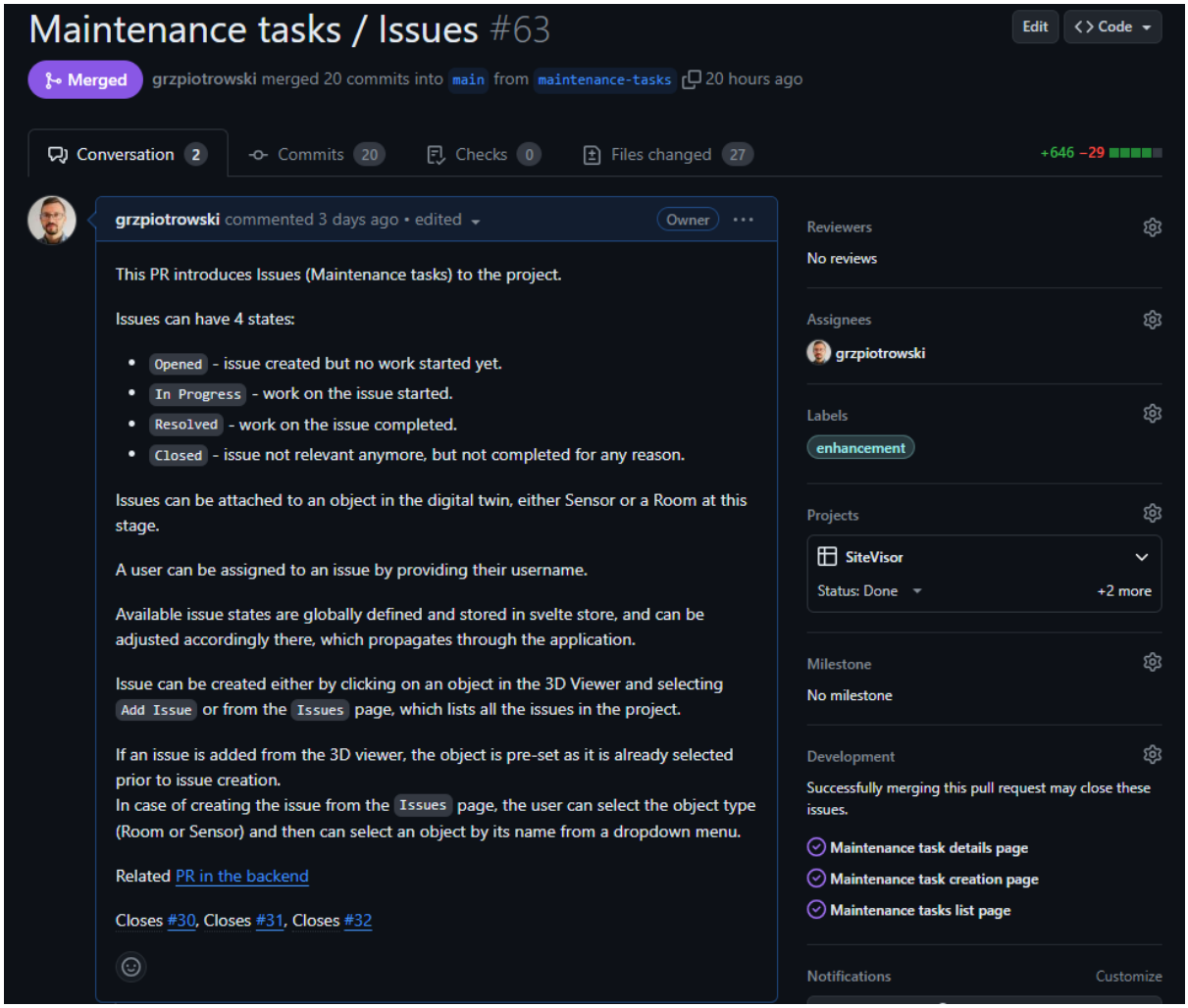

Figure 21. Example of a Pull Request in the project.

The image above shows an example of a Pull Request in the project. The detailed descriptions provide a great documentation of the project throughout its development.

I have also used Issues with a custom *epic* label, to keep track of the progress on a larger chunk of work which was broken down into multiple smaller issues. The *epic* issue would contain these issues listed with checkboxes. A very convenient feature of GitHub is that completed and closed issues would get automatically marked as done in that list.

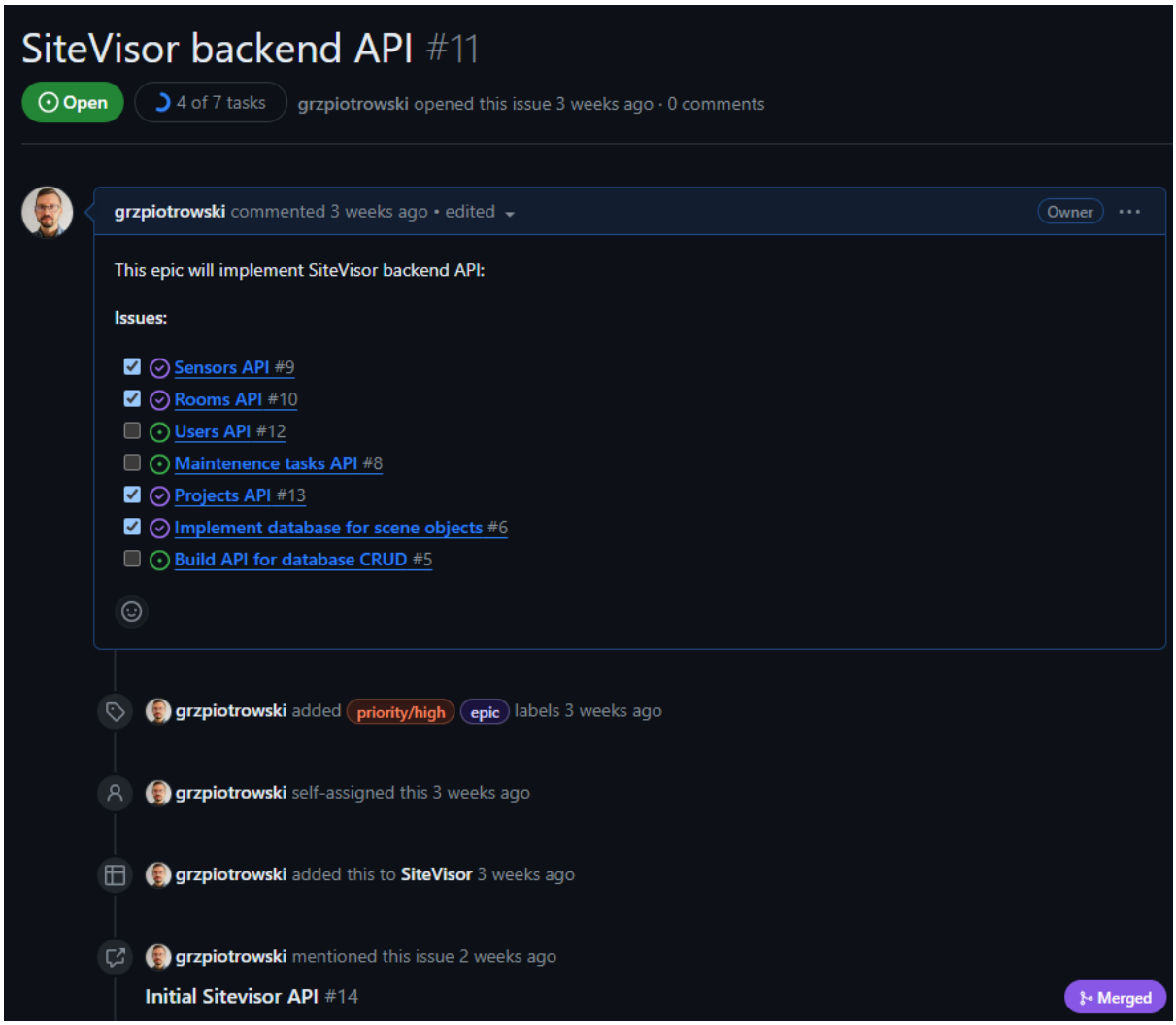

Figure 22. Example of an "epic" GitHub issue used in the project.

5. Implementation

5.1. Prototyping

Towards the end of the research phase of the project, I started the prototyping work to gauge the feasibility of the planned features. This was both for the frontend, focusing on the 3D Viewer aspect, and the backend, learning more and deploying Kafka and testing data pipelines.

Like many projects based around a 3D environment, it all started with a cube, as shown in the image below.

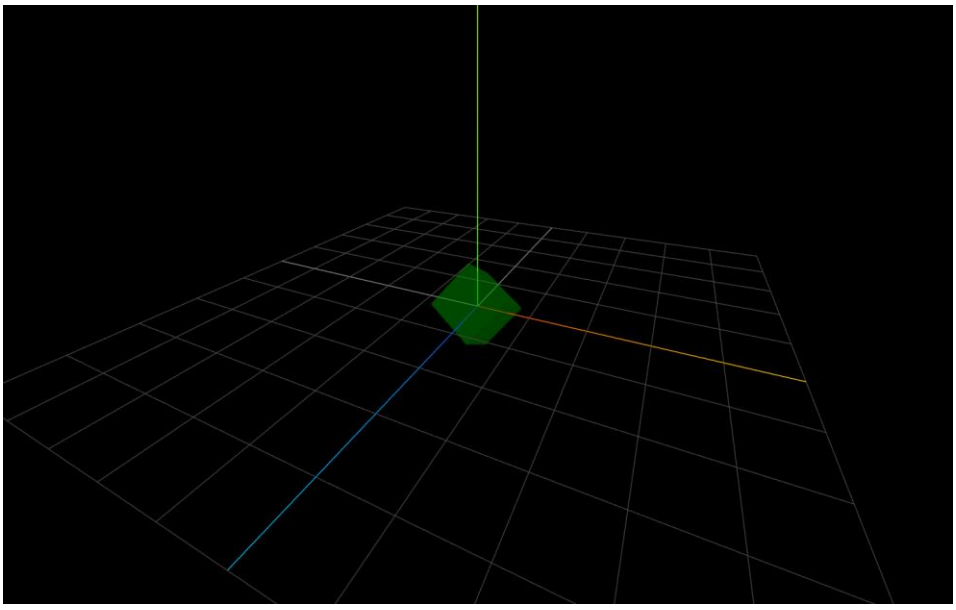

Figure 23. Initial Three.js scene with a Helper Grid, Coordinate Axes, and a simple cube.

The prototype has then evolved to include more interaction like mouse controls of the camera and basic interaction with the Helper Grid which became a reference plane for the cursor to interact with. At every frame, the intersection between the reference plane and the ray originating from the camera and passing through the cursor is checked in the animation loop. The same principle applies for checking if any other object from the 3D scene is currently under the cursor.

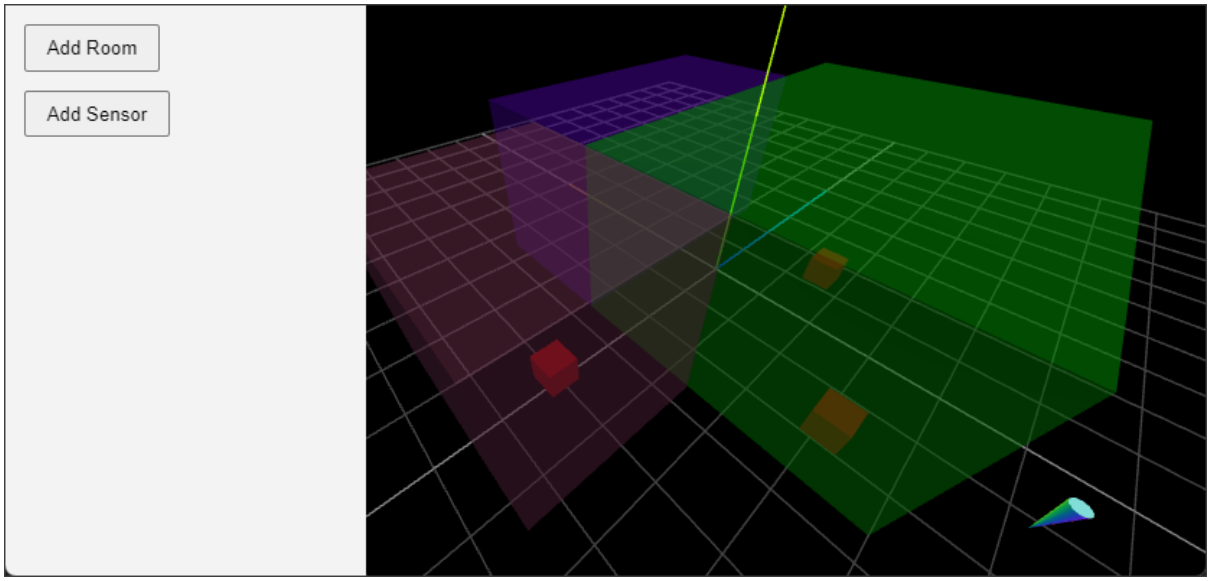

Figure 24. Early prototype of the interface for creating rooms and sensors.

These basic control and interaction components allowed to experiment with the implementation of adding objects to the 3D scene and changing the state of the 3D environment through menu buttons. The image above shows the sidebar menu and adding *Room* and *Sensor* objects to the scene. At this stage there was no data persistence yet.

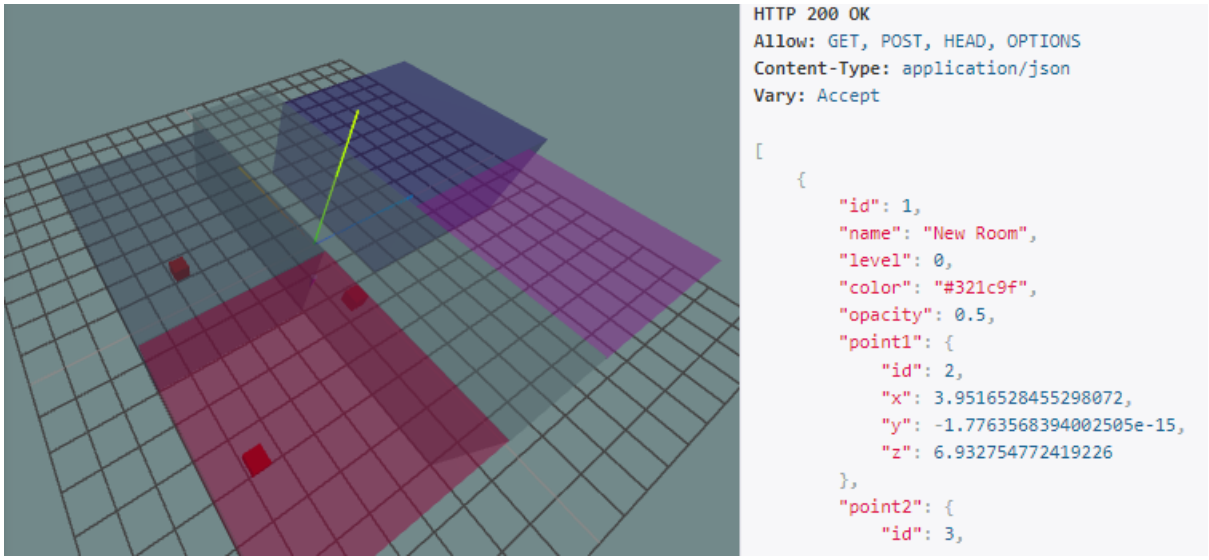

Figure 25. Saving and loading the rooms and sensors from the backend.

The next step in the prototyping phase involved saving and loading the objects to and from the backend database. At this stage the

The initial prototype for the backend involved deploying a Kafka cluster using Strimzi Quick Starts documentation (Strimzi, 2024) and testing the connection, by running a simple producer to send messages and a consumer to receive them.

5.2. Final architecture

The final architecture for the project realises the architecture proposed initially in section 3.3. The current setup supports the real-time data connection from sensors and because of Kafka, the backlog of the data is also delivered to the user even after the connection is re-established after it has been lost.

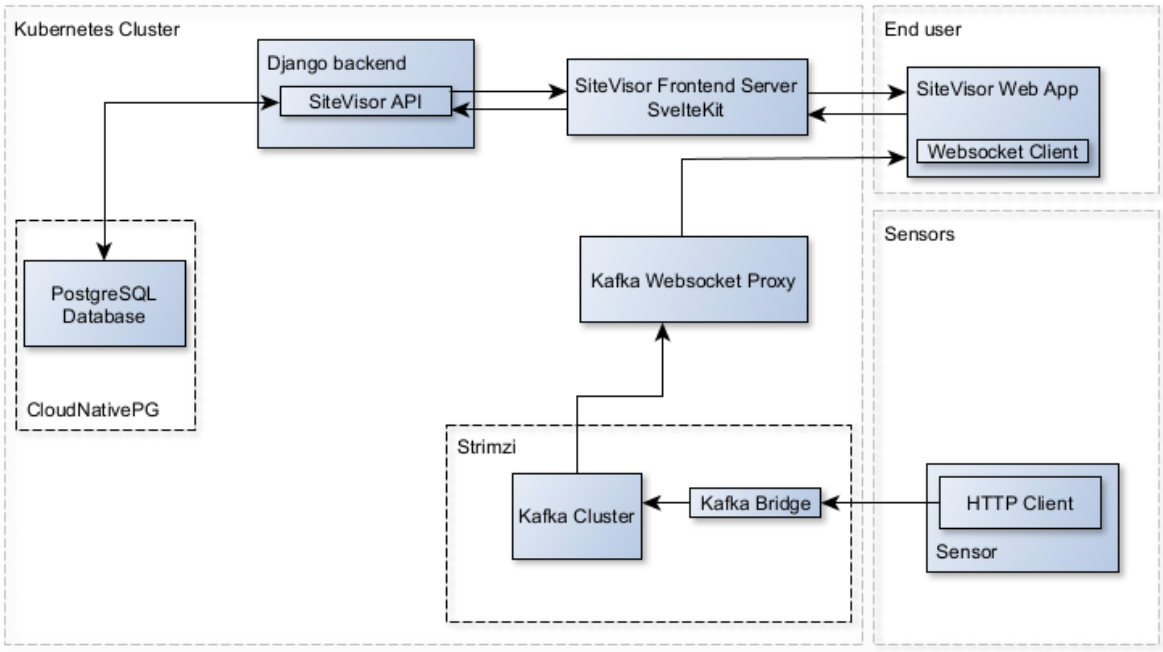

Figure 26. Key components of the implemented project architecture.

