Tran Le

SALES PORTAL

A React Responsive Web Application Managing Offers and

Configurations of Projects

Technology

2022

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Tran Le
Title	Sales Portal: A React Responsive Web Application Managing Projects for Customer Companies
Year	2022
Language	English
Pages	32
Name of Supervisor	Timo Kankaanpää

This thesis project is a product of the team in Ekseli Oy. The objective of this thesis was to design and implement an application that helps companies to manage offers and configurations of every project.

The project is a client solution with database through simple APIs. The client side was built using React with TypeScript and MaterialUI on the front end, and Solid PODs as the decentralized database. The application was tested using a combination of Jest and Enzyme.

The application facilitates the sales management of projects for companies. Maintaining a clear and constant communication with the customers plays a crucial role in performing a successful sale.

Keywords Sales Portal, React, Solid, Jest, Enzyme

TABLE OF CONTENT

ABBREVIATIONS

LTS	Long term support
VS	Visual Studio
UI	User Interface
URL	Uniform Resource Locator
JSX	JavaScript XML
XML	Extensible Markup Language
HTML	Hypertext Markup Language
API	Application Programming Interface
JS	JavaScript
TS	Typescript
CSS	Cascading Style Sheets
IDE	Integrated Development Environment
SP	Sales Portal
DOM	Document Object Modal
NPM	Node Package Manager
DP	Delivery Portal
CRUD	Create – Read – Update – Delete

LIST OF FIGURES AND TABLES

1. INTRODUCTION

1.1. Sales Portal Purpose

A sales portal (SP) is a tool for managing offers and configurations of projects. It presents both sales managers and customers with the same views on the data. This significantly helps with reducing unclarity and miscommunication between clients and the companies' team. However, the write access is limited to the sales lead to prevent customers from being able to update whenever.

A sales portal is one of the best options for managing projects that have many different types of files such as images, videos, and documentation thanks to its affordable cost and ease of usage. If using AWS or Google Cloud, the users need to equip a deep knowledge or expertise related to the field, and with this SP solution using Solid PODs, everything becomes lightweight and effective

1.2. Objectives

This study demonstrates the full development process and architecture of a web application using React. The goal of this study is to document the application structure and the implementation of SP, covering its features and development choices.

The study aims at enthusiastic frontend developers who are seeking initial guidelines on an application development process in professional environment.

1.3. Implementation Plan

Since the need for an SP arose later, it was built into the same Git repository as the Delivery Portal (DP). This significantly improved the implementation timeline thanks to the ready-made components and assets. In addition, the project can take advantage of the existing authorization feature used for DP.

2. TECHNOLOGIES AND TOOLS

2.1. Technologies Used

In the era of technology and advancement, countless tech start-ups are being established to create a solution for all possible problems or improve quality of life for other industries. Therefore, under such competition, it is of the utmost importance to make the correct decisions, not only in business and marketing, but also in technology. When it comes to front-end web development, the most obvious choice for any startup is React, thanks to its enormous amount of enthusiast professionals and community supports.

On top of the React library, due to the complexity of any professional product, intensive testing is needed through the means of automatic test. One of the most powerful and highly used tools for writing tests for React is the combination of Jest and Enzyme. They contain APIs for any possibly imagined test cases, be it either business logic or user interface.

2.2. JavaScript History

JavaScript, one of the most widely used language in the developers' community, was developed in 1995 by a Netscape developer named Brendan Eich. It was originally developed to solve the issue of a long loading time, when websites are growing in complexity and size, since all calculations were handled by the server. Its initial purpose was to run on the browser and handles simple tasks. As a result, even though originally named Mocha, its name was changed to LiveScript. However, just before release to captivate on the infamous Java language at the time, the name got changed one final time to JS. /3/

2.3. JavaScript Definition

JavaScript is a lightweight, just-in-time compiled programming language. It has first-class function, which means function would be assigned as a value to a variable. Besides being known as the scripting language for web pages, many other non-browser environments also use it, such as NodeJS, Apache CouchDB and Adobe

Acrobat. Some characteristics of JS are prototype-based, multi-paradigm, single-threaded, dynamic language, imperative and declarative. /3/

Within the JS ecosystem, there are many JS libraries that significantly boost the speed of building professional, feature-packed web pages. Therefore, JavaScript is used as the primary programming language for developing Sales Portal.

2.4. Pros and Cons of React

React is a JS library which was developed by a software engineer at Facebook named Jordan Walke. Initially, it was built as an internal tool for helping the development of the Facebook's applications. However, at JSConf2013, the team decided to make React open source, marking the start for a flourishing progress of one of the most beloved technologies for building web and mobile applications. /4/

React is a declarative, component-based library responsible for the application's view layer including how the application looks like and feels. In React, mostly JSX is used as a (stands for JavaScript XML) template which allows using HTML tags to render subcomponents and to process HTML syntax in JS code instead of separating the markup and logic in different files.

