

Eveliina Hämäläinen

WEBUI IMPLEMENTATION OF DESIGN PATTERNS FOR DRILLING MINING MACHINE CONTROL SYSTEM

Bachelor's thesis
Faculty of Engineering and Natural Sciences
Examiner: Veli-Pekka Pyrhönen
May 2023

TIIVISTELMÄ

Eveliina Hämäläinen: Suunnittelumallien toteutus poralaitteen ohjausjärjestelmän web-käyttöliittymälle

Kandidaatintutkielma

Tampereen yliopisto

Automaatiotekniikan tutkinto-ohjelma

Toukokuu 2023

Suunnittelumallit ovat ohjelmistotekniikassa käytettäviä yleisiä ratkaisuja usein toistuviin ongelmiin. Suunnittelumalleilla voidaan dokumentoida hyväksi todettuja käytäntöjä ja ne tarjoavat apua erilaisten ongelmien ratkaisujen johdonmukaiseen toteutukseen. Tässä työssä tutkitaan maanalaisen poralaitteen ohjausjärjestelmiä ja ohjelmistoarkkitehtuuria sekä esitellään niissä käytettyjä web-tekniikoita. Lisäksi poralaitteen web-käyttöliittymille kerätään ohjelmistovaatimuksia ja etsitään erilaisia suunnittelumalleja, joiden tarkoituksena on dokumentoida parhaita käytänteitä kohdeyrityksen web-sovellusten kehitykseen. Tämän työn tavoitteena on löytää ja dokumentoida erilaisia suunnittelumalleja poralaitteen web-käyttöliittymille.

Web-käyttöliittymien vaatimustenkeruuta varten haastateltiin kahta kohdeyrityksen ohjelmisto- ja ohjausjärjestelmäasiantuntijaa maaliskuussa 2023. Haastatteluissa korostettiin yleisten ohjelmistovaatimusten lisäksi poralaitteille erityisiä vaatimuksia kuten laitteiden pitkää käyttöikää ja korkeita turvallisuusvaatimuksia. Lisäksi haastatteluissa korostuivat kaksi eri näkökulmaa: mitä vaatimuksia ohjelmistoarkkitehtuuri asettaa ja mitä vaatimuksia asiakkaat asettavat web-käyttöliittymille poralaitteissa. Vaatimukset jaettiin haastatteluaineiston perusteella kolmeen eri kategoriaan.

Tätä työtä varten toteutettiin uusi web-käyttöliittymä Sandvikin sähkökäyttöiseen poralaitteeseen. Poralaitteen web-käyttöliittymille etsittiin erilaisia suunnittelumalleja uutta käyttöliittymää sekä vanhoja käyttöliittymätoteutuksia tutkimalla. Erilaisia suunnittelumalleja tutkittiin myös kirjallisuudesta.

Työn tuloksena löydettiin kolme erilaista suunnittelumallia. Kirjallisuudesta löytyi kymmeniä suunnittelumalleja, joista tähän työhön valittiin yksi. Tämä suunnittelumalli tarjoaa ratkaisun suurien web-sovellusten tilan hallintaan, kun sovelluksessa on paljon samaa tietoa, jota sovelluksen usea eri osa käyttää. Kaksi muuta suunnittelumallia löydettiin tutkimalla tätä työtä varten toteutettua web-sovellusta ja vanhoja sovellustoteutuksia. Ensimmäinen löydetty malli liittyy erilaisten web-komponenttien uudelleenkäyttöön ja niiden varastointiin niin, että ne ovat saatavilla kootusti komponenttikirjastossa. Usein samoja komponentteja käytetään useissa eri projekteissa, jolloin tätä mallia käyttämällä tarve kopioida koodia useisiin eri projekteihin poistuu. Toinen löydetty malli tarjoaa ratkaisun tehokkaaseen web-testaukseen määrittelemällä jokaiselle web-komponentille pakollisen ainutlaatuisen tunniste.

Avainsanat: Web-käyttöliittymä, ohjelmistokehitys, suunnittelumalli, maanalainen poralaitte, web-tekniikat

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

ABSTRACT

Eveliina Hämäläinen: WebUI implementation of design patterns for drilling mining machine control system
Bachelor's thesis
Tampere University
Automation technology
May 2023

Design patterns are generic solutions to commonly occurring problems in software engineering. Design patterns are a means for capturing best practices of implementations and provide a guide for implementing solutions in a consistent manner. This work examines underground drilling mining machines' control systems and software architecture and presents used web technologies. Additionally, the work includes collection of software requirements and study of different design patterns for drilling mining machines' web user interfaces. Design patterns are used to document best practices for web application development for the target company. The target of this work is to select and document various design patterns for drilling mining machines' web user interfaces.

Two software- and control system specialists of target company were interviewed for collection of requirements in March 2023. The interviews highlight specific requirements for drilling mining machines, which are related to, for example, the long-life cycle and high safety demands of the machines. Additionally, two different perspectives were highlighted in the interviews: what requirements software architecture sets and what are the customer demands for web user interfaces in drilling mining machines. Requirements were divided in to the three categories based on interview material.

A new web application for Sandvik battery-electric mining jumbo was developed for this work. To find design patterns for drilling mining machines' web applications, new web application and several already existing web applications were studied. Various design patterns were studied from literature.

Three different design patterns were discovered as a result of this work. Numerous design patterns were found from literature, and one was selected for this work. This design pattern offers a solution for state management of large web applications, when application contains large amount of same data, that multiple different parts of the application use. Two different design patterns were found by studying the web application developed for this work and old application implementations. First discovered pattern is related to reusability of various web components and their storing in a way that they are easily accessible from the component library. Same components are regularly used in various projects, and by implementing this pattern the need to copy code to multiple different projects is not required. Second discovered pattern offers a solution to efficient web testing by defining a mandatory and unique id for each web component.

Keywords: Web user interface, software development, design pattern, underground drilling mining machine, web technologies

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

FOREWORD

I have been able to develop myself, my skills and learn a lot from software development. At the beginning of this work, web technologies were almost completely new for me. At the beginning, the amount of new things felt overwhelming but when I progressed with the work I experienced many successful moments. I appreciate all the experience I gained on the web development.

I want to take this opportunity to thank my supervisor Esa Vikman for interesting topic, the opportunity to learn new things, and assistance during this work. I also want to take this opportunity to thank Jouni Lindgren and Petri Nurminen, who were interviewed for this work. I would also want to thank all my colleagues for their support and help. I would like to extend my sincere thanks to Veli-Pekka Pyrhönen for constructive feedback and support.

Tampere, 29.5.2023

Eveliina Hämäläinen

TABLE OF CONTENTS

1. INTRODUCTION	1
2. MINING MACHINE CONTROL SYSTEM ARCHITECTURE	2
2.1 SICA 2.x Architecture	3
2.2 SICA 3.x Architecture	5
2.3 Used web technologies.....	6
2.3.1 HTML.....	6
2.3.2 CSS.....	8
2.3.3 JavaScript.....	9
2.3.4 TypeScript.....	10
2.3.5 Vue.js	11
2.4 Requirements for web applications in mining machine	13
3. DESIGN PATTERNS	18
3.1 The structure of a design pattern.....	19
3.2 Categories of design patterns	20
3.3 Implementation of design patterns.....	21
3.3.1 The Library pattern	22
3.3.2 The ID pattern	23
3.3.3 Pinia State Management Pattern.....	25
4. SUMMARY	30
SOURCES	32

ABBREVIATIONS AND NOTATIONS

UI	User Interface
SICA	Sandvik Intelligent Controls and Automation
SUP	SUPervisor
MC	Machine Control
GUI	Graphical User Interface
CU	Control Unit
MCC	Machine control module platform with C implementation
CAN-bus	Controller Area Network, electronic communication bus
WLAN	Wireless Local Area Network
IO	Input Output
HMI	Human Machine Interface
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
API	Application Programming Interface
DOM	Document Object Model
NPM	Node Package Manager
XPath	XML Path Language

1. INTRODUCTION

In web development and in the development of many other things, the same problem can be solved multiple ways and some problems occur more often than others. Therefore, similar problems may be solved within a development team of a company in several ways, which may waste time and resources. One solution to this problem is to use design patterns.