One of the components missing from the original design is the Sink connector from Kafka Cluster to the Database, which was supposed to archive the data for future use. Due to difficulties in configuration and prioritising other tasks, this feature was deferred for now. Also, the planned Sensor component was envisioned to use Kafka protocol, but I could not get my ingress configured correctly for that to happen, so HTTP Client on the sensor side was used, coupled with Kafka Bridge in the cluster to receive the data and pass them to the broker.

5.2.1. Database and Data Model

PostgreSQL database was chosen to store the object data. The database was deployed inside Kubernetes cluster using CloudNativePG operator. Deployment was based on the instructions in the documentation (CloudNativePG, 2024). The UML diagram below shows the implemented database model.

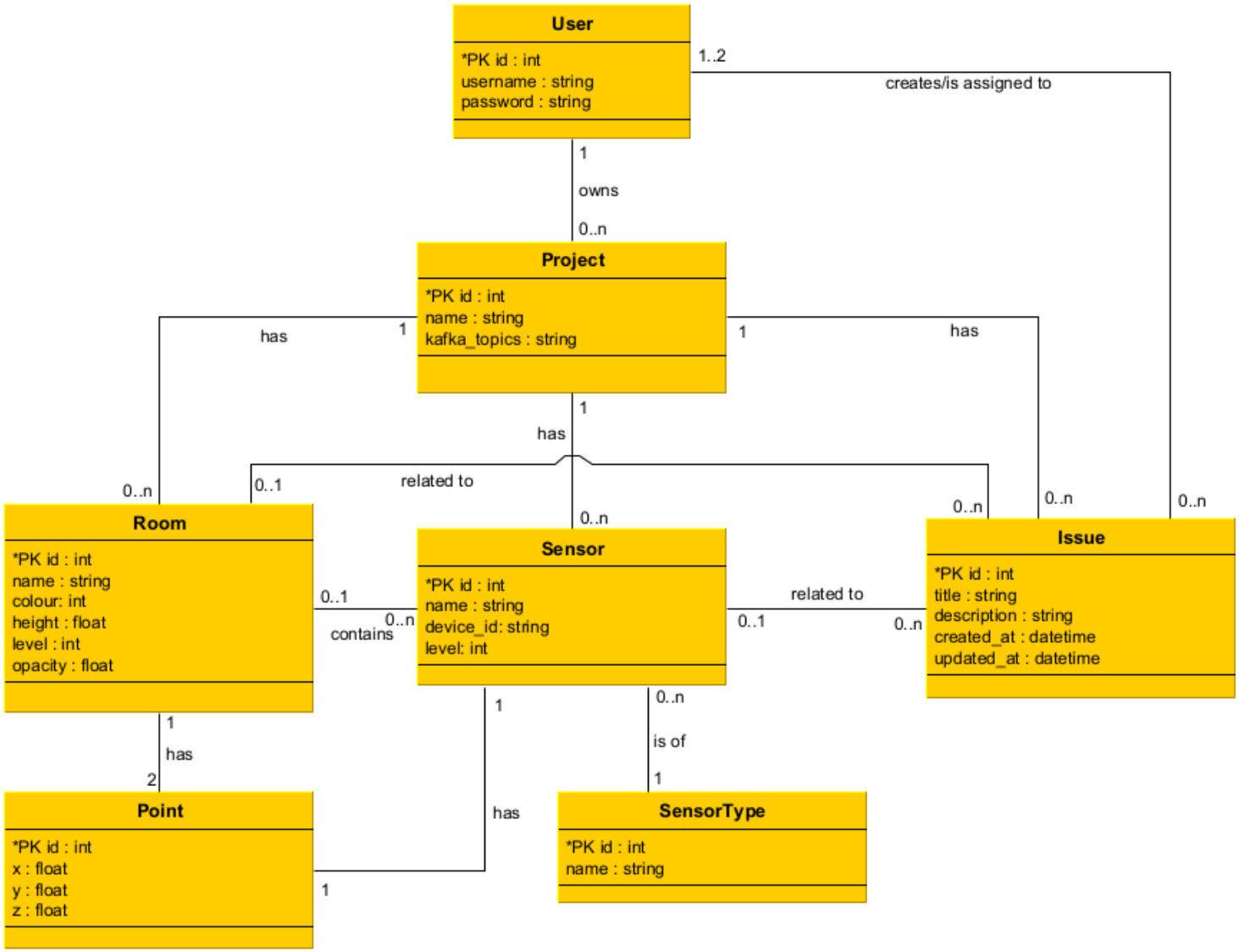

Figure 27. Database UML diagram.

5.2.2. Digital Twin Class Diagram

To keep the visualisation in the frontend maintainable, an object-oriented approach was taken for implementing different objects of the digital twin (Figure 28). Object3D and Mesh are three.js classes and provide a base for the SiteVisor objects, from which custom classes

representing different geometries inherit. These are Volume, Point3D and Plane and provide a solid base for the objects like the Room and Sensor.

These classes are instantiated in the browser on the 3D Viewer initialisation and are constructed using the data stored in the database.

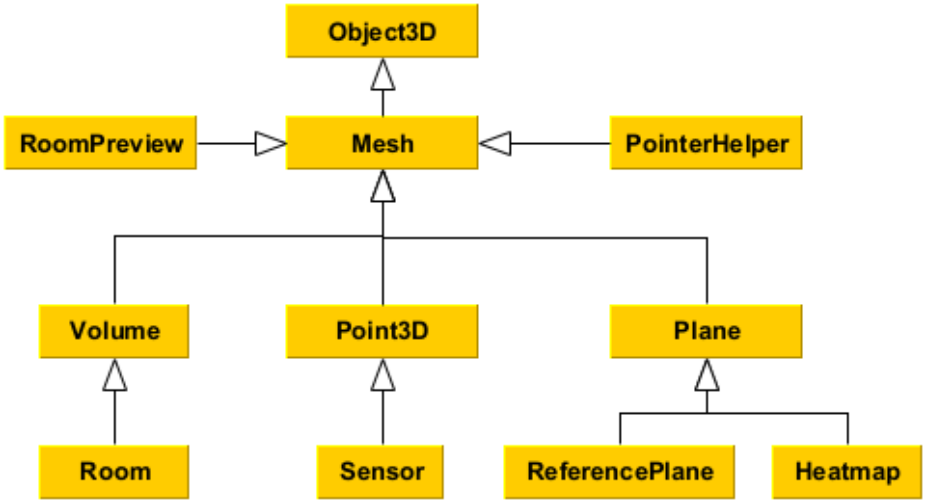

Figure 28. Class diagram for the Three.js scene objects implementing the Digital Twin model.

5.2.3. Real-time Data flow

The diagram below provides an overview of the data flow from the physical sensor to its digital counterpart through Kafka components and the Websocket Proxy.

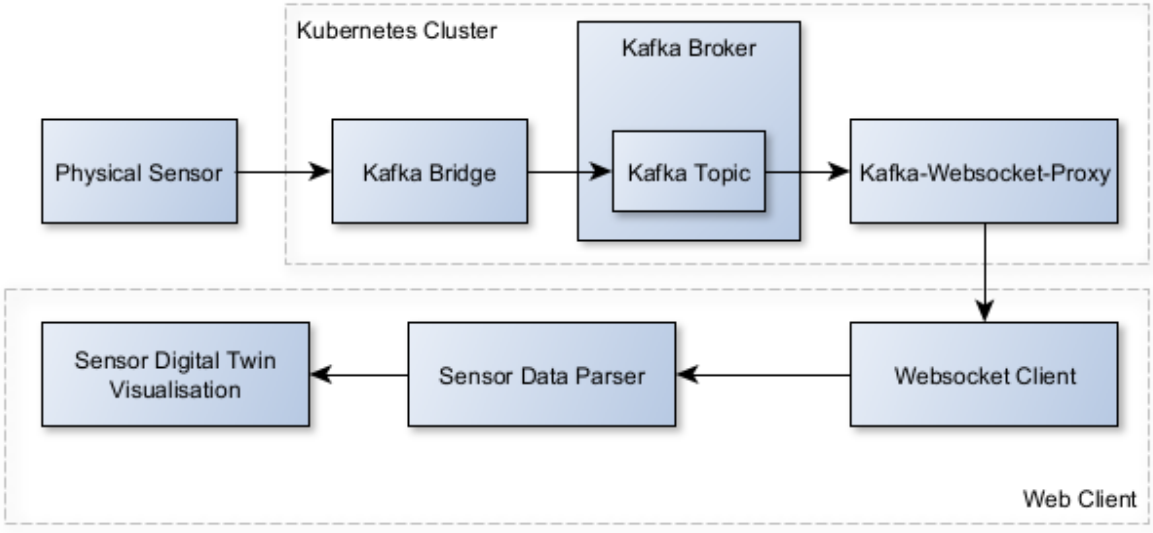

Figure 29. Sensor real-time data flow diagram.

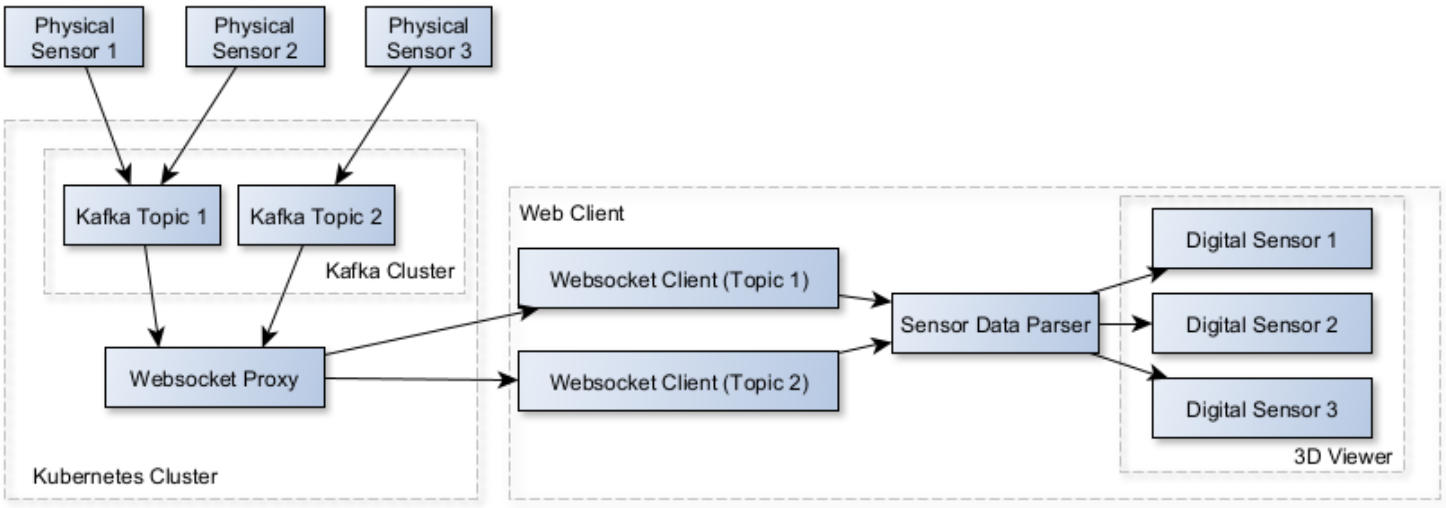

Figure 30. Realtime sensor data flow diagram from Kafka Topics perspective.

The diagram above attempts to show the same data flow but with the focus on Kafka Topics. This was also an important aspect in the project development. Each Physical Sensor sends data to a specified Kafka Topic. Kafka-Websocket-Proxy serves each topic in a separate websocket connection, so we create a Websocket Client per topic. The data is then parsed to unpack the messages and assign it to a correct Digital Twin Sensor based on the id in the message (Figure 31).

```
{
  "records":[
    {
      "value":{
        "sensor_id":"sensor-123",
        "sensor_type":"temperature",
        "data":{
          "value":20.23,
          "unit":"C"
        },
        "timestamp":"2024-04-02T16:09:37Z"
      }
    }
  ]
}
```

Figure 31. Example of a message send from the sensor.

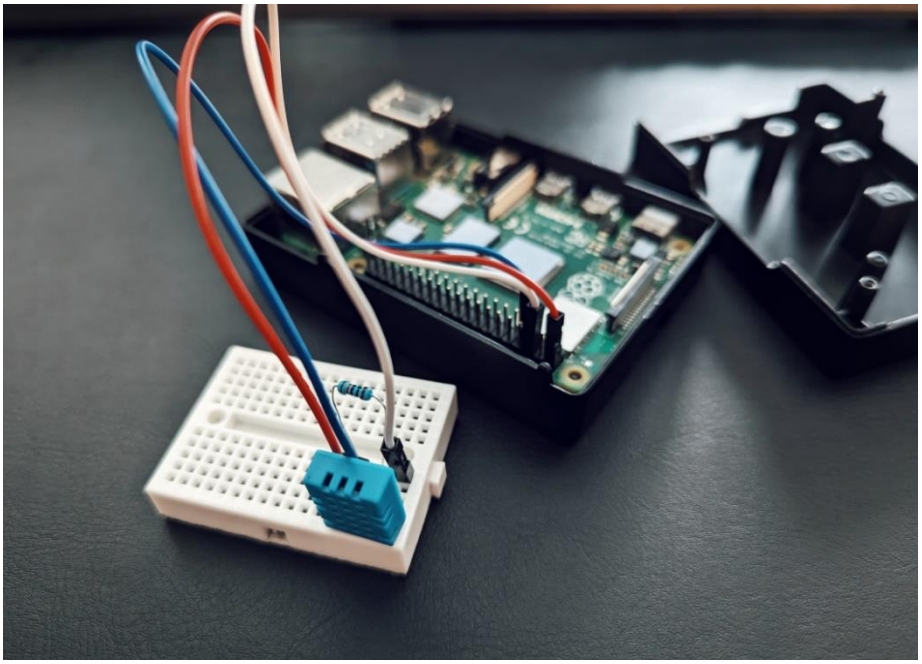

Figure 32. Minimal hardware setup for testing, using DHT11 temperature and humidity sensor with Raspberry Pi 4B.

To demonstrate a physical device working with the system, a basic setup using Raspberry Pi 4B and DHT11 temperature and humidity sensor was built (Figure 32) and a simple code to read and message to data was developed.

During the development and testing however, scripts simulating the sensors with randomly generated data were used.

Status of the Websocket connection to each topic is indicated in the application through a component shown below. Any topic can be clicked to restart the connection if there is any issue with it.

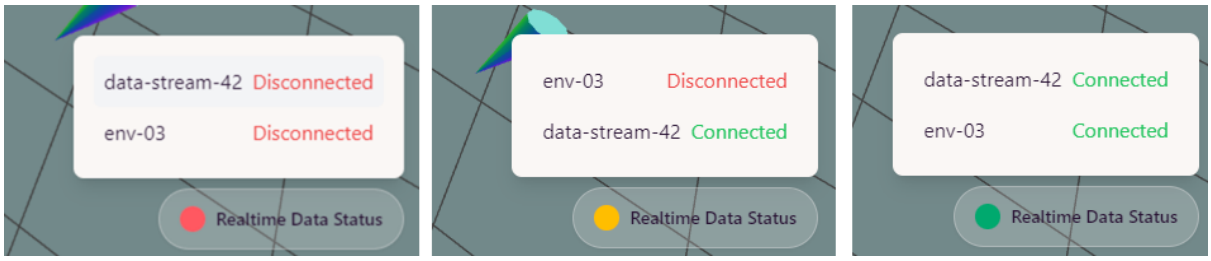

Figure 33. Realtime Data Status listing all Kafka topics connections configured.

5.3. Implemented Features

5.3.1. User signup and login

User authentication is handled by Simple JWT plugin for the Django REST Framework.

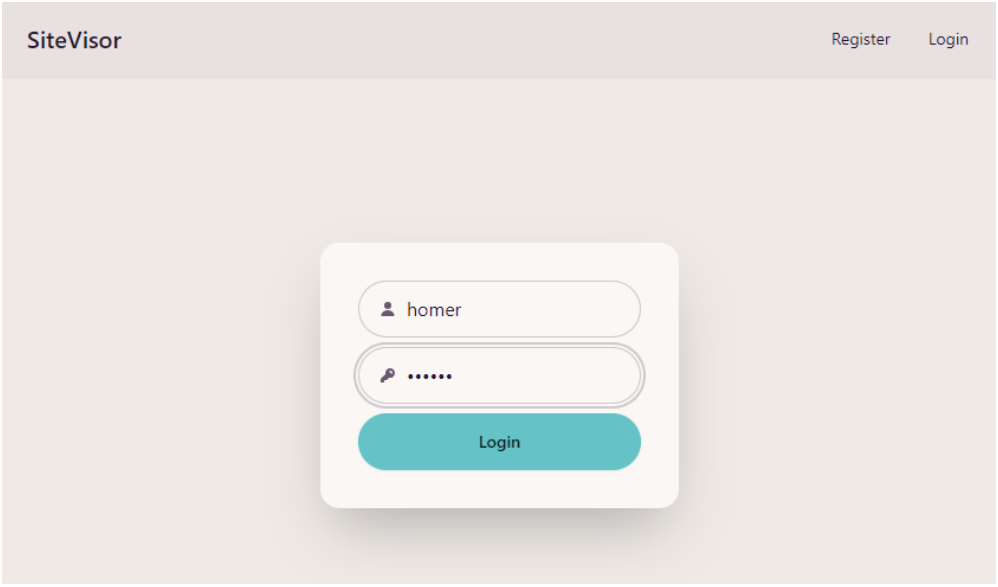

Figure 34. User login page.