Thanks to the popularity of React among the developer's community, it contains an enormous number of plugins and libraries, ranging from tiny helper methods to full scale solution for scaling application. In addition, since React makes use of a concept called virtual DOM, updating a section of a full-page website is faster and more manageable. As a result, choosing React as the main library for the Sales Portal development is an obvious choice.

In some cases, the benefit of having many different libraries option can be a source of difficulties for new developers. By having many good options ready at a line of command, choosing the most suitable one for their specific application is difficult without a deep understanding of the advantages and disadvantages of each. /4/

2.5. TypeScript

Typescript, created by Microsoft, was formerly only used internally in the early 2010s, but it was then open sourced in 2012. Leading its development, Anders Hejlsberg has also been the lead of some other widely used programming languages, such as C# and Turbo Pascal. Since at its core, TS is JS with enforced type checking, its commonly known as the 'superset of JS or 'JS with types'. However, since the browser cannot compile TS, additional tools built into TS is needed to convert it into JS. /5/

With the continuous and fast paced growth in websites complexity, codebase gets harder to manage over time. Therefore, developers desire for a tool to help maintain integrity of the source code of the project to prevent any human errors. TS is one of the best options for this purpose thanks to its features. Since all functions and variables are precisely typed, the types can also act as built-in documentation for the entire application depending on what is acceptable. In addition, given that most IDEs have extensions that provide support for code autocompletion with TS, it strongly improves the quality of life of developers and ensure that no human errors, such as a typo, can occur. /5/

Since Sales Portal has a huge scaling potential, clear documentation and data types play a crucial role in ensuring the continuous and quick development of new features, while maintaining a high code quality.

2.6. Solid POD

Solid is a realization of the inventor, Sir Tim Berners-Lee, original vision of a decentralized exchange of private and public data on the Web. It provides a specification to allow people to securely store any of their data in stores called PODs. Any user can have multiple PODs, each being their own instance, having no correlation with other PODs. The PODs are under full control of the owner, especially in their access rules. /7/

Different from normal development stacks with a dedicated server and storage, Solid provide a simple database solution without the need for time spent of developing a full-scale server. Due to the requirement of maintaining the privacy of the customers' documents, Solid POD is the most suitable choices for Sales Portal development.

2.7. Jest & Enzyme

Jest is JS testing framework built with focus on simplicity and compatibility with different available technologies, such as React, Angular, and TS. Powerful as it is, it has built-in code coverage, mocking, and parallel tests running. Jest contains an approachable and familiar API to allow developers to quickly test their application. Even though Jest is not built specifically for testing a React application, its general-purpose usage is suitable for testing business logic and provide a perfect starting point for more explicit use case, such as React. /8/

As a result, Jest is often used in combination with another library specific tool, such as Enzyme for React. It can be used to test easily any React component output by simulating the DOM behaviors by allowing developers to interact with it using intuitive and flexible API. /9/

Thanks to being one of the most prominent options within the developer community, choosing Jest and Enzyme for setting up application's automated testing is easily agreed upon by the entire team.

2.8. Material UI

Among the web developers' community using React, there is no library more well-known than Material UI. It is one of the best component libraries, a library with many of the most common components of any websites, such as input, dropdown menu, button, with professional designs, available for React developers. Its development started in 2014 with the goal of unifying React and Material Design, a set of guidelines for designing eye-catching application, having rules on margin between components, font sizes, etc. /10/

Due to the limitation of the team size, and the limited number of designers within the team, using a quick but professional pre-designed components is important for prototyping a minimum viable product for pitching the application to potential customers.

2.9. SASS

Due to the existence of different browsers available on the market from different companies, each with their own implementation vision, details and rules, consistent and error proof styling of websites can be the most challenging aspect of web development. While some companies can live with the decision of only supporting some of the most popular browsers, such as Google Chrome, Mozilla Firefox, many large organizations need to constantly make their websites available to all customers. Therefore, preprocessors, such as SASS can be a much-needed help to every developer. Having been under development and support for over 15 years by the Core Team, SASS boasts more features and abilities than any other CSS extensions available. /11/

Since Sales Portal customers can have very different background, their choice of browsers can also vary greatly. As a result, supporting as many browsers option as possible is one of the most important requirements for SP.

2.10. Visual Studio Code

IDEs play an essential role in any developer's career. They are as important to programmers as pens are to writers. Even though there are many IDE options, each have their own advantages and disadvantages. Most IDE have any integrated tools to improve the quality of life and prevent human errors. This often creates a significant amount of overhead when running then. Having understood this issue, Microsoft created an IDE called Visual Studio Code or VS Code in short, using extensions to allow users to custom the IDE to their need, hence significantly improve the performance. /12/

Comparing to other IDEs, being lightweight makes VS Code one of the best options for any web development team, as each member can pick whatever extensions they personally need.

