NHAN DOAN

DEVELOPMENT OF TOOLING FOR SOFTWARE DEVELOPMENT TEAMS

Bachelor's thesis

Degree Programme in Information Technology

2023

South-Eastern Finland
University of Applied Sciences

South-Eastern Finland University of Applied Sciences

Author(s)	Nhan Doan
Degree title	Bachelor of Engineering
Time	May 2023
Thesis title	Development of Tooling for software development teams 22 pages
Commissioned by	Observis Oy
Supervisor(s)	Reijo Vuohelainen

ABSTRACT

This bachelor's thesis aimed to deliver a functional and efficient visual studio editor that could be used to configure a software product's user interface and improve productivity. The outcome of this thesis work is a visual editor that is used to configure SVG-based components, allow adjusting SVG configuration properties, and be compatible with the established proprietary SVG configuration syntax. To achieve this, modern front-end development with JS frameworks, UI/UX design, and the benefits of Lerna and monorepo were studied.

Keywords: TypeScript, Electron, React, Lerna, monorepo

South-Eastern Finland University of Applied Sciences

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank Reijo Vuohelainen, my supervisor, for his support and guidance throughout the thesis process.

I would like also to extend my sincere thanks to my colleagues at Observis Oy, who have shared their knowledge and provided feedback. Their expertise has helped me better understand our company and the market we operate in.

Finally, a special mention goes to my workplace mentor, Ville Kanerva, and Bogdan Moroz, for their patient guidance and assistance during the thesis journey.

CONTENTS

1 INTRODUCTION

Observis Oy (OOY) is a company that aims to provide a world-leading situational awareness system (SAS) to international chemical, biological, radiological, nuclear, and high-yield explosive (CBRNE) monitoring and preparedness markets. The SAS system must be highly configurable to quickly adapt to different use cases and support extensive configuration, including the user interface. Currently, the SAS desktop client's user interface is largely configured manually, which is a time-consuming and error-prone process. To simplify the customization of SAS software, Observis is seeking a way to provide a visual editor that can be used to configure SVG-based components used in the SAS user interface.

2 RESEARCH QUESTIONS AND OBJECTIVES

2.1 Research question

How can a visual editor be developed to configure SVG-based components in the ObSAS user interface?

How can modern frontend development technologies such as React, TypeScript, and Electron be used to develop the visual editor?

What are the benefits of using a monorepo development approach with Lerna for managing large code bases?

2.2 Objectives and significance

The objectives of this thesis project are to:

- Develop a visual editor for configuring SVG-based components used in the ObSAS user interface, which will simplify the customization process and reduce the risk of errors.

- Analyze and evaluate the use of modern frontend technologies such as React, TypeScript, and Electron to implement the visual editor and optimize the software configuration process at Observis.
- Explore the benefits of using a monorepo development approach with Lerna to manage and maintain multiple projects and components in a single codebase.

The significance of this thesis project lies in its potential to enhance the efficiency and productivity of software development at Observis by providing an intuitive and user-friendly tool for configuring SVG-based components. This tool will allow developers to quickly adapt the ObSAS system to different use cases, reducing the need for manual configuration and minimizing the risk of errors.

3 INTERNAL TOOLING

3.1 Importance of internal tooling for development productivity and work efficiency

According to Joe Johnston (2022), Internal tools have increased productivity across the board for most industries and can play a huge role in helping an organization cut costs whilst improving output. Internal tooling is an essential part of any software development organization, as it provides the necessary infrastructure and support for the development and management of software systems.

The development of such tooling enables the software development teams at the company to work more efficiently and productively, as it automates many of the manual processes that were previously required for the development. This reduces the likelihood of errors, increases the speed of development, and enables developers to focus on more complex and value-adding tasks.

By providing a reliable and efficient tooling system, the company can ensure that its employees have the necessary infrastructure to develop high-quality software

systems, which can be delivered to customers in a timely and cost-effective manner. Furthermore, a powerful tooling system enables the company to maintain high levels of software quality, ensuring that their software systems are reliable, and meet the needs of their customers. Therefore, investing in good internal tooling is important for developers and organizations.

3.2 Differences between internal customers and external customers

Developing software for internal customers presents unique challenges compared to developing software for external customers. One key difference is the level of familiarity with the product and its development process. *(Sebastian Duncan, 2023.)*

Internal customers have direct access to the developer and are likely more familiar with the product and its development process. This can be both an advantage and a disadvantage for the developer. On the one hand, internal customers have a greater understanding of the product and its requirements and can provide more detailed and specific feedback and support. This can help the developer to refine the product and ensure that it meets the needs of the end users. When building your tools, you have the power to build them exactly how you want them. *(Joe Johnston, 2022.)*

On the other hand, internal customers may have higher expectations and demands, as they are more familiar with the product and its capabilities. This can place additional pressure on the developer to deliver a high-quality product that meets the expectations of internal customers. *(Elizabeth Kampf, 2020.)*

The development process for internal and external customers requires a different approach, which can be both beneficial and challenging. By understanding the unique challenges and requirements of internal customers, a developer can create a visual editor that meets the needs of the internal stakeholders.