5.3.2. Creating and managing Projects

After logging in, the user is presented with a list of projects owned by them (Figure 35), where they can also create a new project by interacting with the last card on the list.

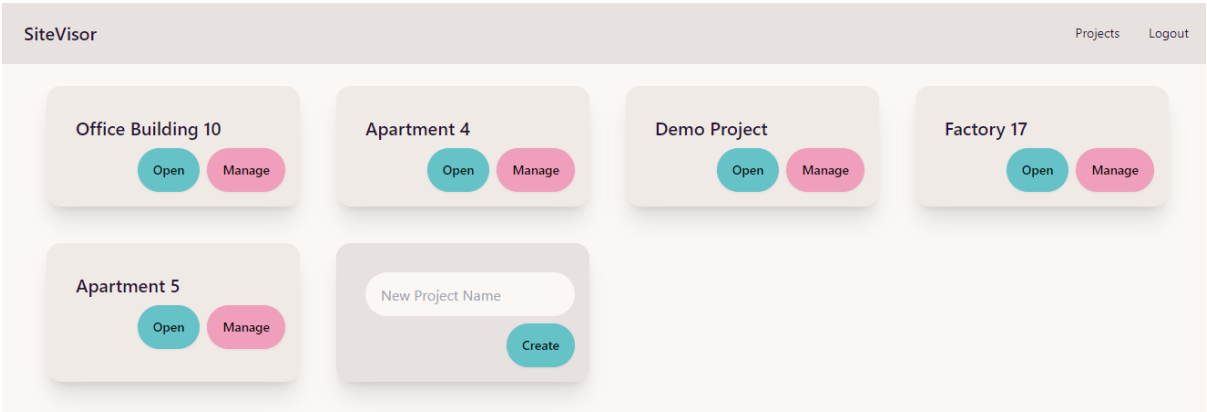

Figure 35. Project listing page enabling to create a new project.

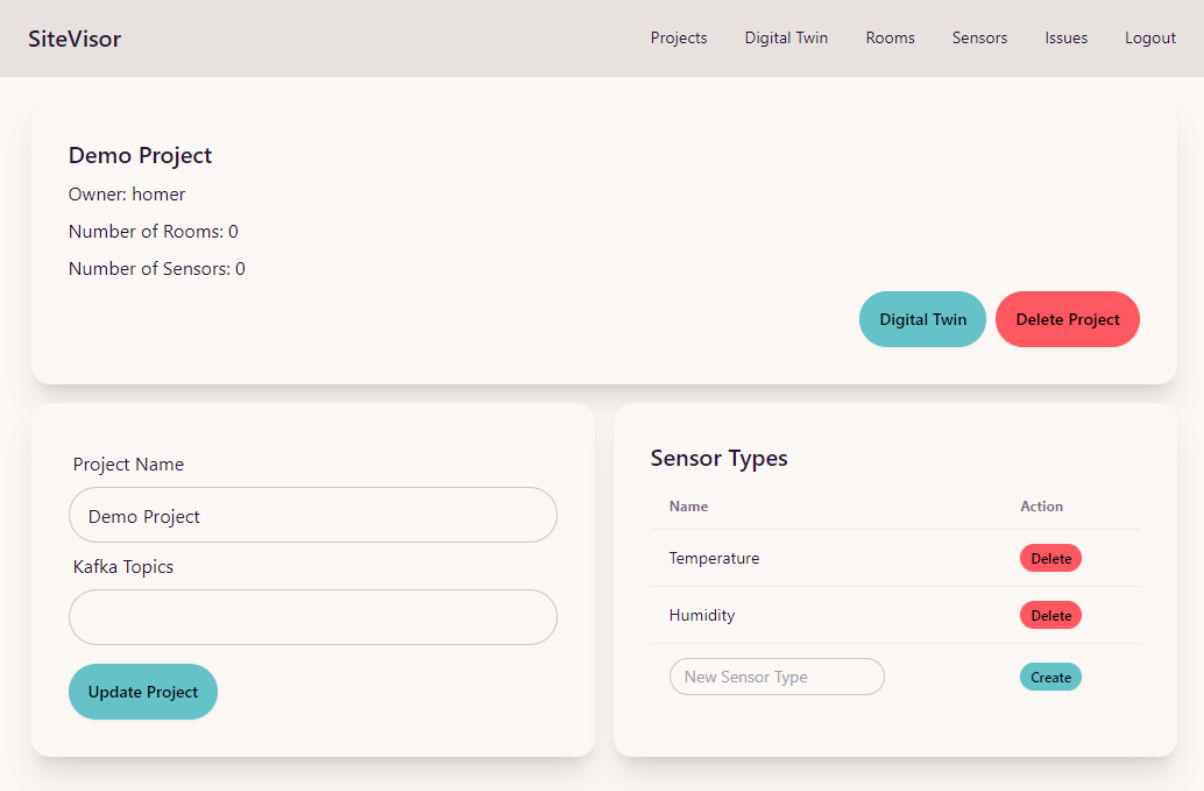

Figure 36. Project management page.

After creating the project, we can use the *Open* button to go into the Digital Twin view or click the *Manage* button to enter the project management page (Figure 36). In the latter, we can see basic details about the project and get the option to update the name and configure Kafka topics for the project. Types of sensors can also be configured here, with *Temperature* and *Humidity* types added by default.

The image below shows an example of customised project configuration with two Kafka topics (*data-stream-42* and *env-03*) specified by entering the comma separated topic names in the appropriate field and the *Pressure* sensor type added to the project.

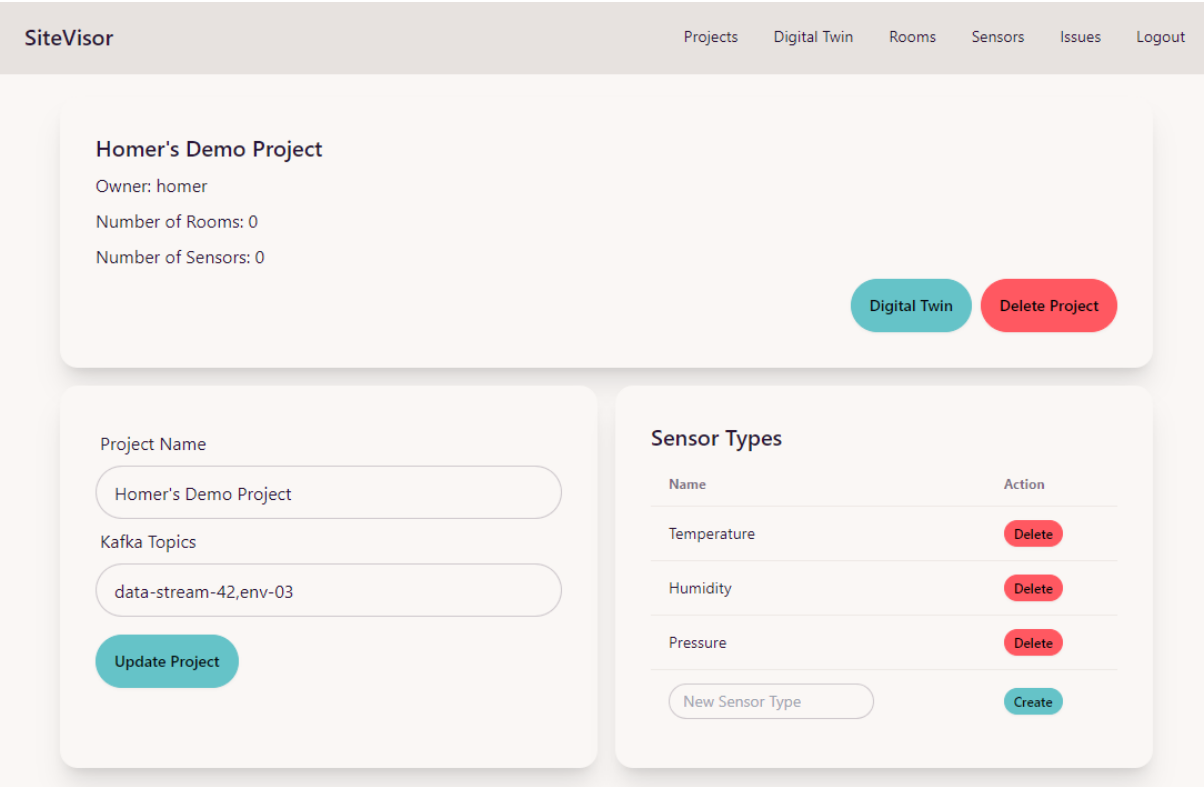

Figure 37. Project configuration example with two Kafka topics and additional sensor type.

After configuring the project, we can move on to the 3D Viewer by using the *Digital Twin* button. The 3D Viewer page is shown in the image below.

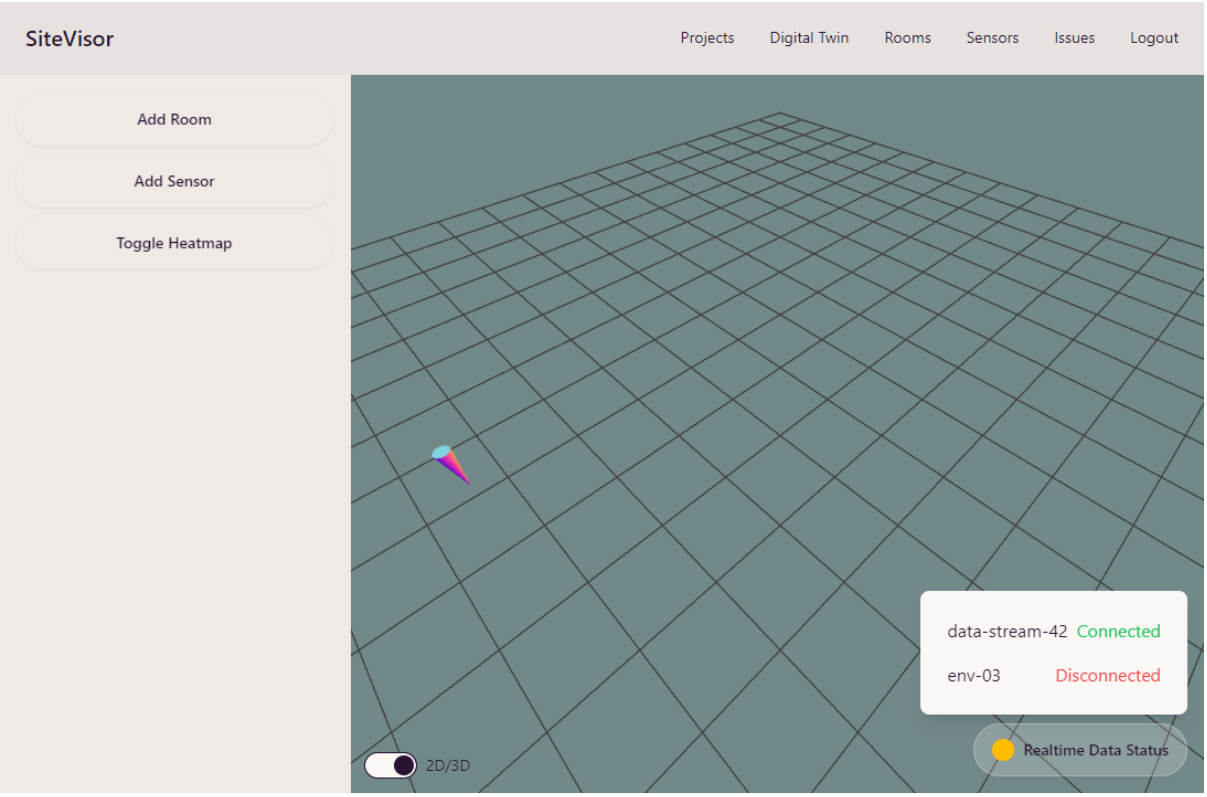

Figure 38. Digital Twin/3D Viewer page in a new project.

5.3.3. Adding and managing Rooms

One of the key implementations is the ability to add objects representing rooms to the 3D scene. Each time the *Add Room* form is opened, it randomises the colour for the inserted room, to help distinguish them from one another.

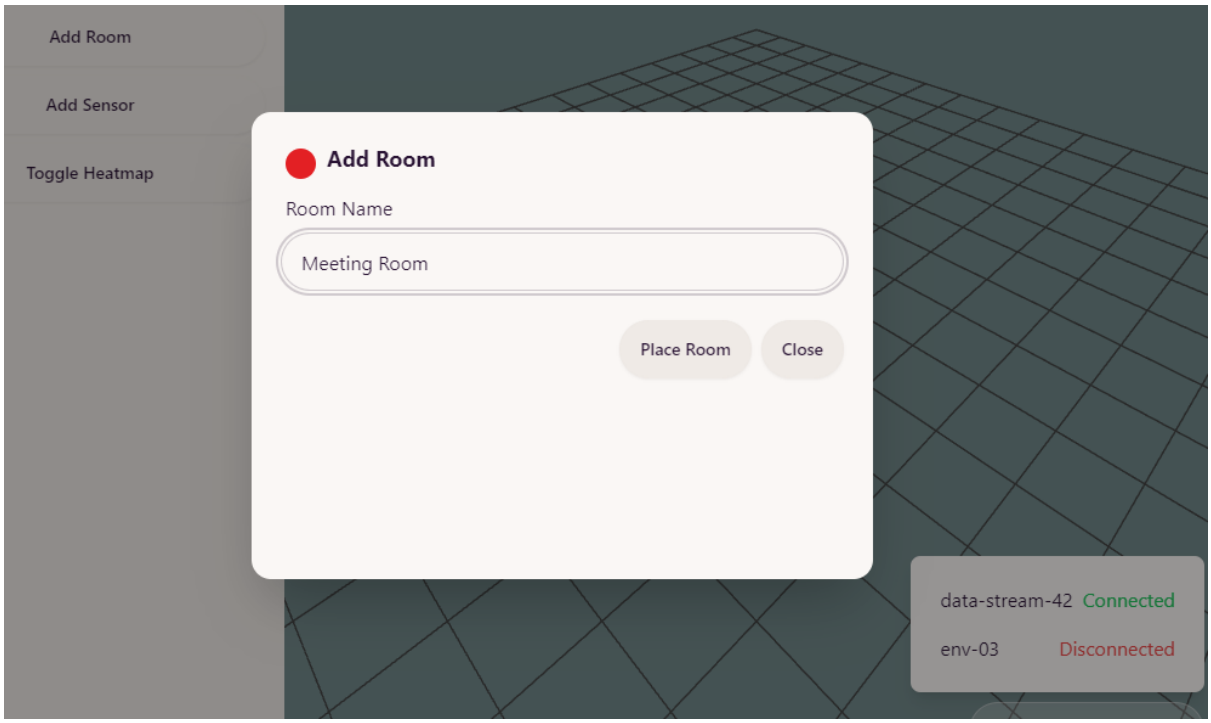

Figure 39. Room creation form.

The user can also click the colour icon to open a colour picker (Figure 40) and select any colour of their choice.

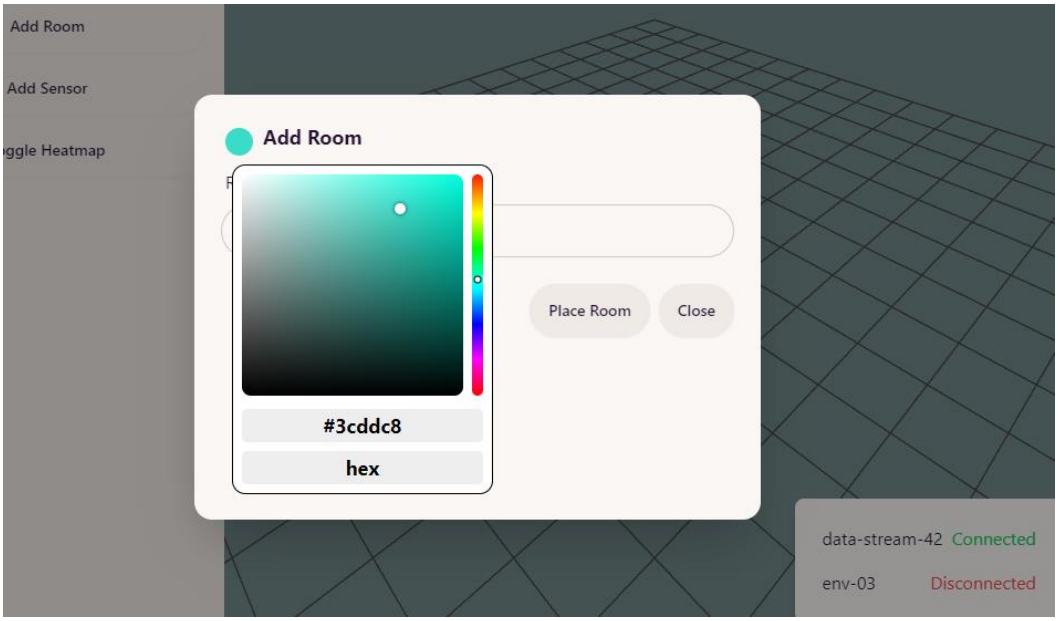

Figure 40. Room colour picker.

When creating a new room, the room geometry switches to a 2D mode, with the existing objects flattened, which makes it easier to see the floor layout. The user picks two points showing the extents of a new room, after they select the first point, a green preview of the new object is shown, dynamically updating as the cursor moves around the scene (Figure 41 and Figure 42).