2.11. Yarn

Yarn was built to become a package manager, a tool for allowing any developer to share code with others from around the world. It allows developers to use community's solutions to different problems, making it easier to develop both small and scalable software. It offers a full-fledge management of project's dependencies, from adding, removing, to upgrading and downgrading packages using simple commands. /2/ Therefore, in order to manage the large number of dependencies used for Sales Portal development, Yarn is easily the best option available.

2.12. Git

When working in a team, typically each member is assigned different tasks, not overlapping with another to avoid duplicate work. However, all the work is required to be combined into a single working application. In addition, it is of the utmost importance for the team to be able to revert to an older version in case new changes render the application broken. As a result, a source control management tool is much needed. Among the different available tools, one that is most used is called Git. It works by having limitless number of local branches working independently of each other. Each branch is like having a snapshot of all the application files at a given state. Developers can then easily update the branch without worrying about affecting any other team members' work. / 13/

Thanks to the aforementioned benefits of git as a source control, the team makes use of git to be able to revert any broken code, or to keep track of all the changes made to SP.

3. SOFTWARE REQUIREMENTS

This chapter describes the characteristics that this product must have to meet the business and customers' needs. It gives a detailed description of a software system to be developed with its requirements based on the agreement between the user and the software developer.

The requirements are divided into two sections: functional, a description of the features and functions of the final product, and non-functional, describing the general characteristics of the system, such as the user interface, user experience.

Table 1 specifies the features as expected by the user. Table 2 specifies the non-functional requirements of the system as requested by the user.

Table 1. Functional Requirements

Reference	**Description**	**Priority**
F1	Fetching projects/offers/configs data	1
F2	Sort projects based on names	1
F3	Create and delete new projects/offers/configs	1
F4	Have rich text editor in the configuration section	1
F5	Allowing upload of images and documents	1
F6	Having image slideshow to automatically scroll through them	2
F7	Comments on each configuration	1

F8	Move configurations in any project's offer to Delivery Portal	1
F9	Ability to update uploaded files titles	1

Priority levels:

Must have–1

Recommended to have–2

Nice to have–3

Table 2. Non-functional characteristics

Characteristic	**Description**
Usability	Intuitive user interface to allow users to not have to think of how the application works.
Safety	Secure, transparent, and decentralized storage solution for every and only logged in users.
Aesthetic	The application styling maintains a simple, professional, and united styling.
Maintainability	Since the application has more scaling potential, code quality remains at a high standard to allow easy development in the future.

4. DESIGN STRUCTURE

For the purpose of efficient and seamlessly collaboration in a team, organized and logical structures are non-negligible. Enterprise level application often have a need to scale up constantly while keeping functional performance. Consequently, optimized structures of both folders and data prevent human errors and code entanglement.

4.1. Application Architecture

Application structure is one of the most fundamental building blocks of any software project since it can act as a simple documentation for the project by having logical layers of application. This allows the application to grow easily without constant clean up. In addition, a clear organization which also follows the industry standard can significantly ease the onboarding process of any newly hired developers.

Figure 1. Packages used in package.json

In all web projects, companies always have their own set of preset shortcuts that allow developers to quickly run, test, and deploy the application to production. In addition, since in the web industry, there are many features and frameworks created with neat solutions. They are publicly available under the form of packages that any projects can utilize as dependencies. All of those can be noticed in the *package.json* file, as presented in Figure 1.

Besides that, in order to help non-developers to have general ideas about the project such as how it works, how to run the application, and what the main features are, README.md plays a crucial role for that purpose.

In most cases, not only do development team members need to view the features of the application, but also the users. In addition, signaling the users about any important bug fixes can have a major impact on the user experience by increasing the transparency between the team and the users. Within the project code base, CHANGELOG.md file is responsible for storing all the news related to any and all releases, be it major or minor versions.

In *azure-pipelines.yml*, there are commands to trigger tests running before going to production. This helps to make sure all features behave as expected and avoid breaking any functions.

Lastly, the most important folder, being the folder that contains the source code, in any application structure is one called *src*. Under the source block, assets folder, including fonts and images, contributes to a good-looking presentation for the pages. The development team have to discuss during the meetings to decide which options could be used and added to assets folder. In addition, since development cycle is a repeated process of implementing, testing, and debugging application performance is valued more aesthetic. Therefore, lightweight images that are purely used as mockups during development are stored under folder called 'stubs' underneath the images.

Imagine a simple e-commerce website having independent components, such as search area, cart button, product list. Following the model, any websites is nothing more than a collection of carefully placed and designed elements. Therefore, in almost all web application source codes, there is a dedicated components folder. Within the folder, apart from user facing elements, such as inputs, carousels, there are also components that handle the backbone of websites. This includes routing between pages, fetching and formatting data. The folder structure is demonstrated in Figure 2.