4 ANALYSIS OF THE CHOSEN TECHNOLOGIES

4.1 Modern frontend development with JS frameworks

Modern frontend development has evolved significantly in recent years, with the introduction of new tools and frameworks that streamline the development process and enable developers to build complex and interactive user interfaces *(Eduardo Pineda, 2021.)*

According to Alex Ivanovs, Front-end frameworks like React and Angular have become increasingly popular in modern front-end development due to their ability to simplify the development process and help maintain performance at scale. React has gained significant traction due to its ability to build reusable UI components and its focus on the declarative programming paradigm. (*Saurabh Sharma, 2023*), *(Alex Ivanovs, 2023.)*

According to Khaleel Inchikkalayil, TypeScript is another technology that has gained popularity in modern front-end development due to its ability to add static typing and other features to JavaScript. *(Inchikkalayil, 2023.)*

4.2 Lerna and Monorepo development

4.2.1 Monorepo

According to Macro, Ricardo, and Gleison (2018) and Nrwl (2022), a monorepo, or a monolithic repository, is a single repository containing multiple distinct projects, with well-defined relationships. In modern software development practices, instead of using separate repositories for each project or component, monorepos store all code in a single repository, often managed by a tool like Lerna. This approach has several benefits:

- Easier code sharing and collaboration: With all code in one place, it's easier to share and collaborate on code between projects and teams.

Developers can easily make changes that affect multiple projects at once, and dependencies are easier to manage.

- Simplified builds and deployments: By having a single repository, builds and deployments can be simplified, reducing the complexity of the development process. It also ensures that all components are built and deployed together, reducing the risk of version conflicts.
- Increased code quality and consistency: Monorepos encourage code reuse and sharing, which can lead to more consistent code across projects. It also enables easier implementation of standards, such as linting and code formatting, across the entire codebase.
- Better testing and bug fixing: With all code in one place, it's easier to test changes across multiple projects and identify and fix bugs that affect multiple components.
- Improved project management: Monorepos enable better project management by providing a central location for all code, reducing the time and effort needed to coordinate and manage multiple repositories. It also allows for more efficient resource allocation and planning.

4.2.2 Lerna

Lerna is a fast, modern build system for managing and publishing multiple JavaScript/TypeScript packages from the same repository *(Lerna, 2023)*. It can be used to improve the development process in several ways:

- Centralized version management: Lerna enables developers to manage the version of all packages in a monorepo from a single location. This ensures consistency and reduces the risk of version conflicts.
- Simplified dependency management: With Lerna, developers can share dependencies between packages in a monorepo. This reduces the duplication of dependencies and simplifies the management of dependencies.

- Streamlined testing: Lerna can be used to orchestrate the testing of multiple packages in a monorepo. This allows developers to ensure that all packages work together and reduces the time required to test changes.
- Improved collaboration: Lerna simplifies collaboration between developers by enabling them to work on multiple packages simultaneously. This reduces the overhead of coordinating changes between packages and speeds up development.

Nrwl (the company behind the open-source build system Nx) has taken over stewardship of Lerna. Nx is a build system developed by ex-Googlers and utilizes many of the techniques used by internal Google tools *(Nx 2023)*. By using tools like Nx or Lerna, developers can efficiently manage and scale their projects with ease. Adopting Nx or Lerna allows for easier collaboration between teams, faster builds, and reduced duplication of effort.

4.3 React

React has emerged as a highly popular JavaScript library for building user interfaces *(Alex Ivanovs, 2023).* According to a Stack Overflow survey conducted in 2019, React has been voted the first most loved web framework, with a 74,5% of respondents expressing their preference for it *(Developer Survey Results 2019).* React has been developed by Facebook in 2011 for internal use and it's now available to the public. It was first used by Instagram in 2012, a Facebook subsidiary *(Marny Lopez, 2022)*. React's popularity is further evidenced by the fact that over 12 million websites worldwide currently use it *(React Usage Statistics, 2023).* Despite being a relatively new library, having been released in 2013, React has quickly become a go-to choice for web developers looking to build modern and dynamic user interfaces.

React has a large and active community, with many open-source libraries and tools available to support development. *(Marny Lopez, 2022.)*

4.4 TypeScript

As a typed superset of JavaScript, TypeScript offers developers the ability to add static types to their code, providing a more reliable foundation for building large-scale applications *(JSQ, 2022)*. As the system will be highly configurable and support a variety of different use cases, it is important to ensure that the code is maintainable and easy to understand. TypeScript can help achieve this by providing static type checking, which can help catch errors and improve code quality.

TypeScript can also provide improved productivity and development efficiency, as it enables developers to catch errors earlier in the development process before they lead to more significant issues. Additionally, TypeScript provides well-presented code documentation, making it easier for developers to understand how different components interact with each other. *(Meredith Shubel, 2022.)*

Furthermore, as TypeScript is a superset of JavaScript, it can be used with many of the same popular JavaScript frameworks, such as React, which was mentioned earlier. This means that developers can leverage the benefits of TypeScript while still using the powerful tools and frameworks that they are already familiar with.