A design pattern is a general solution to a commonly occurring problem, whether for software, web design or buildings. Design patterns are a means for capturing best practices of implementations and provide a guide for implementing solutions in a consistent manner. With web design patterns, the web developers could create reusable and modular code which improves quality and improves development time. Consequently, development becomes more efficient and consistent. [1, p. 4]

Work machine UI (User Interface) is traditionally tightly integrated to machine control system and with fixed resolution. In Sandvik, the traditional UI implementation is done using Qt Framework. The future target is to make user interface views more independent from the rest of the control system using web technologies.

One major benefit when using web technologies is that machine views can be used outside of the machine itself for example in mobile devices. In long lifetime of the machine, the scalable web UIs also bring possibility to transfer widescreen displays in machine control system.

However, just selecting the web UI framework and tools is not enough for implementing long-lifetime Web UIs for mining machines. To use the web technologies efficiently, it is necessary to agree what design patterns or architecture is used in mining machine views.

The target of this work is to define implementation design patterns for mining machine web UIs in a way that UI implementation and maintenance are efficient and fills the mining machine requirements. The work includes literature study of design patterns, collection of requirements for mining machine web UIs and implementation and selection of design patterns used in views. Furthermore, the work investigates drill rig's control system and software architecture and outlines the web technologies used at Sandvik.

2. MINING MACHINE CONTROL SYSTEM ARCHITECTURE

Sandvik manufactures several different types of underground drilling equipment for various needs. Sandvik underground mining drill rigs are used for mining development and production and can be deployed for example installing rock support or drilling to break up ore [2]. Table 1 presents different types of underground drill rigs manufactured by Sandvik.

Table 1. *Different types of Sandvik underground drill rigs [2].*

Development drill rigs	For applications ranging from face drilling for small-scale mine development to large-scale tunneling.
Tunneling Jumbos	For fast face drilling or mechanized long-hole drilling and bolting.
Top hammer longhole drill rigs	For mechanized and automated underground mass mining. Capable of drilling 51–127 mm diameter holes up to 54 meters in length.
Rock support drill rigs	For rock reinforcement.
In-the-hole longhole drill rigs	Produce long and straight holes at depths greater than 100 meters.
Low profile drill rigs	For work in tabular ore bodies, such as chrome and platinum mines.
Narrow vein drill rigs	For narrow vein drifts and small tunnel projects.

Figure 1 presents Sandvik's DD422i development drill rig. It is used for underground mine development and small scale tunneling [2].



Figure 1. DD422i development drill rig [3].

Sandvik iSeries underground drilling mining machines manufactured in Tampere have SICA (Sandvik Intelligent Controls and Automation) based control system. Sandvik iSeries underground drill rigs include different automation options and digital services, for example, fully automated face drilling.

SICA is control system platform. Platform means that SICA includes software libraries, documents and hardware modules and it is not ready-made application. [4, p. 6] Sub-section 2.1 introduces SICA 2.x control system architecture and subsection 2.2 SICA 3.x control system architecture and drivers for moving towards SICA 3.x architecture.

2.1 SICA 2.x Architecture

The SICA 2.x architecture foundation is based on division of two parts, SUP (SUPervisor) and MC (Machine Control). SUP is a soft-realtime environment for GUI (Graphical User Interface), data collection and other similar tasks. Software in SUP level is based on Qt C++ framework which is used for developing GUIs and cross-platform applications [5, p. 1]. SICA SUP contains ready-made views, services, and libraries for applications to be build [4, p. 7].

MC is a hard-realtime control application platform, including CUs (Control Unit) running MCC (Machine Control module platform with C implementation) applications, CAN (Controller Area Network) buses and IO-modules (Input Output) [4, p. 7]. MCC is for real-time machine control, and it is roughly comparable, for example, to Beckhoff's TwinCAT. It is software platform for all supported target hardware. [4, p. 8] SUP-level and MC-level

communicate only with MCon which is a component providing process image for SUP application. Figure 2 presents architecture of SICA 2.x control system in drill rig view.

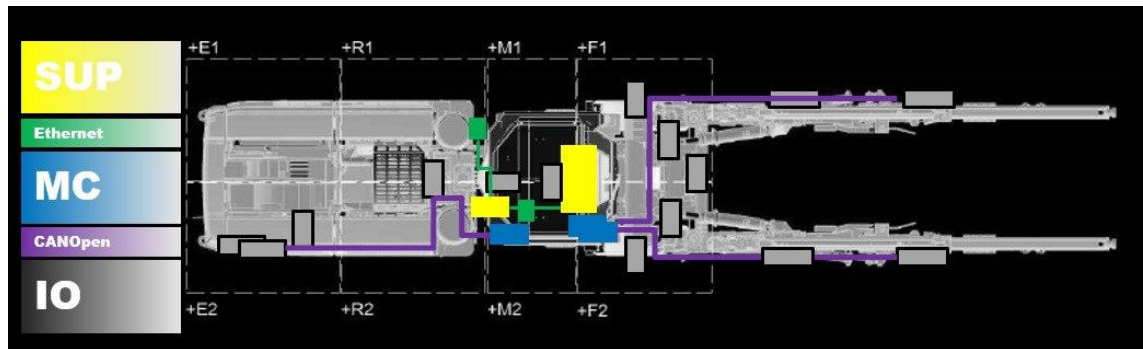


Figure 2. Physical view of the drill rig's control system [6, p. 8].

In figure 2, areas E1-2, R1-2, M1-2, and F1-2 form an upper-level machine coordinate system. Location of different components can be presented using this coordinate system. Figure 3 presents architecture layers in SICA 2.x control system. SUP contains graphical user interface or views and SUP services which includes non-real time control logic and data management [6, p. 6].



Figure 3. Architecture layers and hardware [6, p. 7].

Figure 3 shows the hardware on different layers. SUP-level contains drilling- and tramming display. Ethernet-level contains ethernet switch and WLAN (Wireless Local Area Network) client. MC-level contains embedded Linux computers. Last level contains IO-slave, HMI (Human Machine Interface) control panel and HMI push-button panel.

The 2.x generation software is implemented in one package and is typically composed of shared components, but it must be built and tested as one entity [6, p. 17]. It means that to have SICA-based real product both SUP and MC are needed. SICA 2.x -based control system is currently the standard for iSeries underground drill rigs manufactured in Sandvik's Tampere site.

2.2 SICA 3.x Architecture

The SICA 3.x is a major architecture update, and it moves control system architecture from integrated single SUP having single application towards distributed, multiplatform architecture. Main changes from SICA 2.x architecture are remote invocation and system wide events, clear separation of data, data processing, system control and UI (User Interface).