Figure 2. Application Structure overall in SP

Every component always has its own folder under components folder in codebase, which could include the main implementation file *component_name.tsx*, and optional files for testing *component_name.test.tsx* and styling *component_name.scss*.

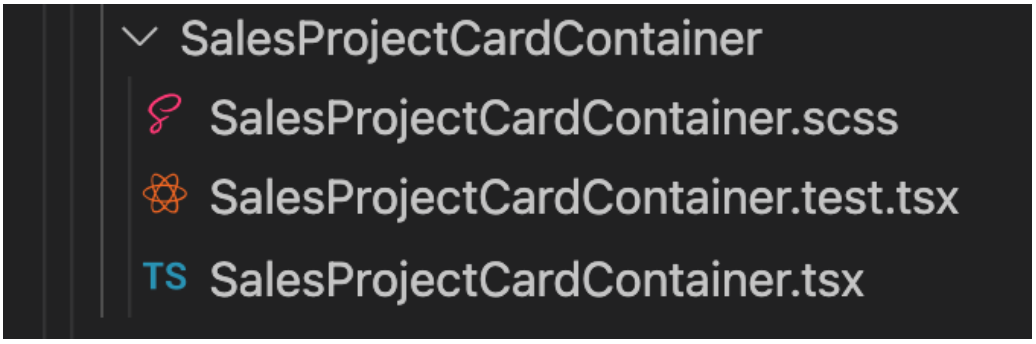

Figure 3. Files structure in component folder

4.2. Data Structure

With the help of TS, every data in the Sales Portal is clearly documented. Starting from the very top, since the application goal is to provide a platform for companies to manage sales of their projects, information about the customers, such as logos and responsible personnel, and projects, such as status and manager name, are the first few things data to store. This allows organizations to have an easier time keeping track of the thousands of ongoing projects.

```
type SalesProject = {
  projectName: string;
  customerName: string;
  managerName: string;
  projectStatus: number;
  customerLogo: string;
  projectImage: string;
};
```

Figure 1. Sales Project type entity

Any given projects can potentially be provided with several versions of offers. Therefore, keeping track of the name of the offer, valid date and status is important

to response accordingly. Since an offer status change drastically and frequently, storing the status variable as simple string is inefficient and can introduce errors.

```
type Offer = {
  name: string;
  validFrom: Date;
  validUntil: Date;
  status: number;
};
```

Figure 2. Offer type entity in Sales Project

Offers from customers are usually made with a variety of configurations. For example, an offer can contain requirements for a nails production machinery, a screws production machinery, and a hooks production machinery. Each machinery has its own set of tools, and rulesets. In addition, configurations can be a convenient and effective means of communication between organizations and their customers. Other than the ability to leave notes, businesses can add images and attachments on the state of the projects. In order to save the images and attachments to load back at a later time, they are quick access through the browser. In the special case of attachments, since the files type can be varied greatly, a separate field for the extension is needed.

```
type Configuration = {
  id: number;
  name: string;
  description: string;
  images?: Images[];
  notes?: Stream[];
  attachments?: Attachments[];
  status: number;
};

export type Images = {
  id: number;
  image?: File;
  blobURL?: string;
  imageURL: string;
};

export type Attachments = {
  dateAdded: Date;
  file?: File;
  fileURL?: string;
  fileTitle: string;
  fileName: string;
  type: number;
};
```

Figure 3. Configuration type entity with types of Images and Attachments

4.3. Module Design

Before adding offers or configurations in a certain project, that project needs to be available. If the project has not been created yet, it can be done by clicking "Add project" button and filling in the needed information. Besides that, the users also have other actions such as Edit or Delete a SP project. The implementation of creating a project is captured in the Figure 6.

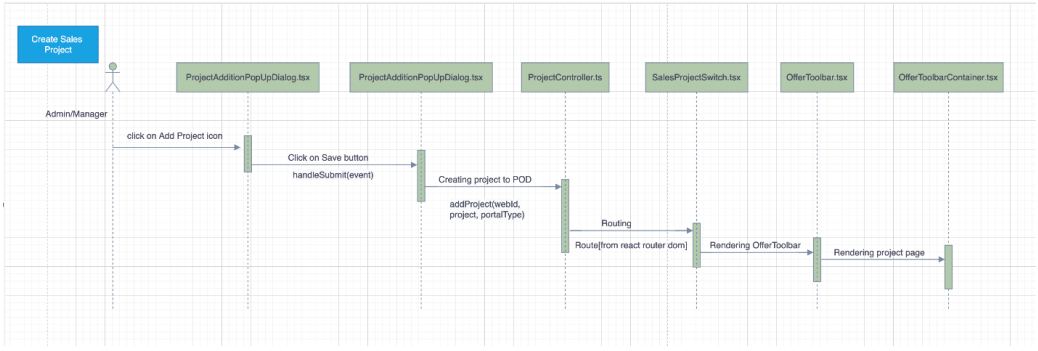

Figure 4. Sequence of interaction in adding SP Project

A SP project could contain many versions of offers. In addition to saving the new offer to the Solid POD for later use, the created offer is also saved into the browser's local storage. This serves the purpose of being able to load back the previous section's offer. Any data saved into the local storage of the browser using the website URL persist until only when the browser application is shut down. If that condition is still met when the user reopen the SP, there would be logics to check the storage and act accordingly.