4.5 Redux

When building complex web applications with React, Redux can be used for managing state. According to Redux, Redux is a JavaScript library that helps manage the state of the application in a centralized and predictable manner. However, Redux is independent from React and can be used without React application. In Redux, the entire state of the application is kept in the store. It is like a container for all the data that the components need to take. Actions are plain JavaScript objects that describe what update needs to be made to the state. Reducers are pure functions that take the current state and the dispatched action and return a new state. *(Redux, 2023.)*

4.6 Electron

According to Kevinsawicki, Electron (previously known as Atom Shell) is an open-source framework created by Cheng Zhao and is now developed by GitHub *(Kevinsawicki, 2015).* It allows for the development of desktop GUI applications using front and back-end components originally developed for web applications: Node.js run-time for the back end and Chromium for the front end. The applications themselves are written in HTML, CSS, and JavaScript. *(Maksimchik 2016.)*

According to Vojin Deronjic, one of the key benefits of using Electron is that it allows developers to use web technologies to build desktop applications, which can be more efficient than developing separate applications for different platforms. With Electron, the developer can create a single codebase that can be used to build applications for Windows, macOS, and Linux, which can save significant development time and effort. Furthermore, as Electron applications are based on web technologies, they can be easily integrated with other web-based tools and frameworks, such as React and TypeScript. *(Vojin Deronjic, 2023.)*

4.6.1 Electron & IPC Communication

Electron applications use IPC (Inter-Process Communication) to establish communication between the renderer process and the main process (Electron). This allows the application component and the main Electron process to talk to each other and share information.

IPC messages are used to send messages between the application component and the main process. The application component uses a module called “electron.ipcRenderer” to send messages, while the main process listens for those messages by using another module called “ipcMain”. This communication is very important. The application component can send messages to the main process for a lot of purposes.

By using IPC communication, the application can work with the main process. It enables continuous data transfer and allows the main process to handle another task while the application is opening.

4.7 SVG

An SVG file, short for scalable vector graphic file, is a standard graphics file type used for rendering two-dimensional images that can offer powerful graphics at any scale and improve search engine optimization. *(Jamie Juviler, 2022).* Additionally, editing SVG files directly can be a difficult and time-consuming process, although developing a visual editor can help mitigate this issue.

In this thesis, SVG is used to create certain elements of the ObSAS user interface, and these SVG elements are configured by editing JSON configuration files. By developing a visual editor for SVG-based components, the configuration process can be greatly simplified, and potential errors can be reduced. The visual editor can provide a user-friendly interface that allows the adjustment of SVG configuration properties and the viewing of the resulting SVG in real-time.

4.8 REST

According to Robert Richards, REST (Representational State Transfer) is a way of designing software architectures for the web. It was introduced by Roy Fielding in his research paper. In simple terms, it's about using URLs to identify and access different pieces of information, called resources. When the user uses a web browser to visit a URL, the server sends back the current state of the resource, which is usually a webpage. The browser then displays the webpage to the user. Clicking on links within the webpage takes the user to other URLs, and the user sees different web pages as a result. *(Robert Richards, 2006.)*

To do different things with resources, REST uses common actions like getting (GET), creating (POST), updating (PUT), and deleting (DELETE). These actions are like instructions for the web service *(Codecademy Team.)*

5 OBSERVIS SAS SYSTEM OVERVIEW

5.1 Description of the Observis SAS system

The Observis SAS (Situational Awareness System) is a software developed by Observis Oy. It's designed to help monitor and respond to chemical, biological, radiological, nuclear, and explosive (CBRNe) threats. The SAS system collects data from different sources like sensors and detectors and processes it in real-time to give users a clear understanding of what's happening. The system processes and integrates this data to provide real-time insights and situational awareness to the users.

According to Observis, the ObSAS user interface is unlike anything else on the market. It’s been designed and realized with the latest tools and guidelines for responsiveness, scalability, and usability. *(Observis, 2023.)*

5.1.1 Generic SVG components of the SAS system

Figure 1 shows an example of the SVG component.

Figure 1: SVG-based component

SAS system can be easily customized and configured to fit different needs and situations. One configurable component of the system is the UI. In the UI, SVG images are used to display different elements of the system, such as floor plans of the monitored areas, or air filtration and pressure control systems.

Observis uses custom software to overlay information on top of the SVG images. This information is displayed as text and as markers on top of the SVG image in specific locations. The custom software developed by Observis also allows the SVG images to be changed programmatically based on real-time events in the system. For example, a color of an SVG element may change to red when there is a radiation alarm raised in the system. The locations and properties of the SVG overlays such as markers are currently configured manually in a JSON file. The location of each marker is specified as coordinates on X and Y axes.

5.2 Identification of problems with the current UI configuration method

Based on the conversation with the company developers, the following problems were identified:

- Manual process: The current method of configuring the UI is a manual process that involves creating and modifying JSON configuration files that follow a specific proprietary structure. It takes a lot of time and effort to complete this.
- Error-prone: Since the UI configuration is done manually, this could lead to making errors while creating or modifying the configuration files. This can result in bugs and inconsistencies in the UI.
- Flexibility issues: The current UI configuration is underdeveloped and not flexible. This makes it difficult to adapt to different use cases quickly and efficiently.
- Lack of UI support for SVG elements: Parts of the UI consist of SVG elements, which are only editable in text JSON files, without any UI support. This makes it harder to configure and test these elements.

5.3 Description of the proposed visual editor

This thesis proposes the development of a visual editor that allows employees to easily configure UI components through a user-friendly interface.