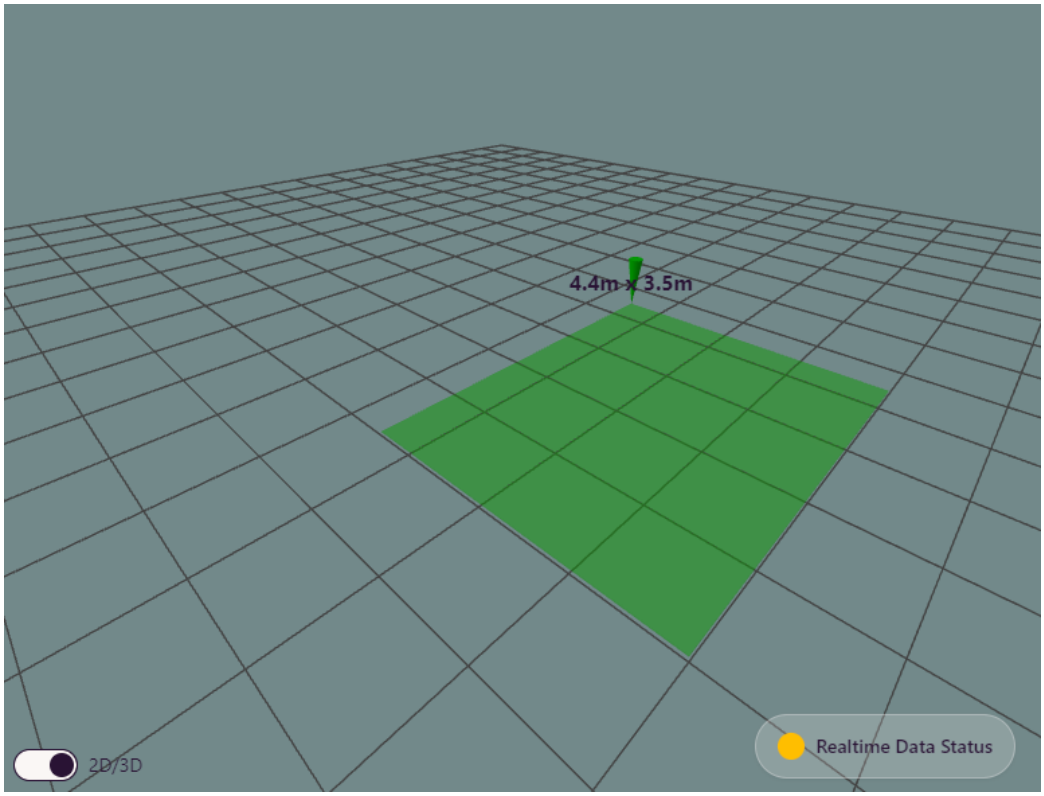

Figure 41. Room creation mode with the green preview shape indicating the location of the new room.

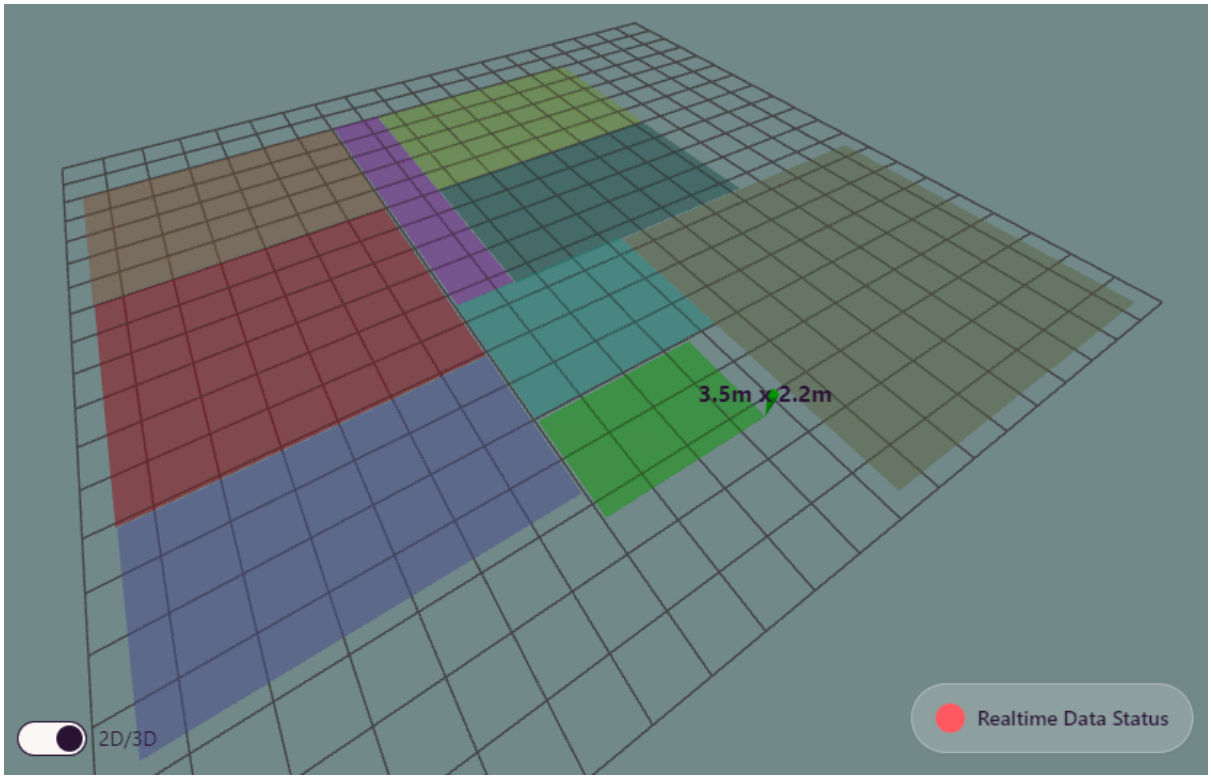

Figure 42. Room layout in the 2D display mode.

Rooms can also be selected and deleted by using the button in the popup window.

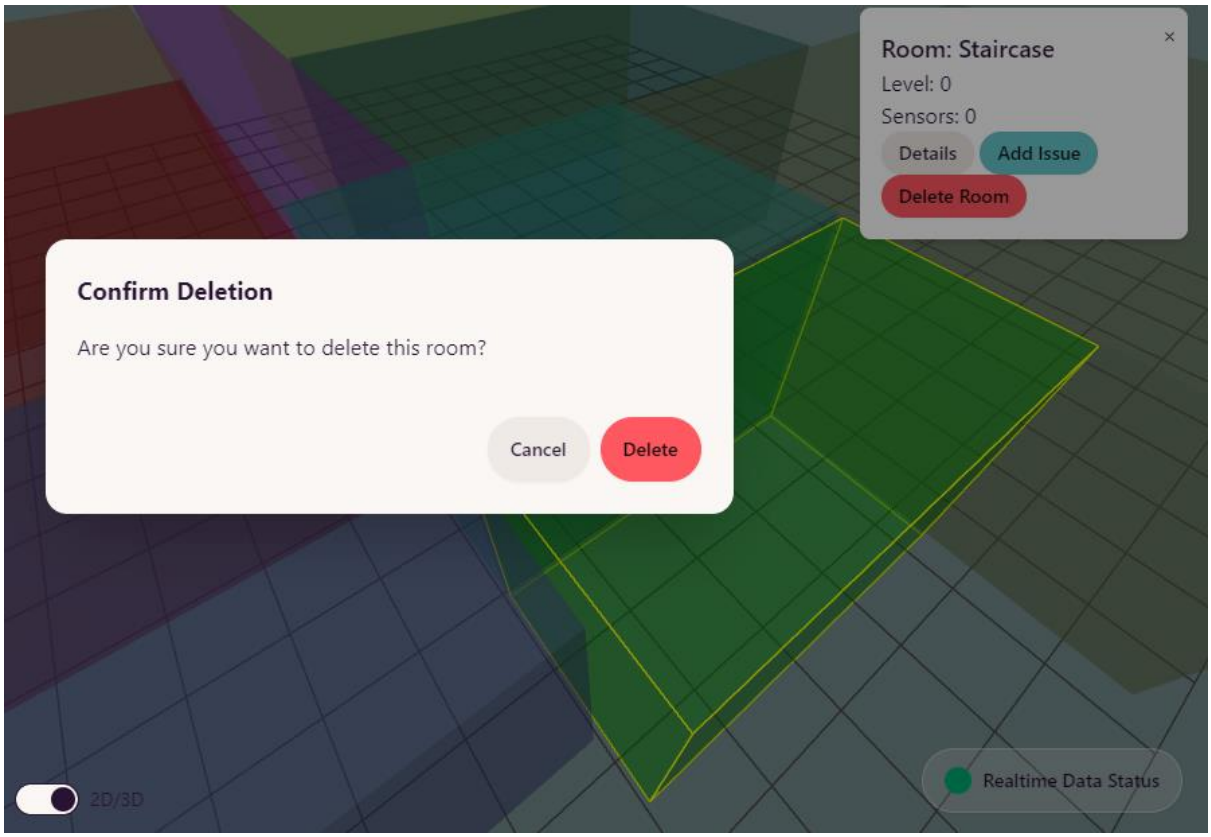

Figure 43. Deleting a Room.

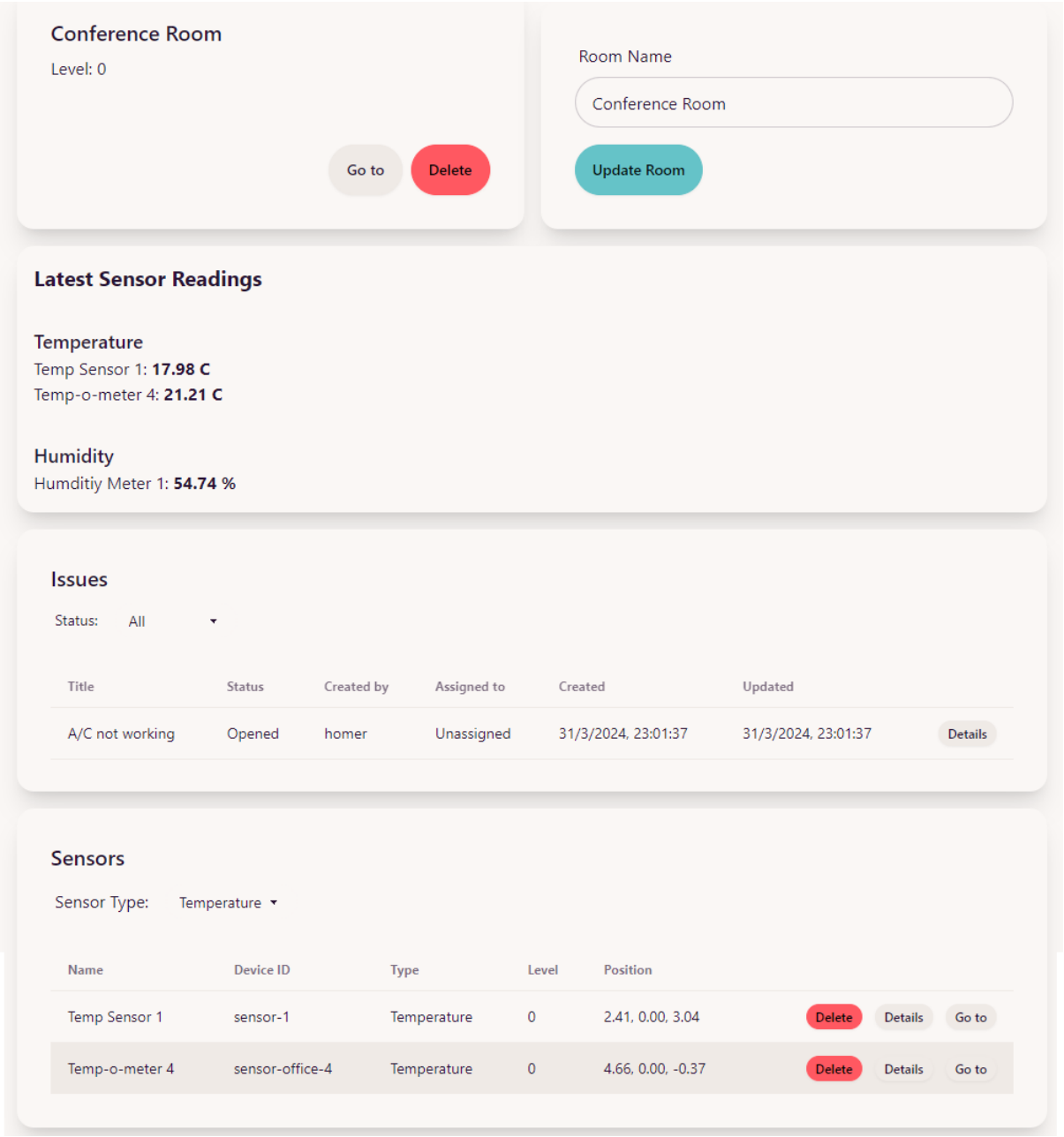

Figure 44. Room details page.

Through the *Details* button, the user can access a page for each room. It enables to view latest sensor readings for the sensors inside, together with the list of issues and sensors inside that room. Also, for each sensor we can click the *Go to* button which takes us directly to the location of the sensor in the Digital Twin Viewer.

5.3.4. Adding and managing Sensors

Sensors can be inserted in the Digital Twin model similarly to rooms. In the form dialog (Figure 45) we need to specify the sensor type and ID. This is used to match the sensor with the data stream.

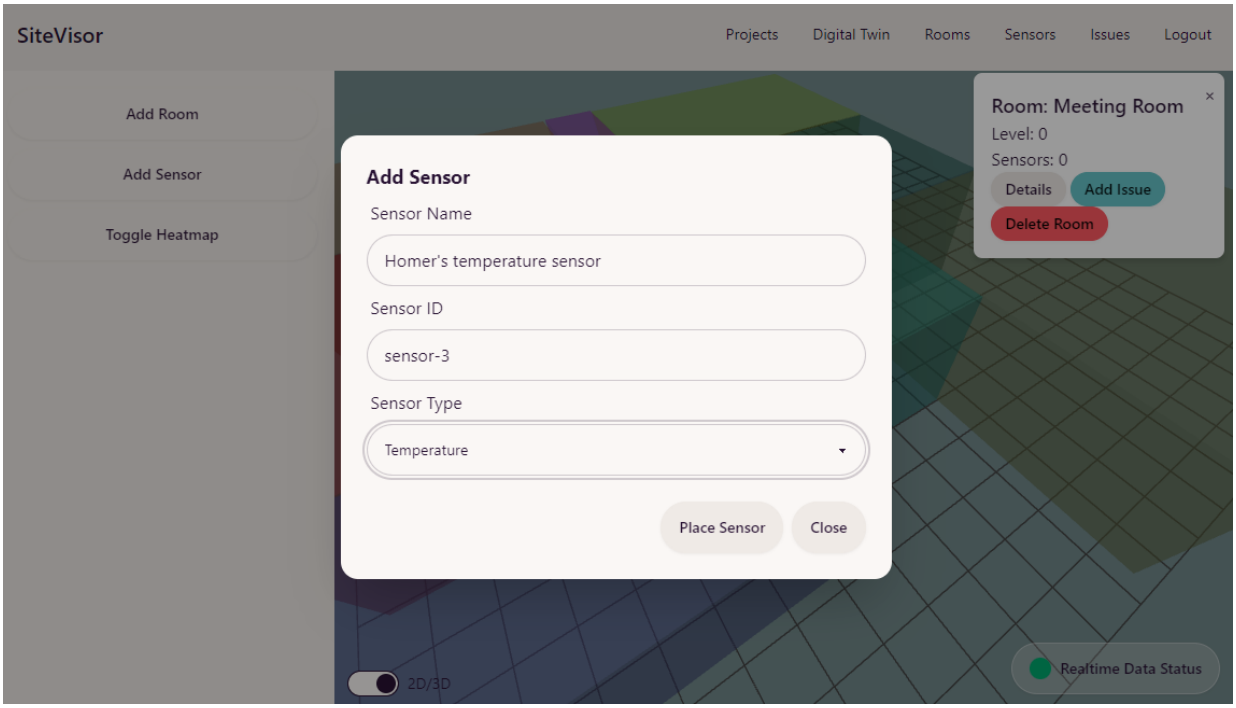

Figure 45. Sensor creation form.

After selecting the sensor, we can view the data plotted in real time (Figure 46). Also, the room is automatically assigned to a room based on the position. It works both ways. When we create a room over existing sensors, they also get correctly assigned together.

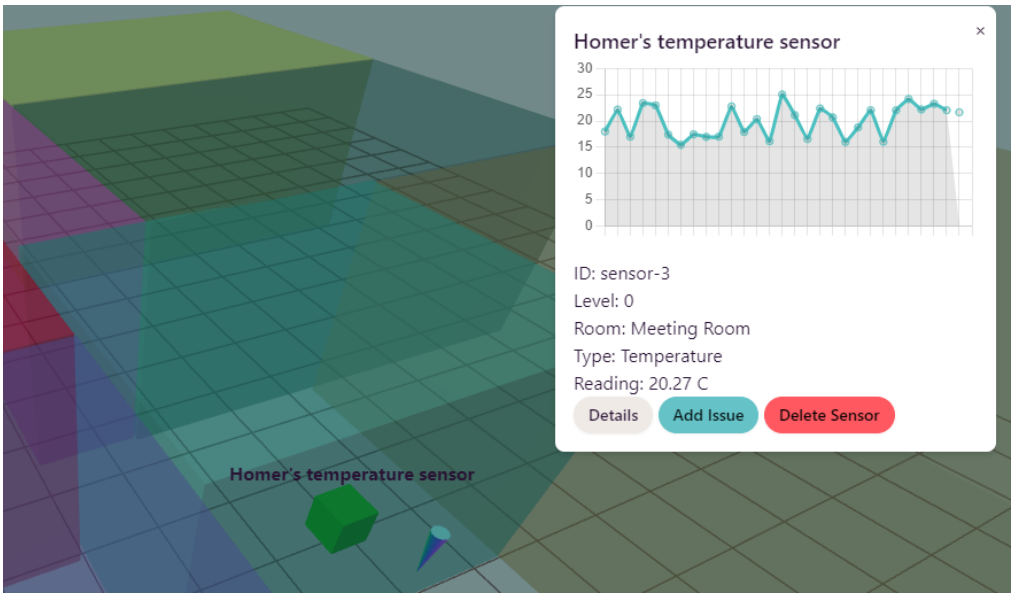

Figure 46. Newly created sensor and the real-time data.