At the initial phase of the project implementation, after a new offer is created, no matter the result, the website is reloaded to get the new offer info. Even though this action does not provide optimal user experience, it helps significantly in speeding up the development process.

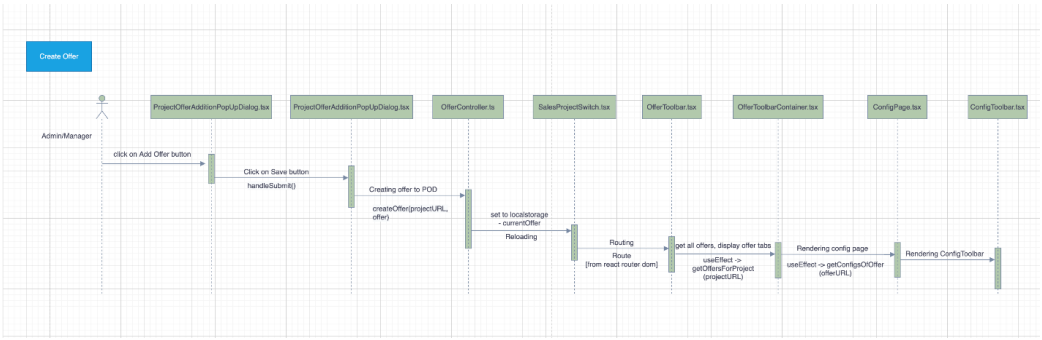

Figure 5. Sequence of interaction in creating offer in any Sales Project

In general, the user and application flow for creating offers and configs are similar. They both save their data to the Solid PODs, for later use, and local storage, for user experience improvement. In addition, after a new offer or config is added, the website is located to display the updated data.

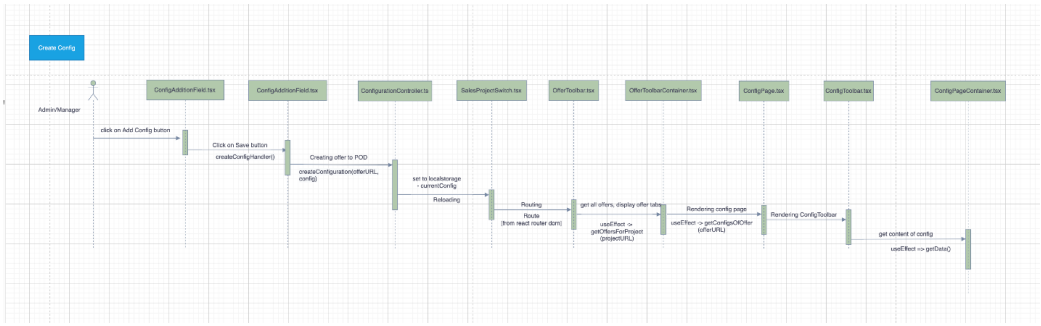

Figure 6. Sequence of interaction in creating configuration in any Offer

As configurations get more update and worked on, it would eventually reach a steady state. When the situation applies, the offer is ready to be packed and delivered. To do this, a "Move to Delivery" button is provided to move the project to the DP, where all things related to deliver a project is offered as help. Once the move succeeds, the project would be automatically deactivated on the SP. Even though the project is disabled, all corresponding files are preserved in case an "undo" action is needed.