The visual editor would provide a more efficient and intuitive way to manage SVG-based components, allowing employees to quickly search and select the right blueprint for their project. Additionally, the editor would improve the configuration process, reducing inconsistencies and errors in the UI across projects.

The suggested visual editor will increase Observis employees' productivity and ensure a high level of quality and consistency throughout the company's projects by providing a simpler and more consistent approach to configuring UI components.

6 DESIGN AND DEVELOPMENT

6.1 Design of the visual editor

The first step of the development of the new editor was to create a UI mockup in Figma. The mockup was made by the UI designer at Observis based on the requirements provided by the thesis worker. This design is made to be simple and functional for Observis employees. It provides all the necessary features, ensuring that employees can easily navigate and utilize its capabilities. Figure 2 shows the SVG Editor main window.

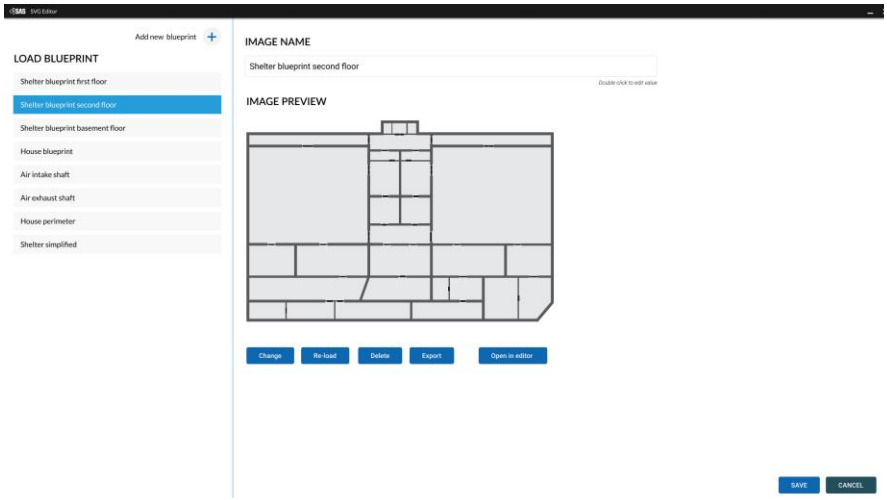

Figure 2. SVG Editor main window (Figma mockup)

Figure 3 shows the feature mockup of the application.

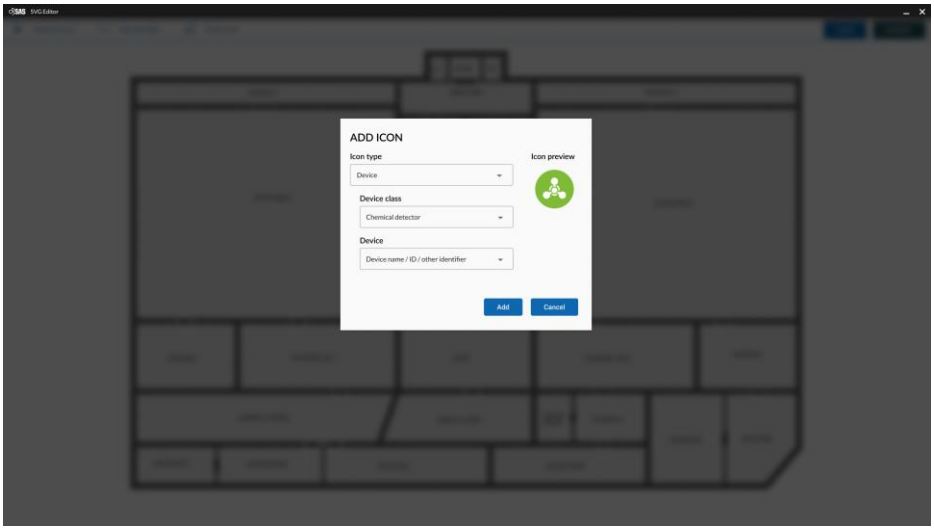

Figure 3. Tool for adding an icon, and text fields in SVG Editor (Figma mockup)

As an internal tool, the visual editor aims to serve Observis employees. This means that the design is flexible and can be modified and enhanced to meet the needs of the employees in the future.

6.2 Implementation

6.2.1 Construction

The visual editor was implemented as a new window to the existing ObSAS client software. Figure 4 shows that an icon had been added to the navigation bar of the main application. By clicking on that icon, the user will be directed to the visual editor.