The SICA 3.x architecture is hardware compatible with SICA 2.x architecture meaning that it supports the lifetime update of any SICA 2.x machine without major changes to hardware. The SICA 3.x system is built from coupled modules, which are not bound together at build time and can be built as independent submodules. Despite of the distributed and modular control system, the user experience must be like using integrated machine. [6, p. 16]

The 3.x generation software consists of small, build time independent releases. It consists of application software release and carrier software release. [6, p. 17] This enables modularization, for example if both diesel- and battery drill rig variants are required for the same mining application, the same application software can be used in both variants, but the carrier software used is different. This limits the number of software variants because there is no need to implement an entire software package for each different product variant. Both releases can be tested as its own entity [6, p. 17]. Figure 4 presents architecture of SICA 3.x control system.



Figure 4. Architecture layers and hardware [6, p. 21].

In SICA 3.x architecture, SUP views are separated from SUP services. SUP services are included in carrier software. Figures 3 and 4 have one difference between each other: the WLAN client component is replaced with data collection and connectivity gateway

unit also called Knowledge Box. This connectivity gateway has multiple functionalities in it such as functions as a WiFi Access Point or a WiFi client [7].

GUIs in SICA 2.x architecture were built with Qt. In SICA 3.x, views are built with latest web technologies including HTML (HyperText Markup Language), CSS (Cascading Style Sheets), JavaScript, and web based Vue.js. Views run in a browser-like environment and system has a front- and backend. The backend contains all data and application state, and it can serve multiple frontends [6, p. 22]. The frontend can display data from multiple backends, and data is available for frontend through APIs [6, p. 22].

During transition from SICA 2.x to SICA 3.x, new views built with web technologies can be embedded as part of Qt display [6, p. 24]. It means that migration of existing content from Qt to web view can be done step-by-step and SICA 3.x features can be implemented in SICA 2.x releases when it is practical.

There are multiple drivers for moving from SICA 2.x architecture to SICA 3.x architecture. There is need to limit the number of software variants and split SUP views from SUP services [6, p. 14]. Also, in 2018, decision was made not to engage in commercial Qt [6, p. 14]. Responsive UIs built with web technologies bring possibility to use widescreen displays in machine control system in the future and machine views can be used outside of the machine for example in a tablet or mobile device or anything that can display web views.

2.3 Used web technologies

Modern web sites and web applications are built with three core technologies of web, HTML, CSS, and JavaScript [8, ch. 8] and with different frameworks such as AngularJS or React. In Sandvik, web applications are developed using the following technologies: HTML, CSS, JavaScript, TypeScript and Vue.js. In the next five subsections, used technologies are briefly explained.

2.3.1 HTML

HTML is specification language and the backbone of all web pages [9, ch. 3]. It provides structure to the content on a website including buttons, videos, images, text and more. The current version used is HTML5, which was launched in 2012 [9, ch. 3]. HTML uses elements to form structure of the web pages and majority of elements have opening and closing tag. [10, ch. 2] One example of the HTML element is `<p>` element that defines a paragraph in HTML document and it has an opening tag `<p>` and a closing tag `</p>`.

Tags are like annotations that provide information about the type of the content they contain. In other words, HTML element is an individual component in an HTML document, and it is formed by opening and closing tags and the media or text it contains.

HTML elements form a tree-like structure in an HTML document and the structure can be described like a family tree. Program 1 presents the structure of a simple HTML document and a syntax of the HTML language.

```
1    <!DOCTYPE html>
2    <html>
3      <head>
4        <title>Example site</title>
5        <link rel="stylesheet" href="style.css">
6      </head>
7
8      <body>
9        <header>
10         <h1>My Example Site</h1>
11         <nav>
12           <a href="#"><li>Navigation 1</li></a>
13           <a href="#"><li>Navigation 2</li></a>
14           <a href="#"><li>Navigation 3</li></a>
15         </nav>
16       </header>
17
18       <main>
19         <p id="paragraph">Here is a paragraph!</p>
20         <button id="hide">Hide</button>
21       </main>
22       <script src="main.js"></script>
23     </body>
24   </html>
```

Program 1. *The structure of the simple HTML document.*

A diagram of the HTML document tree of program 1 is presented in figure 5. The diagram shows more clearly the tree-like structure of HTML elements than HTML document does.

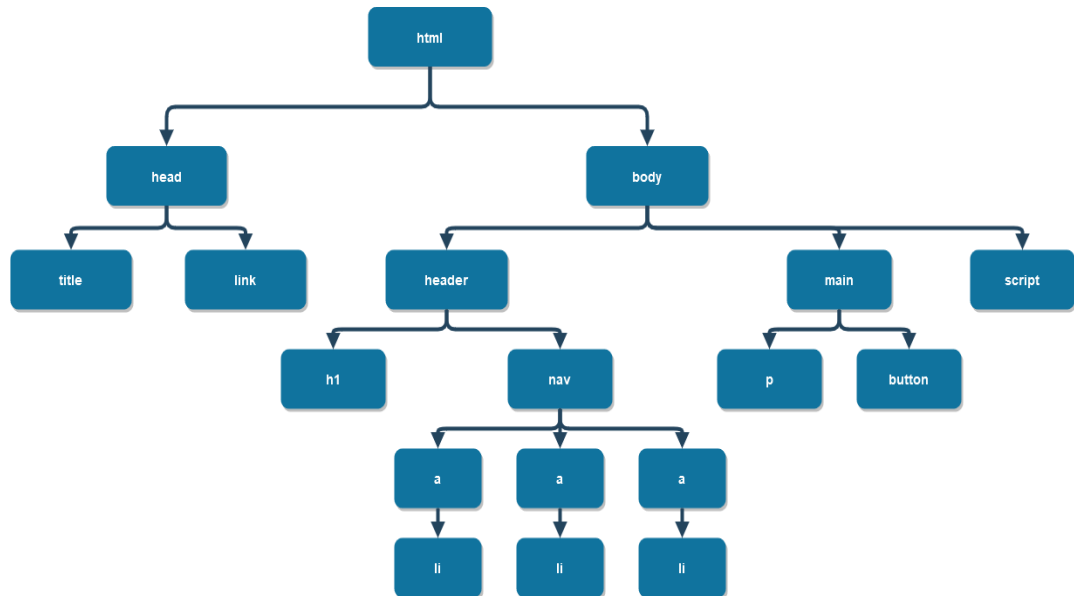


Figure 5. A diagram of the HTML example document.

Program 1 and figure 5 show that HTML elements can be parallel or nested. Nested and parallel elements can be described with family relations. In program 1 and figure 5, the `<html>` element is the ancestor of all elements in the HTML document. It can also be said that all the elements inside the `<html>` are descendants of the `<html>` element. The structure can also be described with parent, child, and sibling relations. For example, the `<nav>` element is the parent of three `<a>` elements and `<a>` elements are children of the `<nav>` element. Elements that are parallel, are siblings. In the example in program 1 and figure 5, the `<h1>` element and the `<nav>` element are siblings.

2.3.2 CSS

CSS is a layout and formatting language, and it is used to format HTML. CSS contains selectors that specify HTML elements to which the style should be applied and visual rules that specify how that selected content should be displayed. [9, ch. 4] For example, CSS can be used to change the font and color of the heading or paragraph. In other words, CSS controls the look of the web page, colors, and appearance. It is also possible to control different display modes for different devices based on the width and height of the screen with CSS [9, ch. 4]. The current version used is CSS3 [9, ch. 4]. Program 2 presents the syntax of the CSS.

```

1  html {
2      font-family: 'Segoe Ui';
3  }
4
5  h1 {
6      color: blue;
7      font-size: 32px;
8  }
9
10 p {
11     color: red;
12 }

```

Program 2. Example of the syntax of the CSS language.