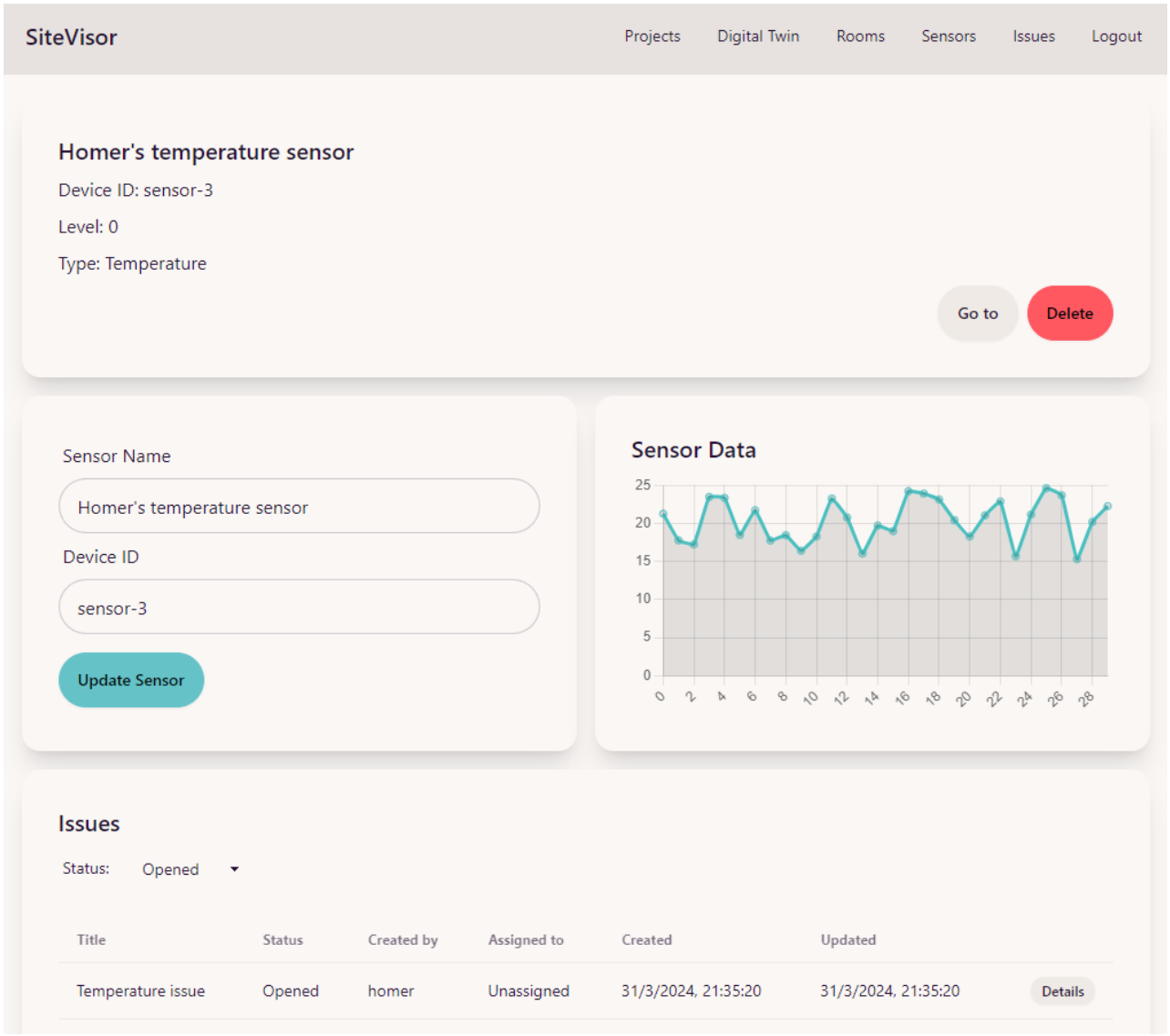

Figure 47. Sensor Details page.

Each sensor also has a details page which lists the basic details about it and allows to update the name and ID. Additionally we can view the real-time data plotted onto a chart and see any existing issues associated with the sensor.

5.3.5. Adding objects to Digital Twin

Adding objects to the Digital Twin needed to work on two levels. First, the user inputs properties of the object (Sensor in the example below) into a standard HTML form. Then the geometry (position) data is supplemented from the 3D scene. The Object Factory object handles creation of the object in the Digital Twin Viewer and returns the complete object (Properties + Position) to be then saved into the database.

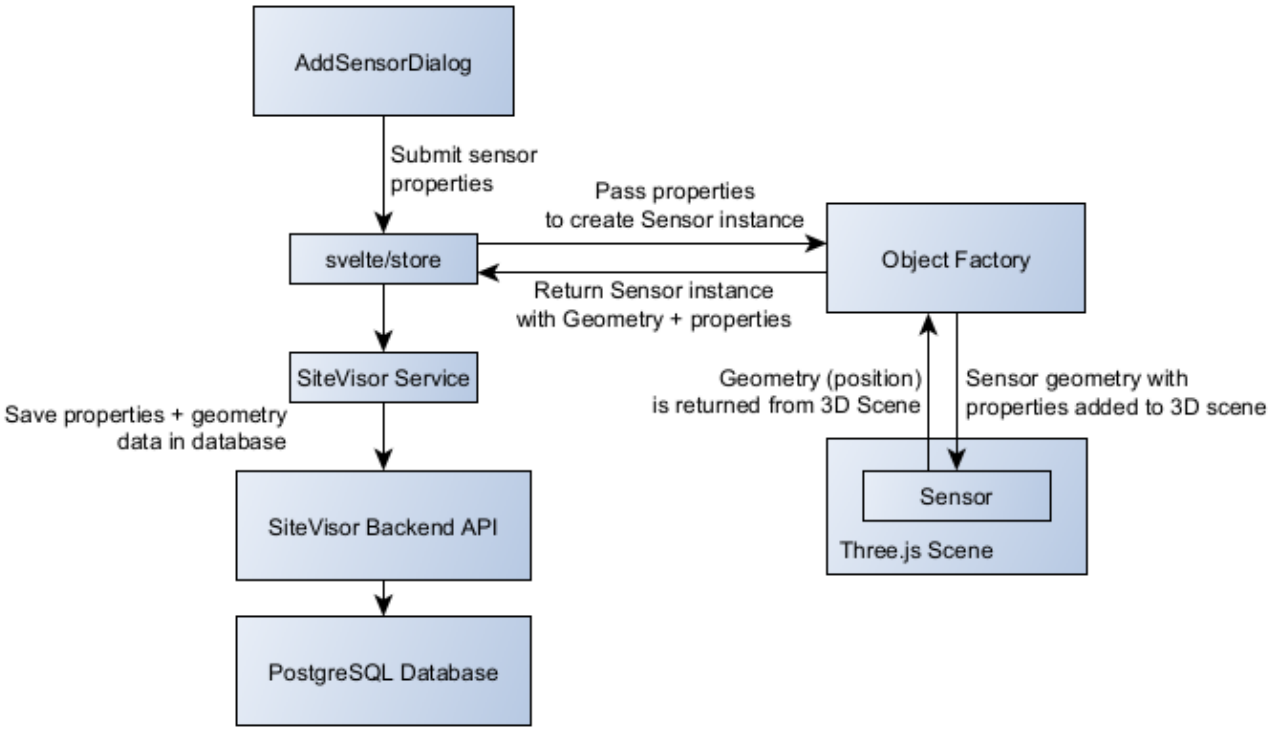

Figure 48. Components used to create an object (Room or Sensor) in the project.

5.3.6. Creating and editing Issues

One of the features planned for the project was asset management through maintenance tasks/issues. In the current implementation users can create *Issues* that are attached either to a *Room* or a *Sensor* object. An Issue can be created either from the Digital Twin view, by selecting an object and clicking the *Add Issue* button (Figure 49), or directly in the Issues listing page.

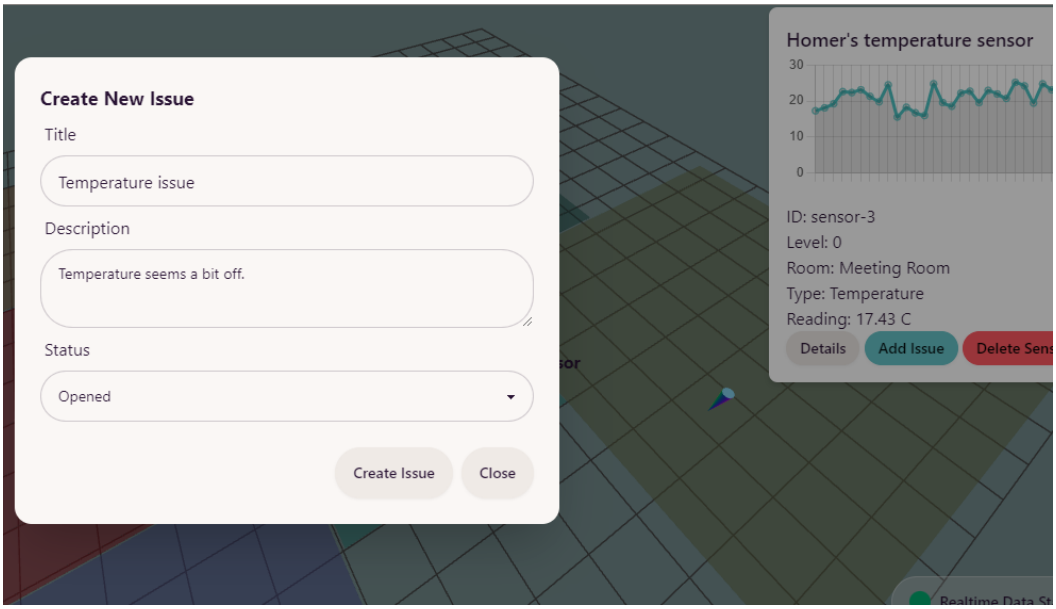

Figure 49. Issue creation form in the 3D Viewer.

Issue creation forms displayed in the Digital Twin view do not have the *Object* and *Object Type* fields, because the object has already been selected in the 3D environment.

When the user creates a new issue from the *Issues* page, as shown in Figure 51, they must pick the *Object Type* first (Sensor or Room) and then they can pick the object by its name from the *Object* dropdown list, which is dynamically populated from the project data.

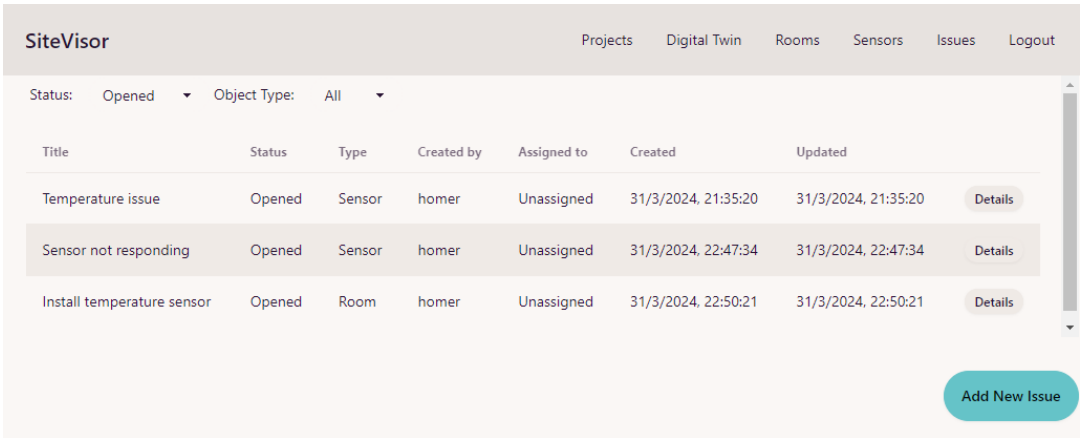

Figure 50. Page listing Issues in the project.

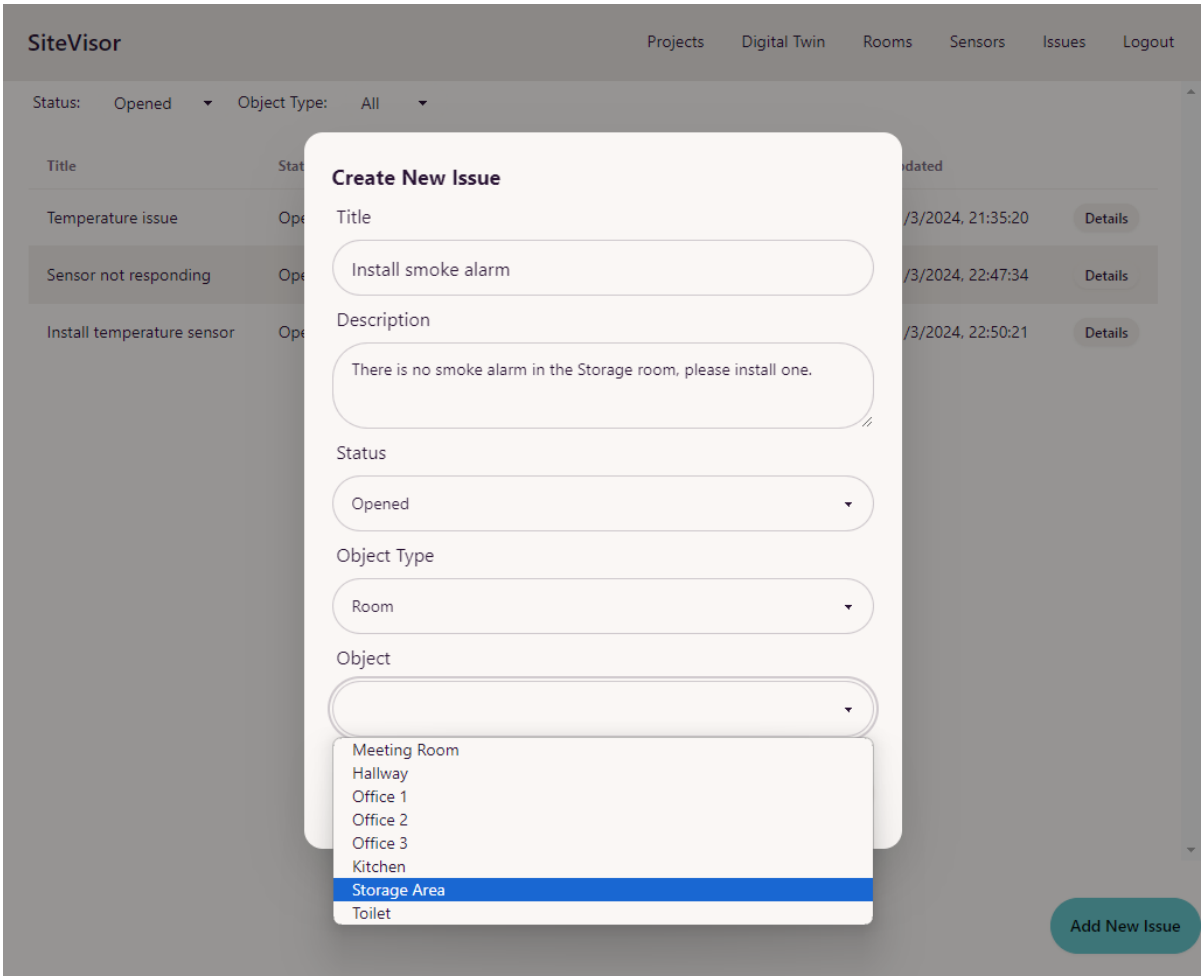

Figure 51. Create New Issue form in Issues listing page.

Issues can have one of the four states:

- Opened – issue created but no work started yet.
- In Progress – work on the issue started.
- Resolved – work on the issue completed.
- Closed – issue not relevant anymore, and not completed for any reason.

A user can be assigned to an issue by providing their username in the *Assigned to* field shown in Figure 53.

Available issue states (*Opened, Resolved* etc.) are globally defined and stored in a svelte store, and can be changed accordingly there, which propagates through the application. This acts as an enumerator for the *Status* field.

Issues details can be viewed (Figure 52) and edited (Figure 53) in the Issue Details page.

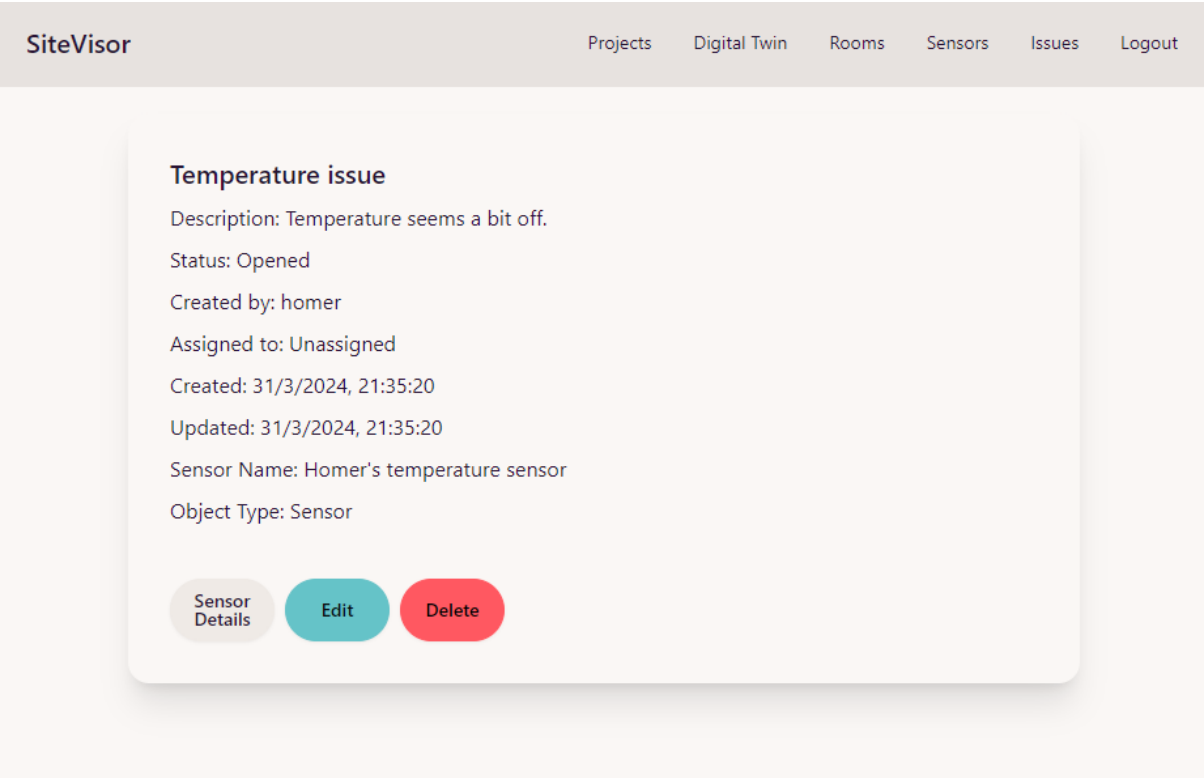

Figure 52. Issue Details page in the viewing mode.