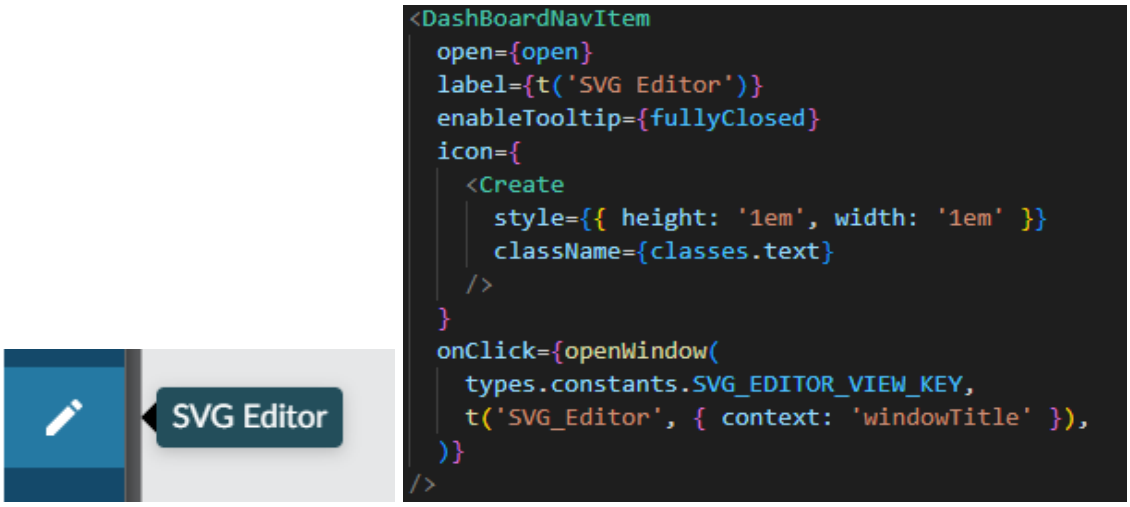

Figure 4. SVG Editor icon

Currently, the visual editor is intended for internal use. The following line of code in Figure 5 was implemented to make sure that the visual editor only appears during the DESIGNTIME mode. DESIGNTIME mode is the special mode to run the software that unlocks additional features for developers. These features are only for the developers and are hidden from the customer.

```
applicationMode === types.navigation.ApplicationMode.DESIGNTIME
```

Figure 5. Conditional statement for designer mode

The visual editor has 2 main components, “SVGEditorPanel” and “LoadBlueprint” “respectively. The “SVGEditorPanel” component is the main panel of the SVG Editor. It creates the home view of the editor and uses the <GenericSvgContainer/> component to show the SVG-based component. The “LoadBlueprint” component displays the name of the SVG image that is currently being edited. It allows the user to add markers overlays to the SVG and save changes to the mapping file. The mapping file is a JSON file that contains all configurations for the SVG overlays, including markers and their locations.

6.2.2 Dependencies

Figure 6 shows that the SVG Editor uses some external dependencies.

- The SVG Editor is built using React, a popular JS library for building UI.
- Material-UI provides a set of pre-built components and styling options.
- Electron allows the creation of desktop applications using web technologies.
- The ob2-generic-svg is a library in the ObSAS system that works with SVG files in the main application. It provides components for loading, displaying, and interacting with SVG files.

```
import * as React from 'react';
import LoadBlueprint from './LoadBlueprint';
import { GenericSvgContainer } from 'ob2-generic-svg';
import electron from 'electron';
import { Grid, StyledComponentProps, makeStyles } from '@material-ui/core';
```

Figure 6. Dependencies

6.2.3 Code structure

When opening the SVG Editor, the “SVGEditorPanel” component will be rendered. This component creates a window using the <DetachPanelRoot>, <DetachPanelHeader>, and <DetachPanelContent> from the “ob2-generic-svg” library. There is also a header with the title “SVG Editor”.
The <ErrorBoudary> component will catch errors that may appear while rendering and throw an error message in case of any errors.
The main content of the SVG Editor panel has 2 sections. The left section takes 3/12 of the space and displays the “LoadBlueprint” component. The right section takes 9/12 of the space and displays the blueprint by rendering the <GenericSVGContainer> component. Figure 7 shows the SVGEditorPanel component.

```
return (
  <div className={classes.root}>
    <DetachPanelRoot>
      <DetachPanelHeader title={t('SVG Editor')} onClose={onCloseClick} />
      <DetachPanelContent>
        <ErrorBoundary FallbackComponent={ErrorFallback}>
          <Grid container style={{ height: '100%' }}>
            <Grid item xs={3} style={{ height: '100%', overflow: 'auto' }}>
              <LoadBluesprint
                svgFileName={props.svgFileName}
                setSvgContainerVisible={setSvgContainerVisible}
              />
            </Grid>
            <Grid item xs={9}>
              {svgContainerVisible && (      You, 5 days ago • Add and save marker
                <GenericSvgContainer
                  svgFileName={props.svgFileName}
                  zoomEnabled={props.zoomEnabled}
                  margin={props.margin}
                  padding={{
                    top: props.paddingTop,
                    bottom: props.paddingBottom,
                  }}
                  ipcRenderer={electron.ipcRenderer}
                />
              )}
            </Grid>
          </Grid>
        </ErrorBoundary>
      </DetachPanelContent>
    </DetachPanelRoot>
  </div>
);
```