The example CSS document in program 2 adds styling to example site presented in subsection 2.3.1 in program 1. *html* CSS selector selects all `<html>` elements and gives them and their descendants font “Segoe UI”. *H1* selector selects all `<h1>` elements and gives them color blue and font size 32. *P* selector selects all `<p>` elements and gives them color red. Figure 6 presents example web site styled with CSS. The web page consists of the HTML document in program 1 and the CSS file in program 2.

My Example Site

[Navigation 1](#)

[Navigation 2](#)

[Navigation 3](#)

Here is a paragraph!

Hide

Figure 6. The example web page styled with CSS.

It is easy to see with program 1, 2 and figure 6 how styles are applied to the web page. Without CSS formatting, it would be difficult to distinguish different parts of the web pages and navigate in the web pages, among other things.

2.3.3 JavaScript

JavaScript is a programming language that provides interactivity to web sites and web applications. It can be used on server and client side of applications. The server side of an application is the backend which usually interacts with databases and runs in data centers. The client side of an application is the frontend, and it runs on the device of the user, usually in browser. [11, ch. 1] For example, popup ads and animated graphics are

features that are built with JavaScript. One of the biggest features of the JavaScript is that it can respond to the browser events like mouse clicks and keyboard actions. Program 3 presents the syntax of the JavaScript language.

```

1   const hideButton = document.getElementById('hide');
2   const elementToHide = document.getElementById('paragraph');
3
4   hideButton.addEventListener('click', () => {
5       elementToHide.style.display = "none";
6   });

```

Program 3. Example of the syntax of the JavaScript language.

The example JavaScript file adds interactivity to example web page presented in subsection 2.3.1 in program 1. In the example, event listener is added to *hide* button. When user of the page clicks this button, the paragraph element's style is changed, and paragraph disappears from the view. Figure 7 presents web page after *hide* button is clicked. It shows that the paragraph element has disappeared from the web page.

My Example Site

[Navigation 1](#)

[Navigation 2](#)

[Navigation 3](#)

Figure 7. The example web page after *hide* button is clicked.

The example JavaScript file shown in program 3 does not affect to other elements of the HTML document. It only modifies the visibility of the selected text.

2.3.4 TypeScript

TypeScript is created by Microsoft, and it was released in 2012. It is often called a superset of JavaScript. TypeScript is a programming language that includes all the existing features and syntax of JavaScript and a new TypeScript specific syntax for using types. It allows developers to add types for data and reports when the types are set incorrectly. [12, ch. 1] For example, if the type of the variable is string and the developer tries to assign a new value which is a number to the variable, TypeScript detects it and throws an error. JavaScript does not have this feature and any type of new value can be assigned to the variable. This can cause, for example, that string operations are executed in a variable that is type of number. Also, without types it can be difficult to keep track of the different data types in large projects. When TypeScript is compiled, it will deliver a

JavaScript file that can run in any browser or runtime environment that is capable of executing JavaScript [13, ch. 2]. Program 4 presents the syntax of the TypeScript language compared to JavaScript syntax.

```
1 // This is TypeScript
2 let message: string = "Hello!";
3 console.log(message);
4
5 // This is JavaScript
6 let message = "Hello!";
7 console.log(message);
```

Program 4. Example of the TypeScript syntax compared to JavaScript syntax.

Program 4 creates a variable *message* and then logs it in the console. Both the TypeScript and JavaScript examples work the same way, and the only difference is that in the TypeScript example, the type is defined for the *message* variable. When compiled, the TypeScript example generates to JavaScript example.

2.3.5 Vue.js

Vue.js is a minimal frontend framework for creating web applications [13, ch. 1]. It is a JavaScript framework, and it builds on top of three main web technologies, including HTML, CSS, and JavaScript. Vue.js is component-based programming model, and it has two core features: declarative rendering and reactivity. Declarative rendering means that Vue.js uses template syntax that extends HTML. It allows developers to declaratively describe HTML output based on JavaScript state. Reactivity means that Vue.js keeps track of state changes and updates the DOM (Document Object Model) of the web site when changes appear. [14] In Sandvik, the used version is Vue.js 3, and it is also the current version of Vue.js. Program 5 presents a simple example of Vue.js web application and the syntax of Vue.js.

```
1   <template>
2     <h1>{{ helloMessage }}</h1>
3   </template>
4
5   <script>
6     export default {
7       data() {
8         return {
9           helloMessage: 'Hello!'
10        }
11      }
12    }
13  </script>
14
15  <style>
16    h1 {
17      color: red;
18    }
19  </style>
```

Program 5. Example of the Vue.js syntax.

Program 5 produces a simple web application where data property with value 'Hello!' is shown on the web page. The message is styled with CSS. Vue.js files usually contain *template*, *script* and *style* sections as shown in program 5. Vue.js templates are valid HTML; script section contains the data and functionality of the component, or the view and style section contain CSS formatting of the component or view.

One of the core concepts in Vue.js is components. Vue.js applications usually consist of several, even tens of components. In Vue.js, components split the UI into independent and reusable parts [14]. Vue.js components form a tree-like structure, such as HTML elements on HTML document introduced in subsection 2.3.1. Every component encapsulates custom content and logic [14].

Figure 8 presents a diagram of the component tree. The diagram illustrates the tree structure in the Vue.js application that contains multiple components.

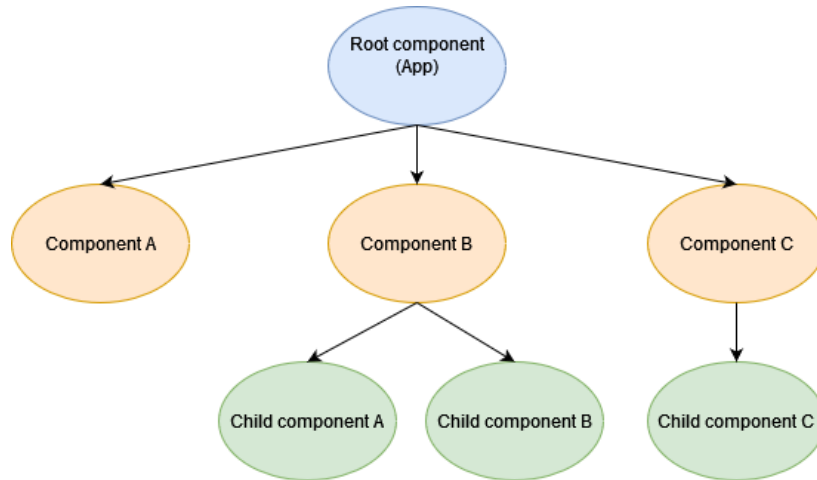


Figure 8. The structure of the Vue.js application containing multiple components.

Like the HTML document structure, the structure of the Vue.js application can be described like a family tree. The root component of the application is the ancestor of all components. Components that are parallel are sibling components. In figure 8, component B is a parent of child components A and B. Parent components use the child components by importing them and registering them in to use on the script section of the component.

2.4 Requirements for web applications in mining machine

This section presents the software requirements for mining machines' web UIs and their implementations. For requirements collection, two Sandvik employees were interviewed. The first interviewee was Petri Nurminen, Subject Expert Software Engineer, and the second interviewee was Jouni Lindgren, Systems Engineering Manager. Tables 2, 3, and 4 have been compiled based on the interviews, and they present found requirements and more detailed descriptions of the requirements. The requirements are divided into three categories: design constraints, and customer and cybersecurity requirements. Table 2 presents design constraints for web applications in Sandvik's underground mining machines. Design constraints refer to the limitations imposed on the design of a system or the development process that must be met to comply with technical, contractual, or business obligations [15, ch. 17].