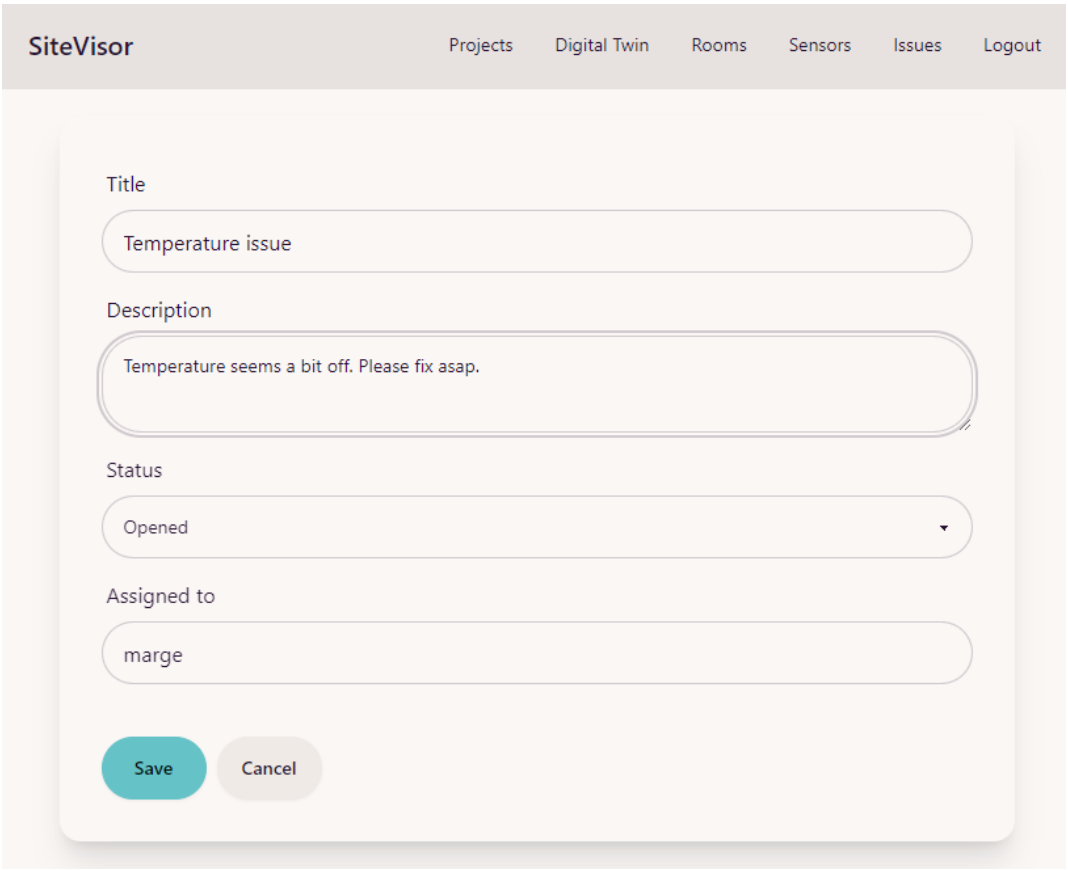

Figure 53. Issue editing form.

5.3.7. Real-time Data Heatmap

The heatmap was one of my key features listed for the Digital Twin/3D Viewer page. The implementation is quite simple and uses Canvas API (Mozilla, 2024) to draw shapes on HTML Canvas element, which then serves as a source of texture for a plane object in the scene.

The initial prototype of the heatmap is shown in the image below. It was done by simply drawing squares of a fixed size with their colour based on the current sensor reading value. This however was not compelling enough as a heatmap and was not providing any additional information from the sensor data.

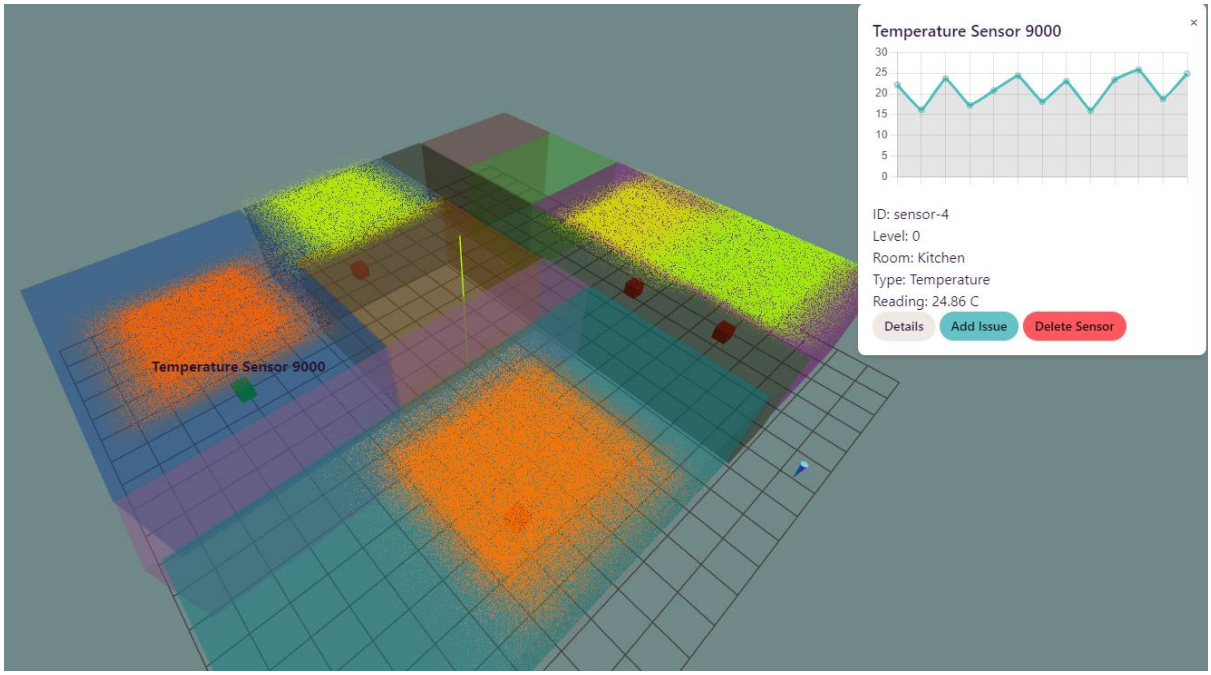

Figure 54. Early experiment with the sensor data heatmap implementation.

In the final implementation, shown in Figure 55, the canvas is divided into a regular grid, with the colour calculated based on weighted values from all sensors of the same type. Each sensor value contributes to the calculated value of each cell in the grid based on the inverse square of the cell – sensor distance. This way I managed to get a full heatmap, which provides additional information about the environment, as it interpolates the values in between sensors.

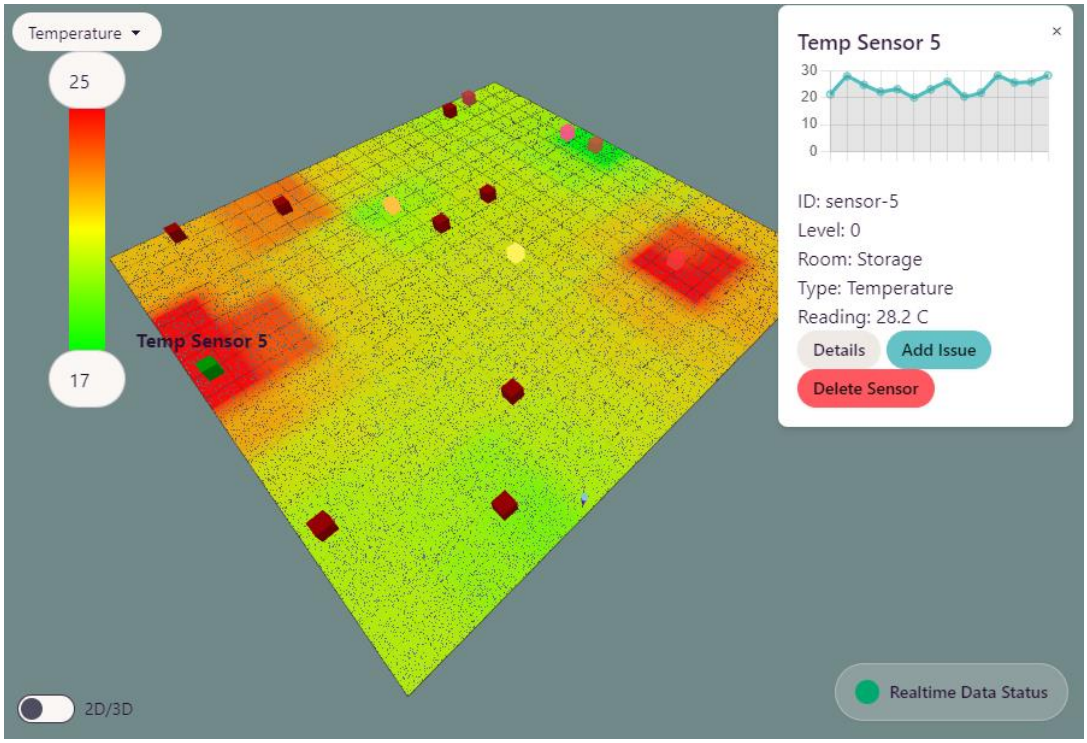

Figure 55. Final implementation of the sensor data heatmap – temperature example.

The user can freely adjust the minimum and maximum reading value at the ends of the gradient scale, which allows for example to identify areas with temperature exceeding a given threshold (red) or under the threshold (green). Sensor type can be selected from the dropdown menu.

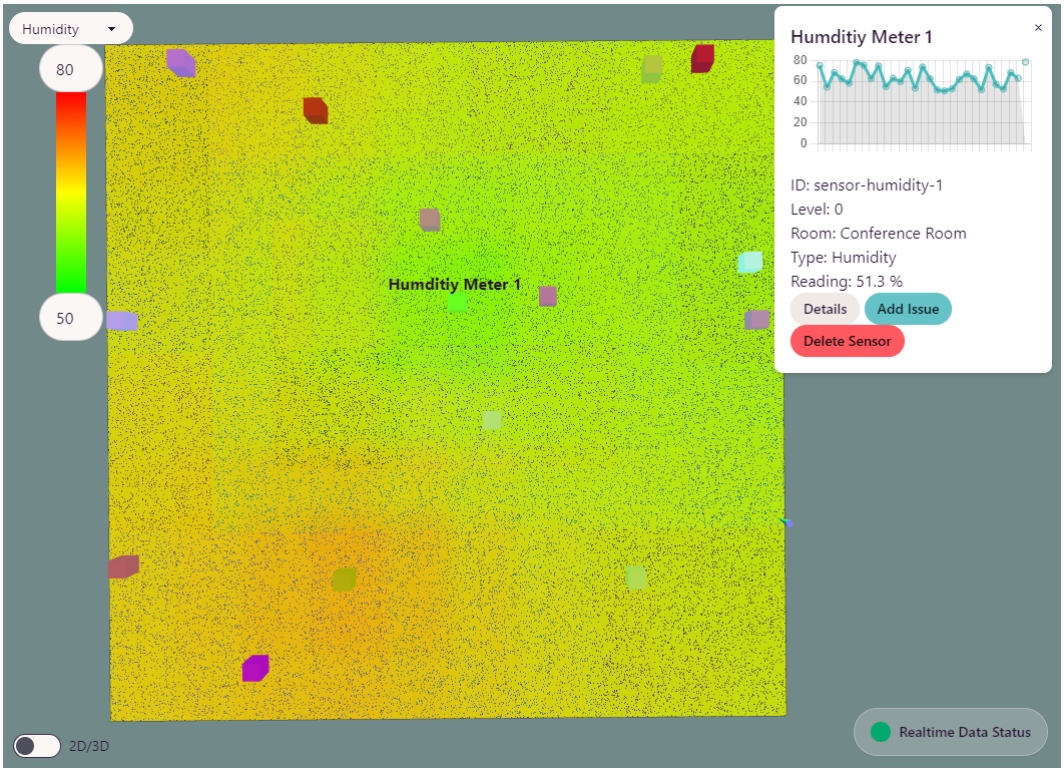

Figure 56. Final implementation of the sensor data heatmap – humidity example.

5.4. Deployment

The project was deployed locally in Kind (Kubernetes in Docker) cluster. The detailed deployment steps are described in the project documentation[1]. The diagram below shows the general flow of the deployment process, from creating the Kind cluster to the application fully deployed.

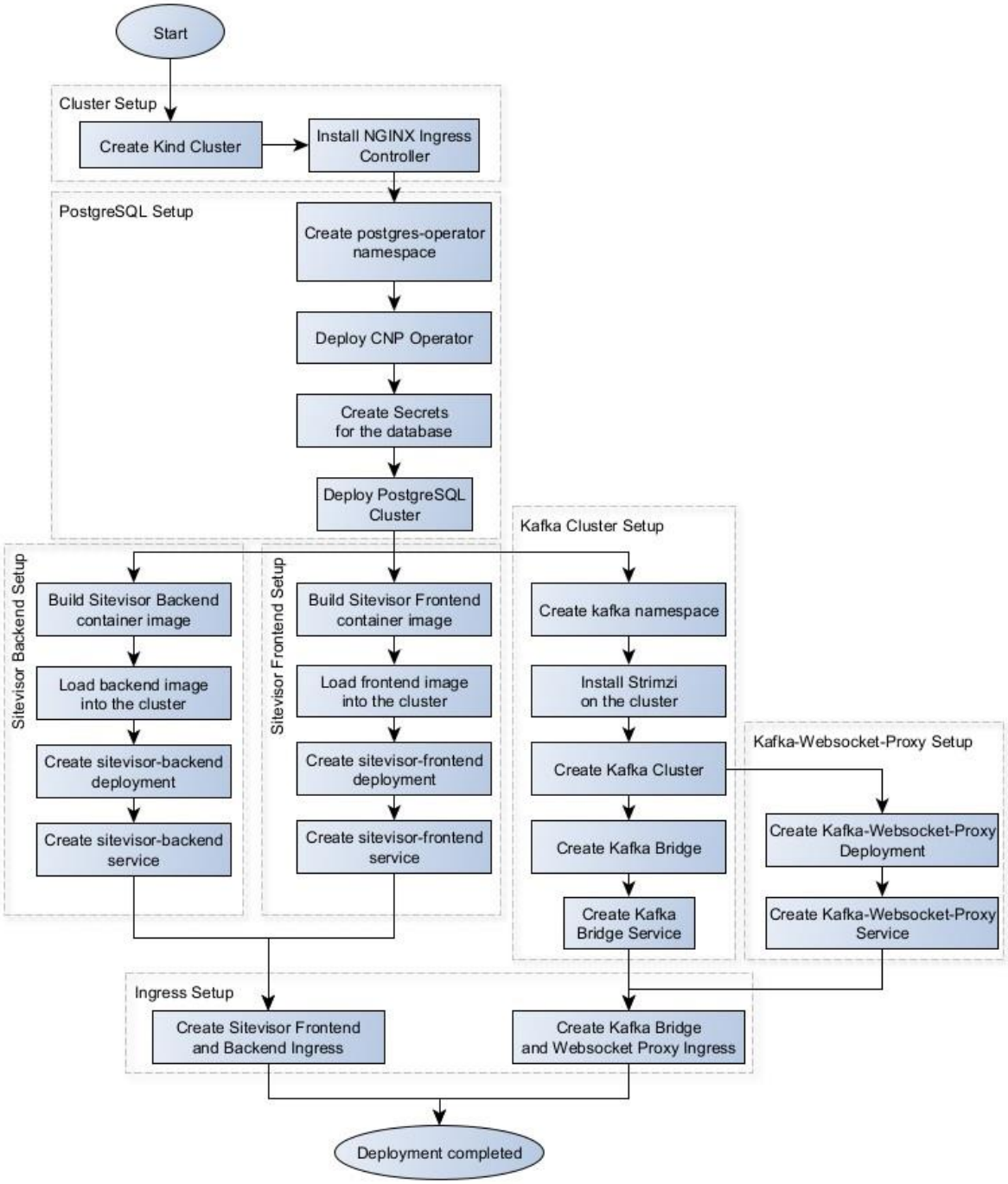

Figure 57. SiteVisor system deployment steps in Kind cluster.

[1] https://grzpiotrowski.github.io/sitevisor/sitevisor-backend/docs/deployment/kind-cluster/

Access to the cluster is managed by NGINX Ingress Controller. The diagram below shows an overview of the Ingress resources created and their relation to the services in the cluster.