Figure 7. “SVGEditorPanel” component

The “LoadBlueprint” component in Figure 8 will load and display the SVG blueprint name and markers on the blueprint. It fetches the mapping file with the SVG file in the “props.svgFileName” and displays the blueprint name. There are text fields that the user can input the value of x and y which are the coordinates for the marker. There also have buttons to add markers and save changes.

```
return (
  <Table stickyHeader aria-label="sticky table">
    <TableHead>
      <Card>
        <Box className={classes.blueprintAndAddButtonRow}>
          <Typography variant="h3" align="center">
            Bluesprint
          </Typography>
          <Button
            color="primary"
            onClick={handleSave}
            aria-label={'Apply changes'}
          >
            Save
          </Button>
        </Box>
      </Card>
    </TableHead>
    <TableBody>
      <Box className={classes.blueprintAndAddButtonRow}>
        <Typography variant="h5" align="center">
          {props.svgFileName}
        </Typography>
        <Button color="primary" onClick={addMarker} aria-label={'Add Marker'}>
          <AddCircleOutlineSharp></AddCircleOutlineSharp>
        </Button>
      </Box>
      <CardContent>
        <Box>
          <TextField
            type="number"
            placeholder="x"
            label={'X coordinate'}
            value={newMarkerXCoordinate}
            onChange={e => setNewMarkerXCoordinate(Number(e.target.value))}
          />
          <TextField
            type="number"
            placeholder="y"
            label={'Y coordinate'}
            value={newMarkerYCoordinate}
            onChange={e => setNewMarkerYCoordinate(Number(e.target.value))}
          />
        </Box>
      </CardContent>
      <CardContent>
        {svgMapping?.markers.map(marker => (
          <p>{marker.name}</p>
        ))}
      </CardContent>
    </TableBody>
  </Table>
);
```

Figure 8. “LoadBlueprint” component

When the user clicks on the “Add Marker” button, the “addMarker” function will be called. The function creates a new marker object and inserts it into a list of markers in the existing SVG mapping file. The updated mapping is then saved into the state of the component. Figure 9 shows the “addMarker” function.

```
  const newMapping = _.cloneDeep(svgMapping);
  newMapping.markers.unshift({
    name: 'Hello world' + new Date().getTime(),
    location: { x: newMarkerXCoordinate, y: newMarkerYCoordinate },
    size: 50,
    icon: 'PRESSURE',
    color: 'DISCONNECTED',
  });

  setSvgMapping(newMapping);
};
```

Figure 9. “addMarker” function

By clicking the “Save” button, the “hadleSave” function will be called. This function will save the updated SVG mapping by passing the new mapping to the IPC renderer. After that, the “props.setSvgContainerVisible(false)” function will be called, it makes the “GenericSvgContainer” component hide temporary. By calling the “props.setSvgContainerVisible(true)” function, the SVG will be visible again. This will render the SVG container and fetch the updated SVG mapping by using the “fetchMapping” function. This is a temporary workaround to ensure the SVG mapping is reloaded after the changes. A more advanced solution would listen to changes to the file and reload automatically. Figure 10 shows the “handleSave” function.

```
const saveMapping = (newMapping: types.genericSvg.SvgMapping) => {
  electron.ipcRenderer.send(
    'saveSvgMappingModel',
    JSON.stringify(newMapping),
    props.svgFileName,
  );

  props.setSvgContainerVisible(false);

  setTimeout(() => {
    props.setSvgContainerVisible(true);
    fetchMapping();
  }, 500);
};
```

Figure 10. “handleSave” function

6.2.4 State management

Figure 11 shows that the “LoadBlueprint” component uses React’s “useState” hook to manage the state. It declares the below states using “useState”:

- “svgMapping”: stores the mapping data for the SVG.
- “newMarkerXCoordinate” and “newMarkerYCoordinate”: stores the X and Y coordinates for a new marker. X and Y are editable by the user from the visual editor.

State updates are handled through the respective setter functions “setSvgMapping”, “setNewMarkerXCoordinate”, and “setNewMarkerYCoordinate”.

```
const LoadBlueprint: React.FC<SvgPropertiesPanelProps> = props => {
  const classes = useStyles();

  const [svgMapping, setSvgMapping] = React.useState<
    types.genericSvg.SvgMapping | undefined
  >(undefined);

  const [newMarkerXCoordinate, setNewMarkerXCoordinate] = React.useState(0);

  const [newMarkerYCoordinate, setNewMarkerYCoordinate] = React.useState(0);
```

Figure 11. Manage states in the “LoadBlueprint” component

These state management allows effective state updates and communication between components, ensuring that the renderer throughout the application is consistent.

7 RESULTS AND FUTURE WORK

As the result of this thesis, the prototype of the visual editor has been created. Figure 12 shows the visual editor.