Table 2. Design constraints for web applications in mining machine [16][17].

Requirement	Description
Supported by the browser	It is not necessarily possible to use the latest browsers on the mining machine, which means that the latest web technologies can not be used. This means that for example latest version of the JavaScript can not be used in development of the web applications.
The development process must be efficient and well-planned	As a result, the software architecture is such that the third-party resources can be used. Third party components can be integrated into Sandvik's own web components and applications.
Reusability of web components	Uniform Sandvik look and feel is wanted for all products and that can be partly achieved with reusable components. The look and feel of software refer to the impression that a user gets from the appearance and functionality of a program's user interface [18]. Also, components should be available to other developers.
Maintainability of the software	The life cycles of Sandvik products are long and that is why the code should be written a certain agreed way to achieve maintainability.
The used technology should not stand out	Qt and web technologies will live side by side for a long time when the transition towards the use of web technologies is made. It should not be possible to distinguish whether the UI is made with web technologies or Qt. Both Qt and web views must have the same Sandvik look.

Reusability of individual frontends	The same frontends can be used in several different product variants. For example, the same frontend can be used in both diesel and battery drill rig variants.
-------------------------------------	---

Table 3 presents customer requirements for web applications in Sandvik's underground mining machines. Customer requirements are specifications or features of a product which customers consider necessary [15, ch. 1].

Table 3. *Customer requirements for web applications in mining machine [16][17].*

Requirement	Description
Possibility to use different screen sizes	As a result, web views in mining machine should be responsive. This means that displays should be scalable. This imposes fewer requirements for used hardware.
Several views open	The overall design should consider that several views could be open at the same time. For example, the operator of the machine and the maintenance personnel can operate at the same time on different screens.
The order of operations	The user experience should be such that it guides the user in which order to perform different tasks. As a result, although the user interfaces are independent from each other, it should be possible to limit the views depending on each other. For example, if calibration of the drilling is in progress, drilling can not be started from different screen. It means that the screens must be aware of each other even though they are independent entities.
Screen specific localizations	Localizations are determined by where the machine is used and who is using the

	<p>machine. As a result, localization is implemented in web UI.</p>
<p>Screen specific units</p>	<p>Units are determined by where the machine is used and who is using the machine. This leads to the need to make a unit conversion in the web UI. For example, if the operator has chosen imperial measurement units, the UI converts metric units to imperial units.</p>
<p>UI works differently at different user levels</p>	<p>Depending on the user and the place of the use of the machine, different additional features should be available in the user interface. Authentication can not only be responsibility of the UI, because the UI can be bypassed if the user knows how to do it. As a result, user management is implemented in the UI and in the backend.</p>
<p>3D views and graphics</p>	<p>3D graphics should be implemented with web technologies, since open-source Qt does not offer tools for 3D. 3D models provide different viewing angles for the user. For example, it should be possible to present point clouds in 3D.</p>

Table 4 presents cybersecurity requirement for web applications in Sandvik's underground mining machines. Cyber security requirements are specifications for cyber security set forth for example by some organization, in this case by Sandvik.

Table 4. *Cyber security requirement for web applications in mining machine. [16]*

Requirement	Description
Can not be misused	The system shall remain safe and should be protected against external attacks. As a result, connections must be protected and encrypted.

Nurminen [16] and Lindgren [17] had a different approach to the requirements, Nurminen approached the requirements from a software perspective and Lindgren from a customer-oriented perspective. The found requirements highlight specific requirements for the drill rigs, which are related to, for example, the long-life cycle and high safety requirements of the machines.

3. DESIGN PATTERNS

Design patterns were first introduced by an architect named Christopher Alexander. He studied design issues by making observations of building and towns, among other things. [19, ch. 1] He discovered that, for certain architectural creations, good constructs had things in common with each other. Alexander looked at the structures that solve similar problems and discovered that he could see similarities between designs that were high quality. He called these similarities patterns. [21, ch. 5] A pattern is described by Christopher Alexander as follows: “Each pattern describes a problem that occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.” [20] Alexander says that a pattern includes four items: the name of the pattern, the problem it solves, how the pattern could be accomplished and the constraints and forces we must consider in order to accomplish a pattern [21, ch. 5].

Alexander produced a pattern language with Sara Ishikawa and Murray Silverstein and in 1977 they published a paper called “A Pattern Language”. Later, they released a complete hardcover book called “A Pattern Language: Towns, Buildings, Construction”. [19, ch. 1]

Approximately three decades ago, software engineers started incorporating the concepts that Alexander had described in his initial documentation on design patterns [19, ch. 1]. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides wrote a book called “Design Patterns: Elements of Reusable Object-Oriented Software” in 1995 [19, ch. 1] [22, ch. 1]. The book was inspired by the work of Christopher Alexander among other authors [22, ch. 1].

In software engineering, a design pattern is a generic solution that can be reused to solve common problems that occur in software design [19, ch. 1]. The purpose of design patterns is to give guidance on how to solve common problems. Although the design pattern may not be a precise solution for the specific problem at hand, it serves as a guide and provides suggestions to facilitate the implementation of solution. [22, ch. 1]

Using design patterns have multiple advantages for software developers because patterns are proven solutions, reusable and often expressive. Patterns provide generalized solutions; help avoid repetition and that way decrease the overall code file-size footprint and help to prevent minor issues that could potentially lead to major problems by encouraging the writing of structured and organized code. Additionally, patterns can help with

communication with other developers. Developers can reference the pattern while communicating with their colleagues. [19, ch. 1]

Before publishing the complete pattern, it should pass some “tests”. To design pattern to be interpreted as “good”, the following properties should be fulfilled:

- A design pattern should solve a particular problem. It means that the pattern should not capture just strategies, it needs to capture solutions.
- The design pattern should not offer an obvious solution and the most valuable design patterns typically provide solutions to issues indirectly.
- The design patterns should be proven to function as described so that their use can be seriously considered.
- The design pattern should describe a relationship. It means that the pattern’s description must detail the system’s architecture and mechanisms, which elucidate how it relates to the code. [19, ch. 2]

To be considered valid, a design pattern must fulfill one additional requirement. A design pattern should display some recurring phenomena. This can often be qualified with the help of the “rule of three”. The rule of three is used to show recurrence with three questions: What are the criteria for determining the success of a pattern? What factors contribute to pattern’s success? Does the design have a wider purpose of use so that it can be considered a pattern? [19, ch. 2]

Patterns are beneficial as they assist in ensuring that all developers within an organization are aligned when designing or maintaining solutions. [19, ch 1] Patterns are also valuable because they offer a general-purpose solution to a specific, frequently occurring problem and prevent the implementation of several different solutions to the same problem. A well-documented pattern can save time and extra work.

3.1 The structure of a design pattern

The structure of the design pattern includes multiple elements. A design pattern can have the following elements:

- **Name.** A unique name that represents the purpose of the pattern.
- **Description.** A brief description of what the pattern helps achieve.
- **Context outline.** The pattern's context is outlined, detailing the scenarios in which it can effectively meet the requirements of its users.
- **Problem statement.** A description of the issue that needs to be solved.