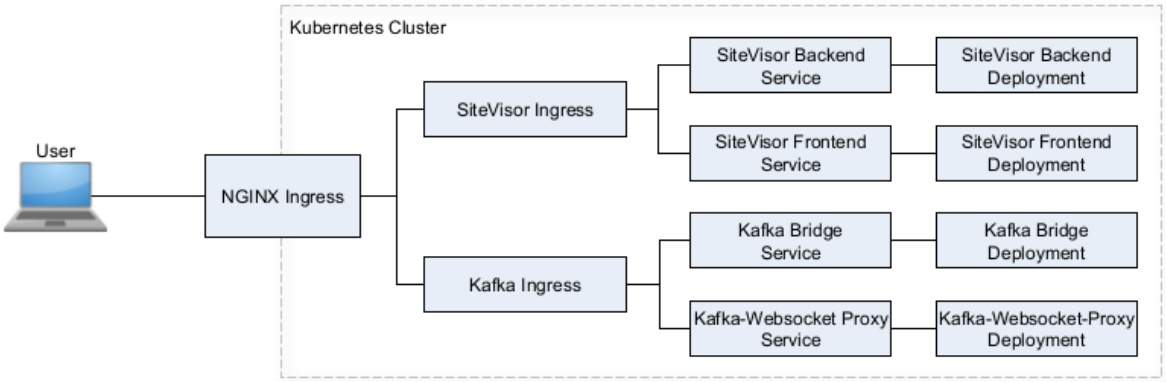

Figure 58. Kubernetes Cluster ingress.

The image below shows all Pods deployed in the cluster after following the described procedure. The tool used to inspect and manage the cluster is called K9s and I can highly recommend it.

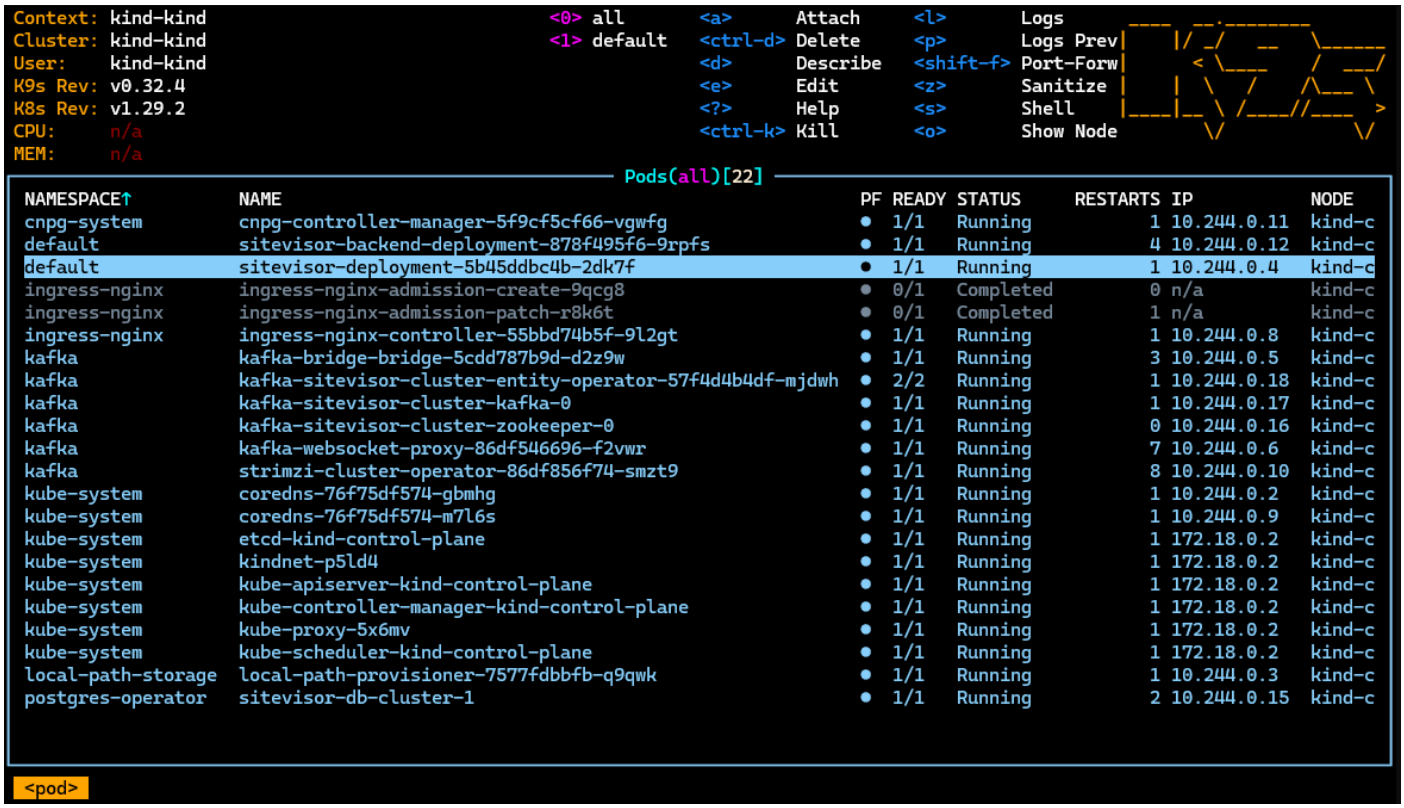

Figure 59. Kubernetes Pods in the cluster as seen in K9s tool.

5.4.1. Container images

Both the frontend and backend applications needed to be containerised to deploy them on Kubernetes. The frontend application was dockerised following an article by (Loic, 2023).

5.5. Documentation

In the spirit of making the SiteVisor project maintainable and approachable by new users and by potential developers willing to join the effort, the project documentation was created along the development and published continuously via a GitHub Action.

Material for MkDocs (Donath, 2024) documentation framework built on top of MkDocs was chosen as a framework for the documentation pages.

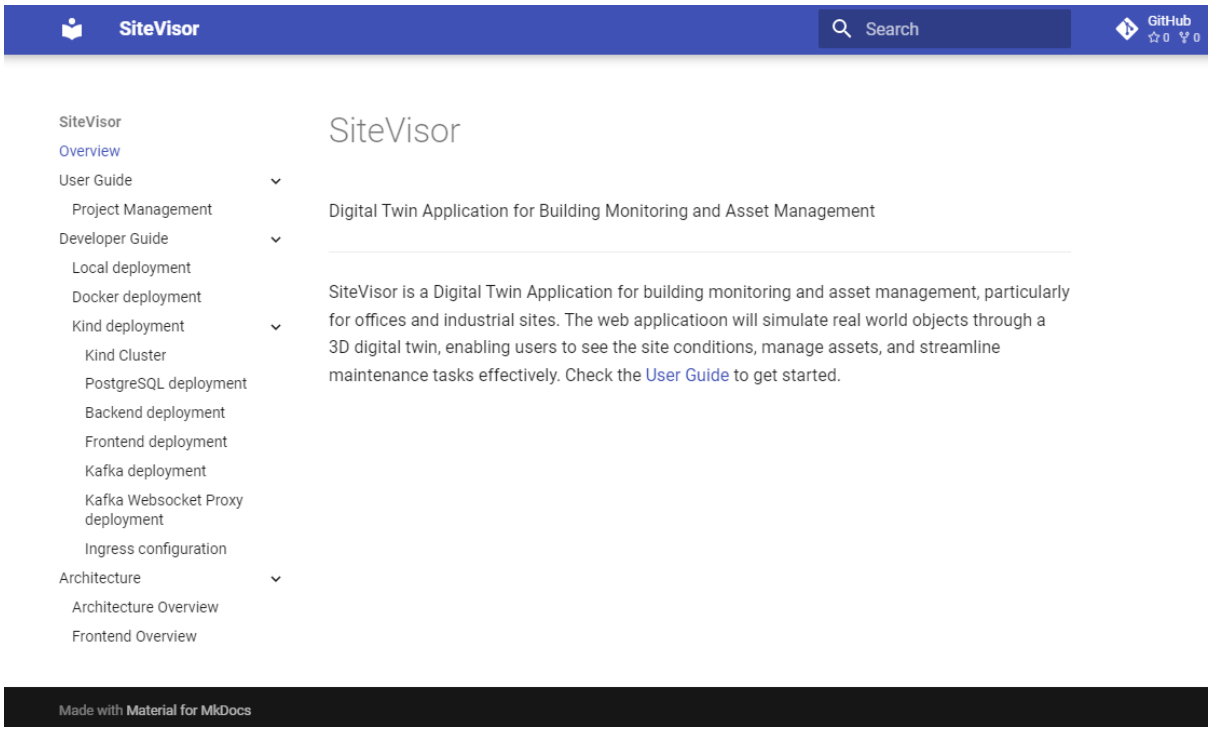

Figure 60. SiteVisor Documentation website hosted on GitHub Pages.

For example. Creation of the cluster and application deployment was fully documented and can be accessed through the documentation pages. Figure 61 shows a fragment of Kafka deployment documentation.

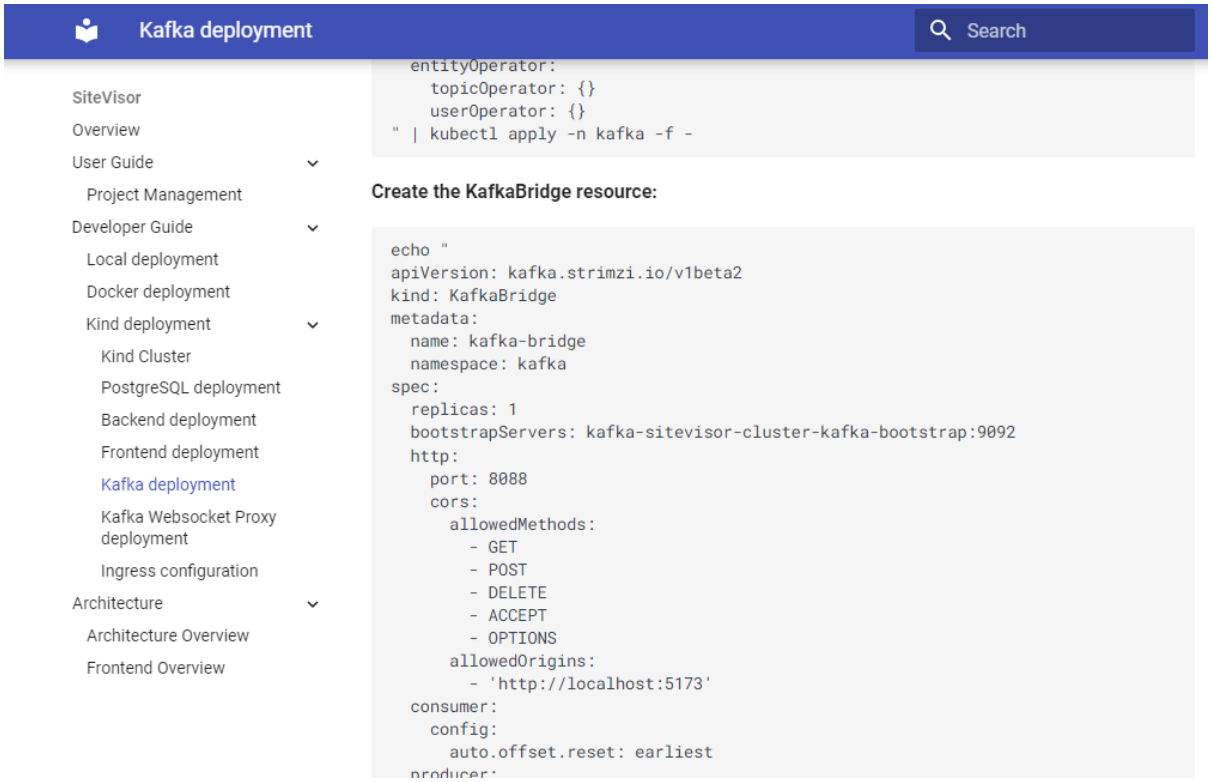

Figure 61. Project documentation page describing Kafka deployment steps.

6. Reflection

6.1. Learnings

The project involved a lot of new learnings on top of the knowledge from the course. To design the architecture and plan the implementation I needed to upskill in several areas. I had a basic knowledge about digital twins, but to realise the vision for the project I had to learn about the topic in much more detail.

During my work placement, I learned about cloud native concepts, containerisation with Docker, and about Kubernetes with the focus on Operator pattern. This time I could look at Kubernetes from a developer perspective. To approach this project, I needed to research and learn Apache Kafka and Strimzi, which were all new to me. I already knew other messaging system – MQTT, which could possibly be set as the backbone of the architecture, but I wanted to learn a new and interesting technology.

I had some knowledge in Three.js prior to starting work on the project as I previously used it in a simple prototype. This time however I needed to expand on that experience and consider many more aspects of this library, due to complexity of the planned features.

Besides the technologies learned, I could greatly appreciate the power of long periods of intense focus when trying to implement new features in the project. Often it was not a straightforward task and required a lot of problem solving and rethinking the approach chosen initially.

Planning – the thing that I almost dreaded because of many unknowns, has become very much engraved in my working process. After completing the project, I think it is crucial to plan ahead, especially when building larger and complex systems.

6.2. Achievements

Most of the goals set for the project were achieved at least to some extent. Some features were implemented in a different form than they were planned, but since this was very much a learning project, I am happy with the results and the whole journey. The table below provides a quick overview of the state of features implementation, when looking back at Section 3 of this report. To summarise, 21 out of 32 planned features were implemented.

Table 1. Summary of feature implementation.

Feature	Implemented
3D visualisation of a building	Yes
Camera controls/navigation	Yes
Object interactivity	Yes
Add rooms, create model from scratch	Yes
Upload 3D model GLTF file	No
Multiple levels in the building	No
Sensors CRUD in 3D Model	Yes
Assets CRUD in 3D Model	No
Issues/Maintenance Tasks CRUD in 3D model	Yes
Assign user to a task/issue	Yes
Data dashboard	No
Charts with real time sensor data	Yes
Charts with archived sensor data	No
Notifications and alerts	No
Simulated sensors	Yes
Heatmap	Yes
User roles	No
Share projects between users	No
Signup/Login	Yes

Feature	Implemented
Project management page	Yes
Projects list page + Create new project	Yes
Historical data page	No
Rooms list page	Yes
Room details page	Yes
Sensors list page	Yes
Sensor details page	Yes
Asset list page	No
Asset details page	No
Issue/Maintenance tasks list page	Yes
Issue/Maintenance tasks creation page	Yes
Issue/Maintenance tasks details page	Yes
Documentation	Yes

Additional statistics about the project across all repositories:

- 71 GitHub Issues were opened.
- 54 Issues were closed/completed.
- 31 merged pull requests.

6.3. Overcoming Challenges

6.3.1. Project architecture and structure

It was quite challenging to design and implement an appropriate architecture for the project. Researching papers and books about Digital Twins was helpful in designing the overall system. The details of implementation however posed quite a challenge, especially when it comes to organising the code and building up all components. SvelteKit brings some structure by design, as it uses the filesystem for routing for example. But then three.js brings a whole new dimension to the application on top of that and linking these two worlds was quite challenging. In search for a sustainable project structure, I reviewed existing projects and templates for mid-sized three.js applications on GitHub and took inspiration from them, but in the end had to develop my own structure.

6.3.2. Django and MongoDB

MongoDB was planned as the database of choice for the project. However, it is not supported by Django natively. This means that a third-party extension needs to be used. This

posed problems with dependencies being out of date and causing errors. I tried Djongo package, but it was difficult to set up the planned model. Eventually decided to switch to PostgreSQL.

In the end, this was a good decision and looking back at the design, it probably makes more sense for the digital twin architecture. The objects and their relations are known from the start as everything must be well structured for the model to work.

6.3.3. Kafka-WebSocket connection

There is no native bridge between Kafka and WebSocket. One solution to use long polling approach, but it is not efficient and not truly real-time. Through searching web I found the Kafka-Websocket-Proxy project and managed to deploy it and weave into the project.

6.3.4. WebSocket connection to Kubernetes

Initially, I wanted to utilise on of the implementation of Kubernetes Gateway API, like NGINX Gateway Fabric, but I could not get the Websocket connection to work with outside of the cluster. For this reason, I decided to switch to NGINX Ingress Controller.

6.4. Future Development

There are many aspects of the application which could be enhanced, or new elements added. The project could not tackle all the different areas of web development at once, and some important features had to be abandoned. However, there is always room for future improvements. Some of which could include:

Focus on Security

Something that was not a key item for this project is safeguarding the system at different levels.

CI/CD Pipeline

Continuous integration and continuous deployment was something that I wanted to add to the project, but due to focus on other tasks, I decided to skip this feature and only the project documentation is continuously deployed on push to the main branch.

Enhanced Asset management

The list of features for the project includes asset management, which was partially implemented through *Issues* for Sensors and Rooms. Assets in the context of this projects were also understood as the building itself, together with its components or any equipment inside it that the user would wish to keep track of. There was an additional object planned to be implemented – a generic Asset, which could represent any physical object mentioned earlier. The specific characteristics important for a piece of equipment for example would be set through a list of custom properties. Unfortunately, due to time constraints, this feature was not implemented as other tasks were prioritised over it.

Rate Limiting

The original proposal for the project listed many things, that in hindsight could likely make for more than one project.

More Digital Twin Features

There are many thing that I could see implemented to enhance the Digital Twin experience, among many:

- Collision detection between rooms to prevent overlapping.
- Levels in the building.
- Dynamically updating objects position based on X, Y, Z sensor data, for example from an accelerometer.

Shared Web Worker for Websockets

Currently the Websocket clients in the application are instantiated in the Svelte store, from where they can be accessed globally. I have researched other options briefly and would like to set up websocket connections in a Web Worker, so they could be made accessible even when the user has multiple tabs with the application opened. Currently the websocket connections would be duplicated in such scenario.

User Roles

Unfortunately, I did not implement user roles as I planned initially. This is something that I would like to see in the application in the future.

Bibliography

Amazon, 2024. *AWS IoT - AWS IoT TwinMaker.* [Online]
Available at: https://aws.amazon.com/iot-twinmaker/
[Accessed 5 February 2024].