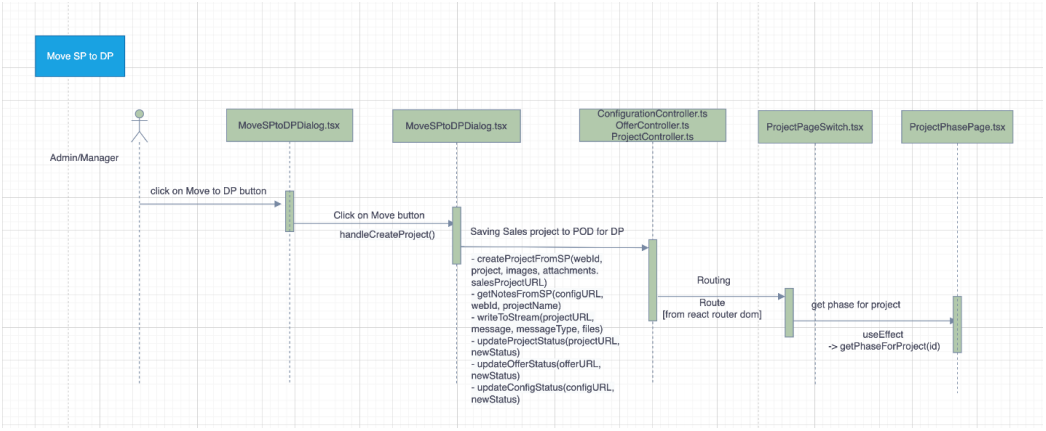

Figure 7. Sequence of interaction in moving any configuration in SP to DP

As mentioned above, when an offer config is ready to be delivered, it is then moved to the DP and deactivate from the SP. However, similar to how testing is used in development to prevent human errors, a button is also placed to allow users to possibly reactivate a disabled project and unpack all offers and configurations.

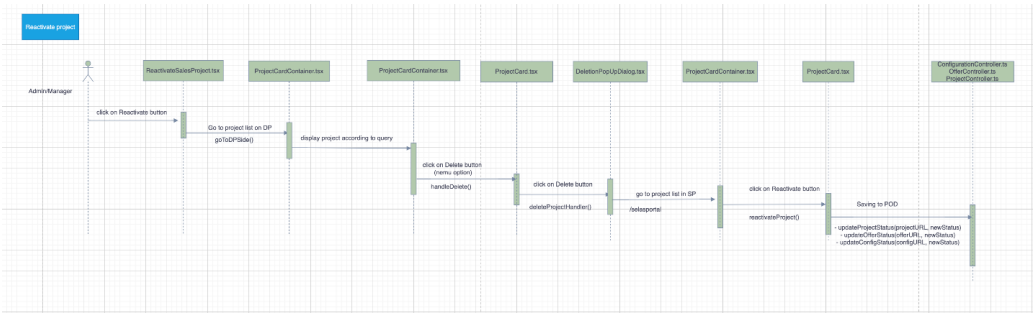

Figure 8. Sequence of interaction in reactivating DP-moved project

5. IMPLEMENTATION PROCESS

This chapter presents the application implementation process of the team at Ekseli Oy and the application structure design of Sales Portal.

5.1. Team Workflow

The starting steps of a project can often be intimidating and time consuming due to the tremendous number of questions that need clarified. In many businesses, a product owner is responsible for handling communication between boards of directors and the team lead or development team. Hence the product owner has to convey the ideas and requirements, such as the goal of the project, from customer to the team lead. The team lead would then discuss with designers and programmers to sketch out a rough layout and resolve any unclarity, such as technical stacks. After which, a biweekly plan of features and accomplishments would be established for the entire team to follow. This process, called Sprint, would be repeated over and over after the end of each two weeks. In addition, there would be meetings held daily among the team to allow the team lead to keep track of the progress. Each team member would say the tasks done the day before, and the ones that is planned for the day. Even though such conferences are meant to be short and concise to allow more time for development, urgent issues can be raised for discussions if needed. These meetings are often known among the developers' community as Daily.

5.2. Overall Technical Stacks

In the technical part, the React framework was used for the client side and setup using *create-react-app* node utility. Among the various styling options, the team proposed one of the best options with default integration with the framework, which are Material UI and SASS libraries. Instead of the default programming language for web applications, JS, TS was used to make the project documentation and code management process simpler and more comprehensive. These supporting libraries were planned to be installed using Yarn. The easiest way to install a Yarn package manager is through the NPM package manager with JS runtime NodeJS. The entire

project was developed using VS Code with Prettier as a linting tool. Lastly, Jest with Enzyme was applied to the project as the test environment. /1/

5.3. Development

Before development can begin, installing Node to the machine was required to set up runtime environment for any websites built with JS frameworks, such as React, Angular, Vue. Even though the latest release version of Node has far exceeded the Long-Term Support (LTS) version, newly added features of Node are rarely needed for production level application development. Then, the initial implementation was generated by a Node module called create-react-app, which is a boilerplate of an example React application from Facebook. Since the project was created using the premade command contains only the bare minimum, additional dependencies were needed to develop the application. Built into the yarn package managers, commands such as *yarn install* or *yarn install + package_name* were used to install dependencies available to the public, or custom dependencies created by the organization. In addition to the commands provided by default for yarn, developers can add custom scripts to be able to efficiently run any needed setups, such as building and testing. For example, one setup that is often used by many businesses is to have separate environment variables, databases, etc. In Sales Portal development, command *yarn dbs* and *yarn db* are to run the project in development and production respectively.

Sales Portal makes use of ready-made components from Material UI for its professional and optimized design. It was implemented to be responsive on different devices and screen sizes, irrespective of the placement of the elements on the users' websites. This allows the website to be accessed and viewed on all devices possible, be it desktops, laptops, tablets, or mobiles. Depending on the screen size, the developers can also utilize different components available to construct an entirely different layout of elements to better fit the screen to keep the user experience optimal while maintaining pleasant UI. For example, a common change used by thousands of developers is the usage of a component called 'burger menu'. It commonly acts as a button to show a group of options to user. In Sales Portal, one key component provided by Material UI is Tabs.