- **Design.** An explanation of the pattern's design approach.
- **Implementation.** A guidance for developers of how to implement the pattern.
- **Illustrations.** Visual representations.
- **Examples.** Examples illustrating the pattern's implementation in its simplest form.
- **Corequisites.** Which additional patterns may be necessary to assist in utilizing the pattern?
- **Relations.** What pattern does this pattern resemble?
- **Known usage.** Is the pattern being used elsewhere?
- **Discussions.** Author's perspective on the advantages of the pattern. [19, ch. 3]

Of the listed elements, the first five are the most important. A good pattern should also ideally provide reference material for users of the pattern and evidence of why the pattern is necessary. [19, ch. 3]

3.2 Categories of design patterns

In object-oriented programming, design patterns can be separated into three main categories based on the type of problem the pattern solves. These three categories are creational design patterns, structural design patterns and behavioral design patterns. [19, ch. 6]

Creational design patterns provide object creation mechanisms. Creational design patterns involve the creation of objects in a way that is appropriate for the given scenario. The basic approach to object creation could result in design problems or added complexity to the design. Creational design patterns aim to solve this problem by controlling the object creation process. [19, ch. 6]

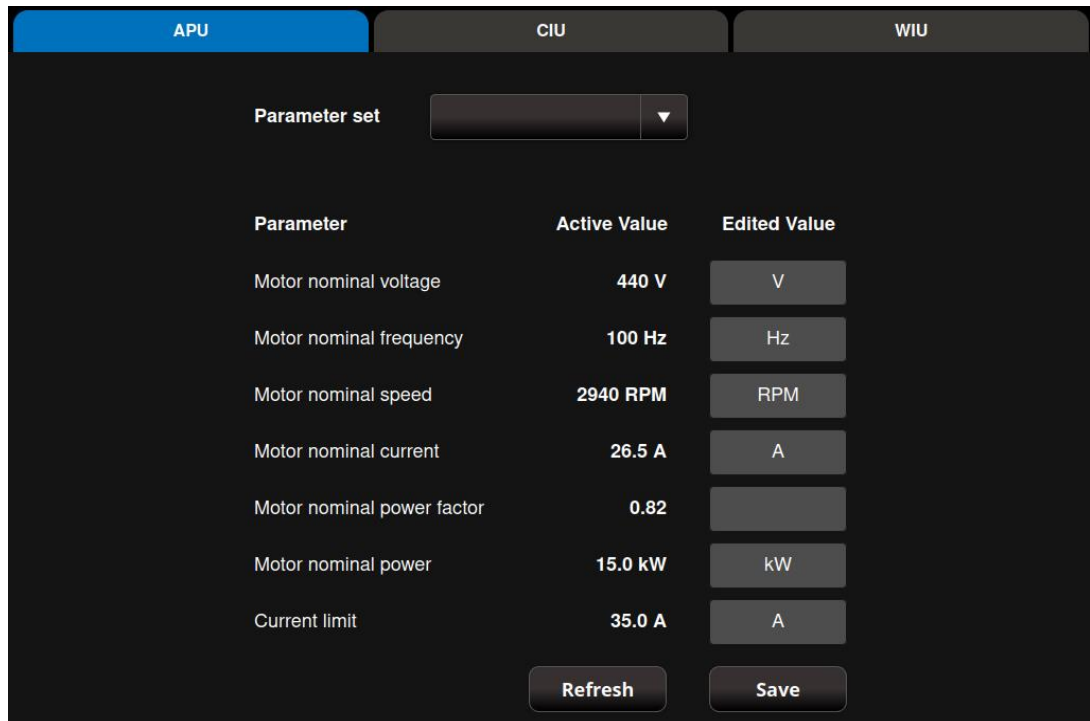
Structural design patterns explain how to compose objects and classes and typically these patterns define straightforward techniques for establishing connections between different objects and classes. These patterns help to ensure that the entire structure of the system does not have to change if one part of the system changes. Additionally, they aid in adapting parts of the system to a specific function for which they were previously unsuitable. [19, ch. 6]

Behavioral design patterns focus on the communication between different objects in the system and the way they operate together. They identify communication patterns between objects and offer solutions that share the communication responsibility between different objects, which increases the flexibility of communication. [19, ch. 6]

3.3 Implementation of design patterns

To find design patterns for Sandvik's web applications, one new web application partly implemented for this work and several already existing web applications were studied. This subsection presents the new application that was implemented and the subsections 3.3.1, 3.3.2, and 3.3.3 present identified design patterns from Sandvik's web applications. The design patterns presented in this subsection include the given name of the pattern, problem to be solved, descriptions, illustrations, guidance how to implement the pattern, examples, and when the pattern should be used.

The name of the new web application implemented is Inverter Parameters App and it is used for maintenance purposes. The display was implemented for Sandvik DD422iE development drill rig, which is a battery-electric mining jumbo. In the view, user can set parameters for three different inverters and select a parameter set from ready-made parameter sets. Figure 9 presents the Inverter Parameters App.



The screenshot shows the 'APU' tab of the Inverter Parameters App. At the top, there are three tabs: 'APU' (selected), 'CIU', and 'WIU'. Below the tabs is a 'Parameter set' dropdown menu. The main content is a table with three columns: 'Parameter', 'Active Value', and 'Edited Value'. The table lists several parameters with their current values and input boxes for editing. At the bottom, there are 'Refresh' and 'Save' buttons.

Parameter	Active Value	Edited Value
Motor nominal voltage	440 V	V
Motor nominal frequency	100 Hz	Hz
Motor nominal speed	2940 RPM	RPM
Motor nominal current	26.5 A	A
Motor nominal power factor	0.82	
Motor nominal power	15.0 kW	kW
Current limit	35.0 A	A

Figure 9. Inverter Parameters App.

The view consists of three separate tabs, one for each inverter, which are named APU, CIU and WIU. Each tab has the same content. The view contains the names of the parameters, current values of parameters read from the inverter, input boxes for manual

editing of the values and a dropdown list containing ready-made sets. The user can edit all the values or just certain values and after the save button is clicked, new values are sent to inverter and active values are changed automatically. The user can also click the refresh button if needed, which reads active values from the inverter.

By examining Inverter Parameters App and old web applications, including, for example DrillConnect App which is for visualizing and interfacing different kinds of underground drill rig's drill data, three design patterns were selected. One of the chosen design patterns was found in the literature.

3.3.1 The Library pattern

The first identified design pattern is the Library pattern. It can be combined with requirements listed in subsection 2.4 and it solves the problem related to these requirements. It can be classified as a creational design pattern. The problem that the pattern solves is that if the same components and views are used in multiple different mining machines, the code may have to be copied to several different machine applications. For example, Inverter Parameter App is included in two different mining machine applications, without the Library pattern, two different applications containing the same code should be created. With the Library pattern code, does not need to be copied in every application: the code can be reused from library.

In the Library pattern, reusable components and views are stored in the common library. The Library pattern is related to the "reusability of web component" requirement presented in subsection 2.4. Its advantage is code reusability and its disadvantage is that changes made in the library for certain application can break other applications using the same library.

The library can be created using, for example, ViteJS which is a frontend build tool. Components and views can be created just as in any Vue.js application. In the library, components and views are exported and configured. After this, the build of the library will be created, and the library is shared in an NPM (Node Package Manager) package.