Amazon, 2024. *AWS IoT TwinMaker - Features.* [Online]
Available at: https://aws.amazon.com/iot-twinmaker/features/
[Accessed 4 February 2024].

Autodesk, 2022. *Autodesk-Forge - Forge Digital Twin Demo.* [Online]
Available at: https://github.com/Autodesk-Forge/forge-digital-twin
[Accessed 9 February 2024].

CIBSE, 2018. *CIBSE Journal - Sensor sensibility: Newcastle University's Urban Sciences Building.* [Online]
Available at: https://www.cibsejournal.com/technical/ubs/
[Accessed 4 February 2024].

CloudNativePG, 2024. *CloudNativePG - Documentation - Quickstart.* [Online]
Available at: https://cloudnative-pg.io/documentation/1.22/quickstart/
[Accessed 25 March 2024].

Dirksen, J., 2023. *Learn Three.js.* Fourth Edition ed. Birmingham: Packt Publishing.

Donath, M., 2024. *Material for MkDocs - Getting started.* [Online]
Available at: https://squidfunk.github.io/mkdocs-material/getting-started/
[Accessed 8 February 2024].

Fu, G., Zhang, Y. & Yu, G., 2021. A Fair Comparison of Message Queuing Systems. *IEEE Access,* Volume 9, pp. 421-432.

Galbraith, B., 2023. *Bloomfire - Data vs. Information: What's the Difference?.* [Online]
Available at: https://bloomfire.com/blog/data-vs-information/
[Accessed 3 February 2024].

GitHub, 2023. *GitHub Docs: GitHub Issues - Projects - Learning about Projects.* [Online]
Available at: https://docs.github.com/en/issues/planning-and-tracking-with-

projects/learning-about-projects/about-projects
[Accessed 8 February 2023].

Immersive Web, 2024. *Getting started building a WebXR Website.* [Online]
Available at: https://immersiveweb.dev/#three.js
[Accessed 8 February 2024].

Joshi, S. D., 2023. Digital Twin Solution Architecture. In: M. Vohra, ed. *Digital Twin Technology - Fundamentals and Applications.* s.l.:John Wiley & Sons, Scrivener Publishing, pp. 47-76.

kpmeen, 2023. *Kafka WebSocket Proxy.* [Online]
Available at: https://kpmeen.gitlab.io/kafka-websocket-proxy/
[Accessed 3 February 2024].

Loic, J., 2023. *Dockerize SvelteKit with Node.js.* [Online]
Available at: https://medium.com/@loic.joachim/dockerize-sveltekit-with-node-adapter-62c5dc6fc15a
[Accessed 31 March 2024].

Microsoft, 2022. *Azure Documentation - IoT - 3D Scenes Studio (preview) for Azure Digital Twins.* [Online]
Available at: https://learn.microsoft.com/en-us/azure/digital-twins/concepts-3d-scenes-studio
[Accessed 5 February 2024].

Microsoft, 2024. *Azure Documentation - IoT - Azure Digital Twins.* [Online]
Available at: https://learn.microsoft.com/en-us/azure/digital-twins/overview
[Accessed 5 February 2024].

Mozilla, 2024. *MDN Web Docs - Drawing shapes with canvas.* [Online]
Available at: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Drawing_shapes
[Accessed 29 March 2024].

Nölle, C. et al., 2022. *Digital Twin-enabled Application Architecture for the Process Industry.* s.l., s.n.

Parrott, A. & Warshaw, L., 2017. *Industry 4.0 and the digital twin,* s.l.: Deloitte University Press.

Rajamurugu, N. & Karthik, M. K., 2023. Introduction, History and Concept of Digital Twin. In: M. Vohra, ed. *Digital Twin Technology - Fundamentals and Applications.* s.l.:John Wiley & Sons, Scrivener Publishing, pp. 19-32.

Rosso, J., Lander, R., Brand, A. & Harris, J., 2021. *Production Kubernetes - Building Successful Application Platforms.* 1st ed. s.l.:O'Reilly Media.

Savaglio, C. et al., 2023. Edge Intelligence Against COVID-19: A Smart University Campus Case Study. In: F. Cicirelli, G. Spezzano, A. Guerrieri & A. Vinci, eds. *IoT Edge Solutions for Cognitive Buildings.* s.l.:Springer, pp. 221-228.

Scott, D., Gamov, V. & Klein, D., 2022. *Kafka in Acion.* NY: Manning Publications Co..

Strimzi, 2024. *Strimzi Documentation.* [Online]
Available at: https://strimzi.io/docs/operators/latest/overview
[Accessed 28 03 2024].

Three.js, 2024. *Three.js Manual - Fundamentals.* [Online]
Available at: https://threejs.org/manual/#en/fundamentals
[Accessed 10 February 2024].

UKCRIC, 2023. *Newcastle Urban Observatory - Urban Sciences Building.* [Online]
Available at: https://newcastle.urbanobservatory.ac.uk/dash/usb/
[Accessed 4 February 2024].

Appendix A – Export from GitHub Project Roadmap (formatted)

Iteration 1		
07/01/2024	Create a starter Three.js scene with basic interactivity	https://github.com/grzpiotrowski/sitevisor/issues/2
09/01/2024	Implement adding objects to the scene	https://github.com/grzpiotrowski/sitevisor/issues/5
Iteration 2		
16/01/2024	Create developer documentation	https://github.com/grzpiotrowski/sitevisor/issues/10
16/01/2024	Create user documentation	https://github.com/grzpiotrowski/sitevisor/issues/9
18/01/2024	Dockerise the application	https://github.com/grzpiotrowski/sitevisor/issues/17
19/01/2024	App deployment on kubernetes	https://github.com/grzpiotrowski/sitevisor/issues/18
22/01/2024	Establish a sustainable project structure	https://github.com/grzpiotrowski/sitevisor/issues/1
Iteration 3		
28/01/2024	Interim Report	https://github.com/grzpiotrowski/sitevisor-project/issues/1
11/02/2024	Showcase Entry Part 1	https://github.com/grzpiotrowski/sitevisor-project/issues/2
Iteration 4		
13/02/2024	SiteVisor backend API	https://github.com/grzpiotrowski/sitevisor-backend/issues/11
14/02/2024	Implement exporting the scene to the backend	https://github.com/grzpiotrowski/sitevisor/issues/3
14/02/2024	Build API for database CRUD	https://github.com/grzpiotrowski/sitevisor-backend/issues/5
15/02/2024	Include documentation from sitevisor-backend	https://github.com/grzpiotrowski/sitevisor/issues/22
15/02/2024	SiteVisor Service	https://github.com/grzpiotrowski/sitevisor/issues/37
15/02/2024	Implement database for scene objects	https://github.com/grzpiotrowski/sitevisor-backend/issues/6
16/02/2024	Implement importing the objects from the backend	https://github.com/grzpiotrowski/sitevisor/issues/4
Iteration 5		
29/02/2024	Add user accounts and projects	https://github.com/grzpiotrowski/sitevisor/issues/6
03/03/2024	Enhance the UI with a CSS framework	https://github.com/grzpiotrowski/sitevisor/issues/11
03/03/2024	Rooms API	https://github.com/grzpiotrowski/sitevisor-backend/issues/10
03/03/2024	Users API	https://github.com/grzpiotrowski/sitevisor-backend/issues/12
03/03/2024	Projects API	https://github.com/grzpiotrowski/sitevisor-backend/issues/13
05/03/2024	Singup/login pages	https://github.com/grzpiotrowski/sitevisor/issues/24
05/03/2024	Projects list page	https://github.com/grzpiotrowski/sitevisor/issues/25
05/03/2024	3D Viewer/Digital Twin page	https://github.com/grzpiotrowski/sitevisor/issues/36
05/03/2024	Sensors API	https://github.com/grzpiotrowski/sitevisor-backend/issues/9
05/03/2024	SiteVisor Frontend Views	https://github.com/grzpiotrowski/sitevisor/issues/41
Iteration 6		
08/03/2024	Add tests	https://github.com/grzpiotrowski/sitevisor-backend/issues/16
08/03/2024	Fix: Helper Grid middle line colour	https://github.com/grzpiotrowski/sitevisor/issues/45
09/03/2024	Object creation dialog	https://github.com/grzpiotrowski/sitevisor/issues/49
10/03/2024	Create Room by dimensions	https://github.com/grzpiotrowski/sitevisor/issues/46
13/03/2024	Implement websocket client	https://github.com/grzpiotrowski/sitevisor/issues/21

Iteration 7		
17/03/2024	Add Websocket server	https://github.com/grzpiotrowski/sitevisor-backend/issues/2
19/03/2024	Implement sensor object interactivity	https://github.com/grzpiotrowski/sitevisor/issues/16
19/03/2024	Projects manager page	https://github.com/grzpiotrowski/sitevisor/issues/26
19/03/2024	Create a simple simulated sensor for testing	https://github.com/grzpiotrowski/sitevisor-sensors/issues/1
20/03/2024	Sensor list page	https://github.com/grzpiotrowski/sitevisor/issues/33
20/03/2024	Sensor details page	https://github.com/grzpiotrowski/sitevisor/issues/34
20/03/2024	Data chart for sensor	https://github.com/grzpiotrowski/sitevisor/issues/55
Iteration 8		
24/03/2024	Showcase Entry Part 2	https://github.com/grzpiotrowski/sitevisor-project/issues/3
24/03/2024	Room interactivity	https://github.com/grzpiotrowski/sitevisor/issues/58
25/03/2024	Maintenance tasks list page	https://github.com/grzpiotrowski/sitevisor/issues/30
25/03/2024	Maintenance task creation page	https://github.com/grzpiotrowski/sitevisor/issues/31
25/03/2024	Maintenance task details page	https://github.com/grzpiotrowski/sitevisor/issues/32
25/03/2024	Maintenence tasks API	https://github.com/grzpiotrowski/sitevisor-backend/issues/8
27/03/2024	Room data page	https://github.com/grzpiotrowski/sitevisor/issues/35
27/03/2024	Room - Sensor relation	https://github.com/grzpiotrowski/sitevisor-backend/issues/25
Iteration 9		
28/03/2024	Selecting a room after creating new sensor fails	https://github.com/grzpiotrowski/sitevisor/issues/65
29/03/2024	Heatmap from Sensor data	https://github.com/grzpiotrowski/sitevisor/issues/54
29/03/2024	Fix: Sensor not assigned to a room when created in 2D mode	https://github.com/grzpiotrowski/sitevisor/issues/67
30/03/2024	Fix: Warning: Too many active WebGL contexts	https://github.com/grzpiotrowski/sitevisor/issues/53
31/03/2024	Add Asset management functionality	https://github.com/grzpiotrowski/sitevisor/issues/12
02/04/2024	Physical sensor client	https://github.com/grzpiotrowski/sitevisor-sensors/issues/3
02/04/2024	Final Project Submission (code)	https://github.com/grzpiotrowski/sitevisor-project/issues/4
04/04/2024	Final Project Submission (final report)	https://github.com/grzpiotrowski/sitevisor-project/issues/5

Appendix B – Ethical Approval Checklist

Included on the next page.

This Ethics Checklist must be completed for all final year undergraduate, taught postgraduate and research projects in the School of Science and Computing.

View your response(s)

Respondent: **Grzegorz Piotrowski** (Group: CM-HDIPCS) Submitted on: Sunday, 19 November 2023, 5:11 PM

Ethics Checklist for Undergraduate, Taught Postgraduate and Research Projects in the School of Science and Computing

All students in the School of Science and Computing who are either (1) in the final year of an undergraduate/BSc degree, or (2) on a taught postgraduate/MSc programme **must complete this Ethics Checklist before conducting their project** regardless of the project type or discipline. The Checklist should also be completed by anyone (whether staff member or student) conducting a **research project** (whether programmatic or not) within the School.

The purpose of this Ethics Checklist is to **identify projects that will require formal ethical approval** from the School Research Ethics Committee, or the SETU Research Ethics Committee, before they can proceed.

Students/applicants should note that this Ethics Checklist is a **formal declaration**, and great care must be taken to **answer all questions accurately**. Students should consult with their project supervisors/advisors regarding any aspects or questions that they are unsure of before completing and submitting the Ethics Checklist.

Students/applicants must **answer all questions** presented to them until the Checklist questionnaire is completed.

Feedback Report

No human experimentation issues (UG).

No animal experimentation issues (N/A).

No issues regarding the use of human tissues.

No animal tissue or biological fluids issues.

No ionising radiation issues.

No primary data collection issues (N/A).

No underage/vulnerable people issues (UG).

?

No issues regarding existing/secondary data use (N/A).

No controversial data issues.

No issues related to the collection of rare or protected plants.

No issues regarding the use of genetically modified (GM) plant material.

Instructions:

1. If the above feedback is **entirely green** then, based on your answers, there is **no need to apply for ethical approval** for your project.
2. If **any** part of the above feedback is **yellow/amber**, then there is at least one issue with your project that needs to be reviewed and **you must apply for ethical approval** to continue your project.
3. If **any** part of the above feedback is **red** then there is a serious ethical issue and **you cannot continue your project** as currently planned.

It is recommended that you print this Feedback Report to a PDF file for your records. You should also forward and discuss this Feedback Report PDF with your project supervisor. They will be able to advise if you have any further questions or if you need to apply for ethical approval.

1 * Are you a student on a **final year undergraduate** programme, a **taught postgraduate** programme, or are you conducting a **research project**?

- (•) Final Year Undergraduate
- ( ) Taught Postgraduate
- ( ) Postgraduate Research Project
- ( ) Other Research Project

2 * What is the **working title** of your project?

SiteVisor - Digital Twin Application for Building Monitoring and Asset Management

3 * Who are the project **supervisors/advisors/principal investigators**?

Colm Dunphy / TBC

4 * Does your project involve **human experimentation**?

( ) Yes (•) No

5 * Does your project involve **live animal experimentation**?

( ) Yes (•) No

(6) * Is the planned animal experimentation limited to **non-invasive procedures only** (such as feeding, weighing, or taking naturally voided faecal or hair samples), and does **not** involve any invasive procedures (such as taking rectal faecal samples or blood) from live animals?

( ) Yes ( ) No

7 * Does your project involve the use of **human** remains/cadavers/tissues/cells/biological fluids/embryos/foetuses?

( ) Yes (•) No

(8) * Do you intend to only use established **commercial human cell lines**, and no other **human** remains/cadavers/tissues/cells/biological

fluids/embryos/foetuses in your project?

○ Yes ○ No

9 * Does your project involve the use of **animal cells, tissues** or **biological fluids**?

○ Yes ◉ No

(10) * Do you intend to only use (1) **established commercial animal cell lines**, or (2) **slaughterhouse-derived tissues/fluids**, or (3) **fluids collected as part of routine animal husbandry** (e.g. milk) and no other animal tissues or biological fluids in your project?

○ Yes ○ No

11 * Does your project involve the **collection of rare or protected plants**?

○ Yes ◉ No

12 * Does your project involve the generation or use of **genetically modified (GM) plant material**?

○ Yes ◉ No

(13) * Do you agree to (1) only use **established genetically modified (GM) plant cell lines, seeds, or plant products** in your project, (2) **not generate new plant mutations** using chemical or other means, and (3) follow specified SETU **containment and use protocols** for GM plant materials at all times?

○ Yes ○ No

14 * Does your project involve the use of **ionising radiation**? (e.g. use of gamma ray spectrometry)

○ Yes ◉ No

(15) * Do you agree to carefully **follow the instructions** of the SETU designated **Radiation Protection Officer (RPO)**, and **adhere to all legal requirements** as set out in the Radiological Protection Act 1991 (Ionising Radiation) Regulations (2019), regarding the use of ionising radiation materials and equipment?

○ Yes ○ No

16 * Does your project involve the **collection of any new (or primary) data** from **individual people or groups**?

○ Yes ◉ No

(17) * Does your project involve the **collection of any new (or primary) individual or group data** that is **personally or uniquely identifying**? (e.g. data about people or organisations/companies/groups that could be used to identify those individuals or groups; data collection might take any form, including internet and social media data, etc.)

○ Yes ○ No

(18) * Will you ensure that participants who you are collecting data from are provided with **fair warning** and must provide **explicit informed consent** for any data collected?

○ Yes ○ No

(19) * Will you ensure that any project-related data collection, data storage, and data use is in **full compliance** with the **EU General Data Protection Regulation (GDPR)** and the **Data Protection Act (2018)**?

○ Yes ○ No

(20) * Does any of the data that you intend to collect include **sensitive or private personal information** about individuals, or **commercially sensitive information** about organisations/companies/groups?

○ Yes ○ No

21 * Does your project involve **persons under the age of 18 years** (i.e. minors), or **any vulnerable groups**? (e.g. prisoners, refugees, those in care, addiction service users, etc.)

○ Yes ◉ No

22 * Does your project involve the use of **existing (or secondary) human data**? (i.e. data originally collected for another purpose)

○ Yes ◉ No

(23) * Is the existing or secondary human data you intend to use either (1) **anonymous/non-personally identifying** and in the **public domain**, or (2) available with **explicit and specific informed consent or permission** for the data to be **legally** reused in the way you intend?

○ Yes ○ No

(24) * Are any aspects of the primary/secondary data you intend to use for the project **controversial** in nature?

○ Yes ○ No

25 * Before you submit the Ethics Checklist, you must **confirm all of the following**:

- ☑ I understand that the Ethics Checklist is a formal declaration.
- ☑ I have answered all questions on the Ethics Checklist carefully and truthfully.
- ☑ The supervisor/advisor (or principal investigator) for the project is present as the Ethics Checklist is being submitted, or they have given me explicit permission to submit it in their absence.
- ☑ I have had adequate ethics training and/or instruction prior to completing the Ethics Checklist.
- ☑ I understand, and agree to abide by, the general ethical principle of "do no harm" for this project.
- ☑ I will follow the instructions given in the Feedback Report.

26 * Authentication Code (ask your project supervisor/advisor for this code)

Enter Student Number:

Enter the Authentication Code below and click "Verify Code" Verify Code

Note: If an INVALID authentication code is used then this submission is NULL and VOID

5978