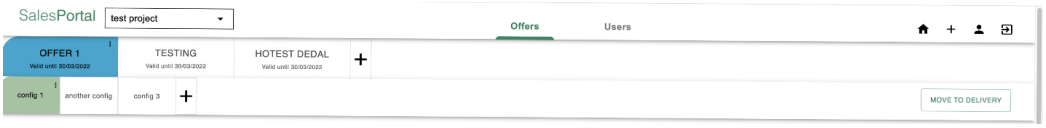

Figure 9. Tabs component used in Offer and Config bars

Since each project can be receive many offers from different companies, each with potentially tens of configurations, presenting all of them to the user in a clear and accessible means plays an important role. Of the many approaches, such as a dropdown list, tabs stand out as the perfect candidate as users can easily view and add another offer, as well as scroll through them quickly. As a result, configs listing for each offer makes use of the same Tabs component.

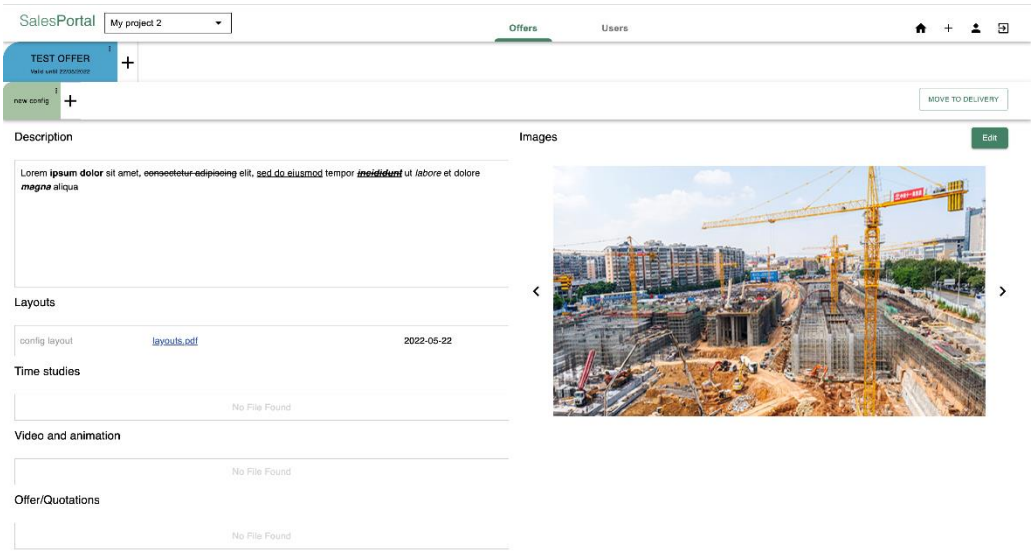

Figure 10. Configuration view in SP Offer

Since the description is not only used by the internal team, but it is also user-facing since the customers would also have a read-only few of all the offers and configs. As a result, the description of the configs would need custom stylings, such as bold, italic, underline, and lists, to highlight different sections. In addition, being able to add headers to clearly separate paragraphs and communicate efficiently to customers can have a vital role in improving the user experience. Lastly, being able to add links to any external resources is often expected of a text box from any users. One of the best tools available in React for rich text editing purposes is DraftJs. It offers

simple API to apply special stylings to any text sections. It accomplishes such effect through an object that stores every info regarding the range of text and the stylings to be applied. One major advantage of such approach is the out-of-the-box support for saving and loading from database by simply stringifying the object when storing and parsing back into the correct object when loading back up.

Often, not everything can be easily explained or demonstrated through nothing but words. Other than description for the configs, admins also need to clearly show images of the layouts, or videos of how certain actions can be done, or even written offers made. Material UI has plugins to allows developers to quickly implement any file type uploads, while keeping a professional looking user interface. While the development for document file upload is simple without any additional steps, the same cannot be applied for images. Since most users these days expect any images upload to have a preview to prevent mistakes, any added images would need to load from the users' local machine and display before those are saved. By default, the file upload plugin provides a local URL to the file location for development of such feature.