The library pattern can be used by installing the NPM package of the library and importing wanted components or views in the application. Program 6 presents how to start use library in the application's main file after the installation of the library is done with NPM.

```

1   import { createApp } from 'vue';
2   import App from './App.vue';
3
4   import UgComponents from '@sandvik-sica/ug-components';
5
6   createApp(App)
7     .use(UgComponents)
8     .mount('#app')
```

Program 6. *The main.ts file in the application.*

After the library is included in the main file of the application, the components from the library can be used by importing single components and views in the Vue.js application. Program 7 presents how to use a single component from the library in the Vue.js application. In this example, the view implemented for this work is imported and used from *ug-components* library.

```

1   <template>
2     <InverterParametersView />
3   </template>
4
5   <script setup lang="ts">
6     import { InverterParametersView } from
7       '@sandvik-sica/ug-components';
8   </script>
```

Program 7. *The usage of the components from the library.*

The library pattern should be used when views and components are used in multiple different projects. Multiple libraries, including commonly used views and components are already in use in Sandvik's web applications.

3.3.2 The ID pattern

The second identified design pattern is ID pattern. This pattern can be classified as a creational design pattern. In Sandvik, two different test automation frameworks are used for test automation: Robot Framework and Sandvik's own test automation framework. Both frameworks use Selenium for testing web applications. Selenium offers a variety of libraries and tools for testing web applications.

SeleniumLibrary in Robot Framework supports finding elements based on different ways such as the HTML element id, XPath or CSS selectors. Robot Framework uses keywords for testing. Keywords are single test steps and in SeleniumLibrary, all keywords support

finding elements based on the id attribute of the HTML element. [23] The problem is that if the HTML does not have an id, it quickly becomes complex to find an element using attributes of HTML element or CSS selectors. For example, often multiple HTML elements have the same class attribute. In complex cases, XPath (XML Path Language) expressions can be used, but often they can also get complex, and hence, maintaining test cases becomes difficult [23]. In some cases, without element id's, the XPath expression is the only way to locate an element. The problem can be approached with an example. The code line below presents an XPath locator to element that can not be found with CSS selectors or HTML attributes and the element does not have an id.

```
xpath=/html/body/div/div/div[2]/div/div/div/div[1]/div/div[2]/span
```

XPath uses path syntax to identify elements in the HTML document. The element that the example locates is deeply nested element and the XPath locator shown is complex. The same element located with the element's id with the XPath expression is shown in the code line below.

```
xpath=//*[@id="percussion-pressure-gauge"]
```

In the ID pattern, every component created in Vue.js is given a unique id. When the component is created, the mandatory *id* prop is declared to component. In Vue.js props are attributes that can be registered on any component and with props a parent component can pass data to child components [14]. Program 8 presents a simple Vue.js component which have a mandatory *id* prop.

```

1   <template>
2     <button :id="id">Click me!</button>
3   </template>
4
5   <script>
6     export default {
7       props: {
8         id: {
9           type: String,
10          required: true
11        }
12      }
13    }
14  </script>
```

Program 8. A simple Vue.js component with a mandatory *id*.

When the component is used, the *id* prop needs to be given to component. Program 9 presents how to pass an *id* prop to component.

```
1   <template>
2     <Button id="button"></Button>
3   </template>
4
5   <script>
6     import Button from './Button.vue';
7
8     export default {
9       components: {
10        Button
11      }
12    }
13  </script>
```

Program 9. *The usage of the component with mandatory prop.*

The component presented in program 9 can be tested with id with test automation frameworks. The ID pattern does not only solve the complex syntax problem, locating elements by id is fast for browsers and it makes testing as efficient as possible. This pattern should be used for every component in the Sandvik's web applications to ensure efficient web testing.

3.3.3 Pinia State Management Pattern

This section presents the already existing design pattern found from literature that can be used and is already used in the web applications of Sandvik's underground drill rigs. The third design pattern presented is Pinia State Management pattern. State Management Pattern is a behavioral design pattern. State management is an important part of the web applications: all complex applications need state management. The problem that the pattern solves can be approached with a simple example.

Program 10 presents simple Vue.js app. It implements a simple counter functionality.

```

1   <!-- View -->
2   <template>
3     <div>{{ count }}</div>
4     <button @click="increment">Add</button>
5   </template>
6
7   <script>
8   export default {
9     // state
10    data() {
11      return {
12        count: 0
13      }
14    },
15    // actions
16    methods: {
17      increment () {
18        this.count++;
19      }
20    }
21  }
22  </script>

```

Program 10. Simple Vue counter app. Adapted from [21].

The app in program 10 contains the following parts: the state that contains the application data and its status, the view that shows the state and the actions that manage how the state changes in reaction to user inputs from the view [24]. The example shown is simple and state management works well in an example for very simple one component application.

The problem is that state management quickly becomes more complex when application has multiple components that share a common state. The first problem is that multiple components can depend on the same piece of state and the second problem is that actions from different components may need to mutate the same piece of state. [24] In a simple situation when there are not many components, the first problem can be solved with props. However, props do not work in the case of sibling components and passing data through props to deeply nested components make the state quickly unmanageable. The second problem can be solved with emitting events, again, in a simple situation. In Vue.js, component can emit custom events in template expressions meaning that in a child component event can be triggered and data can be passed up to the parent component [14].

Figure 10 presents a diagram of a more complex Vue.js application that contains multiple components that depend on the same state.

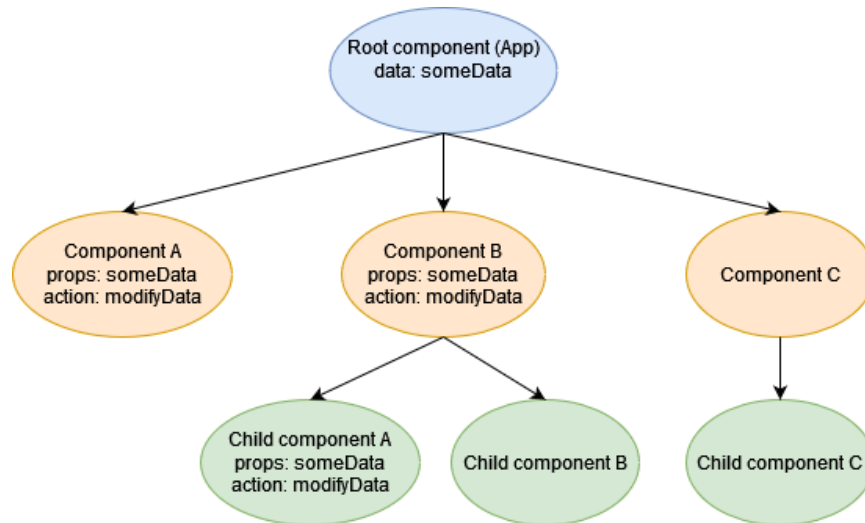


Figure 10. State management with props.

In figure 10, multiple components use the same data, and the data is passed down to child components with props. Multiple components can also modify the same data with actions. The situation in the example is still simple, but the state management is already considerably complicated.

Problems presented can be solved by extracting the shared state out of the components and manage state in a global store [24]. Pinia is one of the two libraries and a State Management Pattern in Vue to handle the state of the application. Pinia serves a centralized store for all components in a Vue.js application. [25] It means that Pinia allows to share a state across components and pages in Vue.js. In Sandvik, Pinia is used in multiple web applications.

Pinia store consists of the following parts: the state, the actions, and the getters. The state holds the data of the store, the actions are used to mutate the state and the getters are used to access the data of the store. [25] Next, it will be presented how a store can be created and therefore also how to implement the State Management pattern.