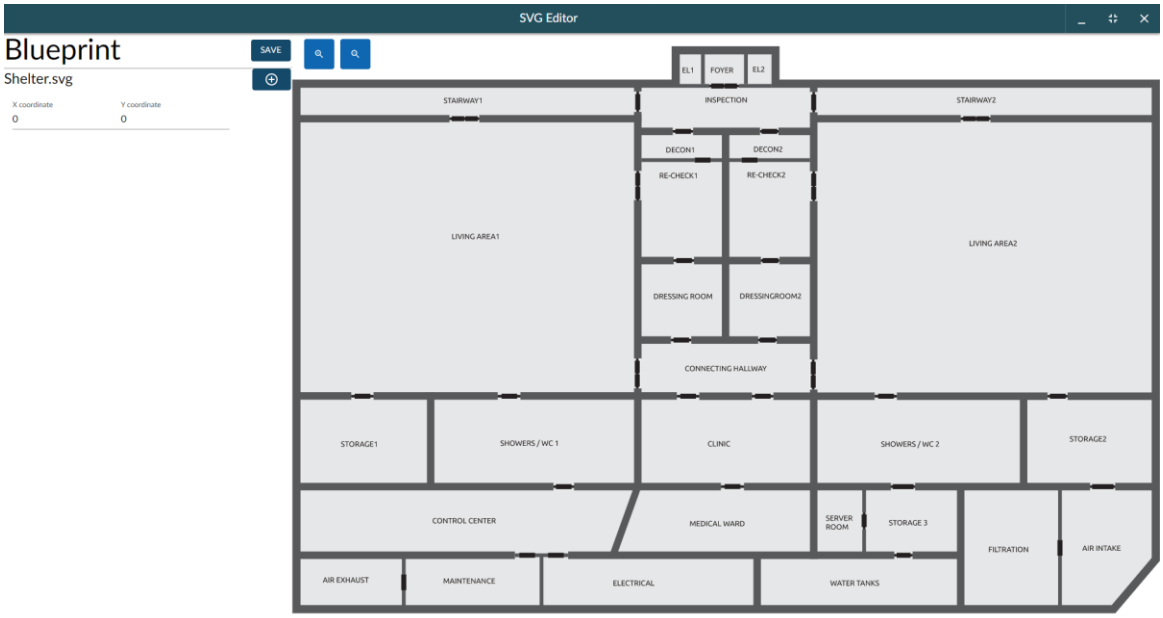

Figure 12. The SVG editor

The user can set the location of the marker by inputting the X and Y coordinates. Figure 13 shows that the marker has been added to the mapping file by clicking the plus button after inserting the coordinates.

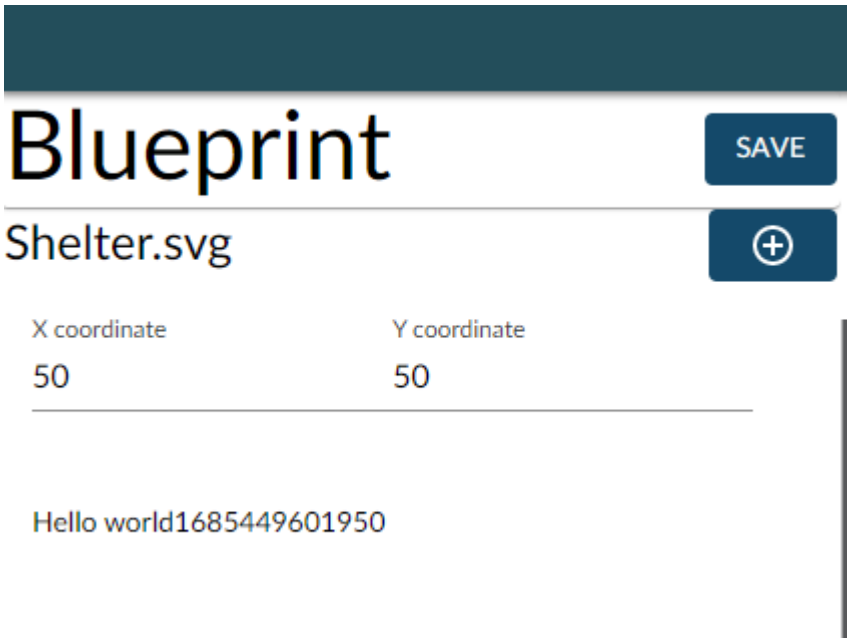

Figure 13. Add marker to the existing mapping file

After the save button was clicked, the marker appeared in the SVG image. Figure 14 shows the marker has been added to the top left corner of the SVG image.

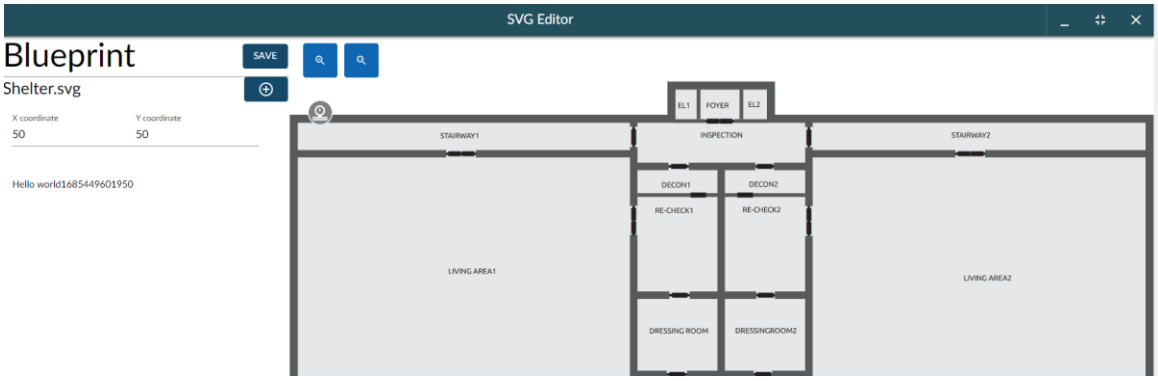

Figure 14. The marker was successfully added to the SVG image

However, the visual editor was not completed because of the time limitation. A feature that allows for the simulation of real-time messages need to be included. This addition would enable users to view the status of the devices through markers. Another feature needed for future development is the ability to preview modified markers. This feature would allow users to move the marker to wherever they want (drag and drop for example) and modify other properties of the marker, not just the coordinates.

Continued efforts in refining and expanding the visual editor's capabilities will create an even more powerful tool for users at Observis. Regular updates and development will ensure that the visual editor meets the need of the users and the company.