Firstly, Pinia needs to be installed. Multiple different package managers can be used, for example, NPM (Node Package Manager). Secondly, a Pinia store needs to be created. It is highly recommended that stores are kept in folder named “store” or “stores”. Program 11 presents a simple store named *somedata*.

```

1  import { defineStore } from 'pinia';
2
3  export const useSomeDataStore = defineStore('somedata', {
4    state: () => {
5      return {
6        someData: 0
7      }
8    },
9    actions: {
10     addone() {
11       this.someData++;
12     }
13   },
14   getters: {
15     getSomeData(state) {
16       return state.someData;
17     }
18   }
19 })

```

Program 11. A simple Pinia store in file *somedata.ts*.

After the installation and creating a store, the store needs to be declared in the application. Program 12 presents how to declare the store in the application's main file.

```

1  import App from './App.vue';
2  import { createApp } from 'vue';
3  import { createPinia } from 'pinia';
4
5  const pinia = createPinia();
6  const app = createApp(App);
7
8  app.mount('#app');
9  app.use(pinia);

```

Program 12. The *main.ts* file in the application.

Now the state and actions can be accessed using *useSomeDataStore* call at any component in the application. All the components in the application can use the state and mutate it with the actions regardless of where they are located in the component hierarchy.

As said before, Pinia is one of the two libraries used to state management in Vue.js. Another library and State Management pattern is Vuex. Like Pinia, Vuex also offers a centralized store for state management. However, Pinia has been chosen as the State Management pattern used in Sandvik's web applications. Pinia provides a simpler API than Vuex and it supports usage of Composition API [25]. Most important feature in Pinia compared to Vuex is that it has type inference support when used with TypeScript [24]. Pinia, unlike Vuex, also offers an opportunity to create many instances off the same store, which was found to be useful in web applications in Sandvik. This feature enabled,

for example, the management of data across the three distinct tabs displayed in this project. Using a distinct instance of the same store for each tab prevented the creation of three nearly identical stores. With Vuex, this would not have been possible.

The state management pattern should be used when application's size is medium or large and there is same data that is used in multiple components. A store should contain data that can be accessed from everywhere in the application.

4. SUMMARY

The target of this work was to identify design patterns for Sandvik drilling mining machines' web applications. Drill rigs' control systems and software architecture was also studied. Additionally, the work gathered software requirements for the web applications of drill rigs. Also, for this work, a new view was implemented with web technologies for Sandvik battery-electric mining jumbo.

This work found that drill rigs set application-specific software requirements for software development. Special requirements arose from, for example, high safety demands and working areas of the machines. Additionally, in the future when different technologies develop further new requirements arise. Web technologies achieve a lot that Qt can not such as scalable views, which enable the use of UIs outside of the machine itself, for example, in mobile devices or in any device that contains modern web browser. Additionally, with web technologies, 3D graphics can be implemented. In the future, there is need to observe, for example, different drilling point clouds from different angles and tunnel depth with the help of UI.

During the work and especially during development of new web application, several targets for development were discovered to unify web development. Among other things, the importance of communication and documentation was highlighted. One solution to unification of web development is using and documenting design patterns. With the help of design patterns, it is possible to transfer data from best practices inside the organization, also making communication more fluent. Design patterns can also help new employees at the beginning and prevent design errors.

Numerous design patterns can be found from literature and selecting patterns for this work was challenging. Selection of patterns was affected by new web application developed and the selected patterns are used in the new web application. Only one pattern was selected from literature.

As a result of this work, three various design patterns were selected. First pattern is related to reusability of various web components and their storing in a way, that they are easily accessible from the component library. Multiple component libraries are already in use in Sandvik, and by utilizing them, the target is to reduce the need to copy code to multiple different applications. This pattern can also be connected to discovered requirements because it offers opportunity to use same components in different applications. Second pattern is related to test automation and to enhance web testing by unique ids

given to elements and components in web applications. In web testing, locating elements by id is the fastest way for web browsers. Third selected pattern is related to state management of large web applications. Usually, in large web applications, application contains large amount of same data, that multiple different parts of the application use. Without this pattern, state management of large applications would be extremely complex.

In the future, it is recommended to continue searching and documenting new design patterns and best practices. With these recommendations more harmonized way of working can be achieved in web development.

SOURCES

- [1] J. I. Hong, J. A. Landay, D. K. Van Duyne, The Design of Sites: Patterns for Creating Winning Web Sites, Second Edition. Pearson, 2006, 1024 pp.
- [2] Underground drill rigs and bolters, Sandvik, 2023. Available (referenced 1.3.2023): <https://www.rocktechnology.sandvik/en/products/underground-drill-rigs-and-bolters/>
- [3] MediaBank, Sandvik, 2023. Limited availability (referenced 1.3.2023).
- [4] J. Viitala, SICA Training: SICA/MCC, pdf document, Sandvik, 2022. Limited availability (referenced 1.3.2023).
- [5] N. Dey, Cross-Platform Development with Qt 6 and Modern C++, Packt Publishing, 2021, 442 pp.
- [6] V.-P. Jaakkola, Control system architecture evolution, pdf document, Sandvik. Limited availability (referenced 1.3.2023).
- [7] Knowledge Box, Sandvik, 2023. Limited availability (referenced 9.5.2023).
- [8] K. Taylor, B.E. Smith, Getting a Web Development Job For Dummies. For Dummies, 2015, 312 pp.
- [9] J. Krause, Introducing Web Development, Apress, 2016, 89 pp.
- [10] B. Frain, Responsive Web Design with HTML5 and CSS, Fourth Edition. Packt Publishing, 2022, 498 pp.
- [11] L.L. Svekis, M. van Putten, R. Percival, JavaScript from Beginner to Professional. Packt Publishing, 2021, 546 pp.
- [12] J. Goldberg, Learning TypeScript. O'Reilly Media, Inc, 2022, 318 pp.
- [13] H.R. Ribeiro, Vue.js 3 Cookbook. Packt Publishing, 2020, 562 pp.
- [14] Guide, Vue.js. Available (referenced 4.4.2023): <https://vuejs.org/>
- [15] D. Leffingwell. Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise. Addison-Wesley Professional, 2010, 562 pp.
- [16] P.Nurminen, Subject Expert Software Engineer, Sandvik Mining and Construction Oy, Tampere. Interview 8.3.2023.
- [17] J.Lindgren, Systems Engineering Manager, Sandvik Mining and Construction Oy, Tampere. Interview 17.3.2023.
- [18] Tietotekniikan termitalkoot, Sanastokeskus ry, 2001. Available (referenced 10.5.2023): <https://sanastokeskus.fi/tsk/fi/termitalkoot/haku-266.html>

- [19] A. Osmani, Learning JavaScript Design Patterns, Second Edition. O’reilly Media, Inc, 2023, 286 pp.
- [20] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fikshdal-King, S. Angel, A Pattern Language: Towns, Buildings, Construction. Oxford University Press, 1977, 1071 pp.
- [21] A. Shalloway, J.R. Trott, Design Patterns Explained: A New Perspective on Object-Oriented Design, Second Edition. Addison-Wesley Professional, 2004, 480 pp.
- [22] S. Timms, Mastering JavaScript Design Patterns. Packt Publishing, 2014, 290 pp.
- [23] SeleniumLibrary, Robot Framework. Available (referenced 4.5.2023): <https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html>
- [24] Guide, Vuex. Available (referenced 23.4.2023): <https://vuex.vuejs.org/>
- [25] Guide, Pinia. Available (referenced 23.4.2023): <https://pinia.vuejs.org/>