8 CONCLUSION

In summary, this thesis project has successfully solved the original problem by creating a visual editor that helps configure SVG-based components in the ObSAS user interface at Observis Oy. This is a functional prototype for an editor. This prototype proves the feasibility of a complete solution in the near future.

Based on this work, Observis can develop this editor further, leading to a useful tool that will reduce manual work.

Furthermore, the programming techniques and tools used in this project have broader applications in other types of software development. By utilizing web technologies and component-based architectures, intuitive and customizable interfaces for configuring different elements can be created in various domains. For example, in a content management system, similar programming techniques can be employed to develop a visual editor that enables users to easily customize and arrange webpage elements. The flexibility and potential of web technologies and component-based approaches make them applicable to diverse software projects.

REFERENCES

Joe Johnston. 2022. What are internal tools? WWW document. Available at: https://budibase.com/internal-tools/ [Accessed 4 April 2023].

Sebastian Duncan. 2023. Understanding the difference between internal & external customers. Web page. Available at: https://realbusiness.co.uk/internal-and-external-customer#:~:text=Internal%20customers%20are%20employees%20or,purchase%20your%20products%20or%20services. [Accessed 4 April 2023].

Elizabeth Kampf. 2020. Want to drive success? Give your employees the same experience as your customers. XM blog. Available at: https://www.qualtrics.com/blog/internal-customers/ [Accessed 5 April 2023].

GitHub Inc. 2023. Developer Experience. WWW document. Available at: https://microsoft.github.io/code-with-engineering-playbook/developer-experience/. [Accessed 4 May 2023].

Eduardo Pineda. 2019. The evolution of Frontend Development. WWW document. Available at: https://www.epineda.net/the-evolution-of-front-end-development/. [Accessed 14 March 2023].

Alex Ivanovs. 2023. The most Popular Front-end Frameworks in 2023. WWW document. Available at: https://stackdiary.com/front-end-frameworks/. [Accessed 14 March 2023].

Khaleel Inchikkalayil. 2023. While TypeScript is gaining popularity, it's unlikely that it will completely dominate over JavaScript. Web page. Available at: https://www.linkedin.com/pulse/while-typescript-gaining-popularity-its-unlikely-over-inchikkalayil/. [Accessed 6 March 2023].

Lerna. 2023. The Original Tool for JavaScript Monorepos. Web page. Available at: https://lerna.js.org/. [Accessed 19 March 2023].

Nx. 2023. Webpage. Available at: https://lerna.js.org/docs/lerna-and-nx. [Accessed 20 March 2023].

Macro Tulio, Ricardo Terra & Gleison Brito. 2018. Monorepos: A multivocal literature Review. Publication archive. Available at: https://arxiv.org/abs/1810.09477. [Accessed 4 March 2023].

Nrwl. Monorepo Explained, 2022. WWW document. Available at: https://monorepo.tools/
Developer Survey Results. 2019. Stackoverflow webpage. Available at: https://insights.stackoverflow.com/survey/2019. [Accessed 4 March 2023].

Marny Lopez. 2022. Why React is so widely adopted by web developers?. Personal blog. Available at: https://www.devlane.com/blog/why-react-is-so-widely-adopted-by-web-developers [Accessed 19 March 2023].

React Usage Statistic. 2023. Web page. Available at: https://trends.builtwith.com/javascript/React [Accessed 6 March 2023].

Saurabh Sharma. 2023. Web page. Available at: https://dev.to/itsjzt/declarative-programming--react-3bh2 [Accessed 6 March 2023].

JSQ. 2022. What is TypeScript: A Powerful Programming Language? WWW document. Available at: https://iq.js.org/questions/typescript/what-is-typescript-a-powerful-programming-language [Accessed 4 March 2023].

Meredith Shubel. 2023. What is TypeScript. Web page. Available at: https://thenewstack.io/what-is-typescript/ [Accessed 6 March 2023].

Kevinsawicki. 2015. Atom Shell is now Electron. The original announcement post. Available at: https://www.electronjs.org/blog/electron [Accessed 6 April 2023].

Maksimchik J., 2016. Electron Awesome. WWW document. Available at: https://slides.com/juliamaksimchik/electron-awesome [Accessed 6 April 2023].

Vojin Deronjic. 2023. Electron Software Framework: The Best Way to Build Desktop Apps. WWW document. Available at: https://www.pangea.ai/dev-web-development-resources/electron-software-framework-the-best-way-to-build-desktop-apps/ [Accessed 6 April 2023].

Jamie Juviler. 2022. SVG Files: What They Are and How to Make One. Web page. Available at: https://blog.hubspot.com/website/what-is-an-svg-file#what-is-svg [Accessed 16 May 2023].

Robert Richards. 2006. Pro PHP XML and Web Services pp 633-672. eBook.
Codecademy Team. What is REST. Webpage. Available at: https://www.codecademy.com/article/what-is-rest [Accessed 16 May 2023].

Observis. 2023. ObSAS Situational Awareness. Web page. Available at: https://www.observis.fi/obsas/products/obsas-software/ [Accessed 16 May 2023].

Redux. 2023. Web page. Available at: https://redux.js.org/introduction/getting-started [Accessed 24 May 2023].