Every config has the same set of possible types of data to be added. The admins can add description, video, images, and custom data types, such as layouts, and time studies. Therefore, to fulfill the need for a single database capable of storing all the mentioned types securely, Solid PODs are one of the most obvious choices, especially in the age of control decentralization. In addition to storing files, being able to manipulate or update the data at will is provided to the admins. To allow different operations on the stored files, controllers were built to communicate with POD APIs following CRUD protocol, which stands for Create, Read, Update, Delete.

```
export async function createOffer(projectURL: string, offer: Offer): Promise<Offer> {
  try {
    if (!(await fc.itemExists(projectURL + offer.name.trim() + "/offer.ttl"))) {
      let newOffer: Offer;

      const newDocument = createDocument(projectURL + offer.name.trim() + "/offer.ttl");
      await newDocument.save();

      let profileDoc = await fetchDocument(projectURL + offer.name.trim() + "/offer.ttl");
      let profile = profileDoc.getSubject(projectURL + offer.name.trim() + "/offer.ttl");
      profile.addString(data.solid.write.offerName, offer.name.trim());
      profile.addDateTime(data.solid.write.validFrom, offer.validFrom);
      profile.addDateTime(data.solid.write.validThrough, offer.validUntil);
      profile.addInteger(data.solid.write.offerStatus, offer.status);
      await profileDoc.save();

      newOffer = {
        name: offer.name,
        validFrom: offer.validFrom,
        validUntil: offer.validUntil,
        status: offer.status,
      } as Offer;

      profileDoc = await fetchDocument(projectURL + "/project.ttl");
      profile = profileDoc.getSubject(projectURL + "/project.ttl");
      profile.addString(data.solid.write.offerName, offer.name.trim());
      await profileDoc.save();

      return newOffer;
    }
  } catch (err) {
    throw new Error("Failed to create an offer");
  }
}
```

Code snippet 1. Create an Offer in SP

```
export async function getOffer(offerURL: string): Promise<Offer> {
  const profileDoc = await fetchDocument(offerURL + "/offer.ttl");
  const profile = profileDoc.getSubject(offerURL + "/offer.ttl");

  return {
    name: profile.getString(data.solid.write.offerName),
    validFrom: profile.getDateTime(data.solid.write.validFrom),
    validUntil: profile.getDateTime(data.solid.write.validThrough),
    status: profile.getInteger(data.solid.write.offerStatus),
  } as Offer;
}
```

Code snippet 2. Get offer in SP

```
export async function updateOfferStatus(
  projectURL: string,
  offerName: string,
  newStatus: number
): Promise<void> {
  let profileDoc = await fetchDocument(`${projectURL}/${offerName}/offer.ttl`);
  let profile = profileDoc.getSubject(`${projectURL}/${offerName}/offer.ttl`);

  profile.setInteger(data.solid.write.offerStatus, newStatus);
  await profileDoc.save();
}

export async function deleteOffer(projectURL: string, offerName: string): Promise<void> {
  const profileDoc = await fetchDocument(projectURL + "/project.ttl");
  const profile = profileDoc.getSubject(projectURL + "/project.ttl");

  profile.removeString(data.solid.write.offerName, offerName);
  await profileDoc.save();

  await fc.deleteFolder(`${projectURL}/${offerName}`);
}
```

Code snippet 3. Update offers in SP

For example, both offer, and configs have their own set of functions within a controller that handle the connection with the PODs to operate on the data. As presented in Code snippet 1, Code snippet 2, and Code snippet 3, a controller for offers contain all the types of operations of CRUD, with a small change of naming from "READ" to "GET", even many methods to different scenarios on the frontend. For example, to display a list of offers or configs, a method for getting all the corresponding data is needed to provide a list of required information. Although getting all available data at once, at first sight, is the reasonable choice, a common pitfall for this is the time for the user. As a result, selective return of data is crucial. In the case of showing the list of offers or configs, returning the names, and dates are sufficient until a further time. Only until the user request files of a specific config, which happens when the user selects a config, does the resources become relevant.

Following the same philosophy of avoiding lengthy load time on client, configs should only be updated one at a time. Consequently, performing update on all areas can consume extra resource and time. In order to enable that option, a mechanism for checking whether a provided field was changed during an edit section using a simple array length and object shallow equality comparison. The difference between a deep and a shallow comparison of objects lies in the importance of the

ordering of entries. For a deep comparison, the same order of all keys and values in all layers is needed to be considered equal.

5.4. Testing

Testing is one of the crucial steps in development process to make sure that all features are functioning properly before going to production. As human errors are an inevitable aspect of anything, limiting that can significantly save time and cost over long period of time. To ensure the application is working as intended, there are three types of tests named Unit Testing, Integration Testing and End-to-End Testing. In this project, Unit Testing was primarily used for its simplicity comparing to the others. The tests were written in Jest combining with Enzyme, under *component_name.test.tsx* files inside the corresponding component folder as mentioned in Application Structure. The tests can be run using a *yarn test* command. Though being the most time efficient testing method of the three, unit testing the project along can take up to 40 seconds. This is mostly due to the size, and the high test-coverage of the project, meaning the many lines of application code were thoroughly tested.

In each test file, there would be list of all built test cases needed to be done one by one. The node engine would run all the tests asynchronously, meaning side by side without waiting for the others to complete. This not only increases the test speed, but also enforces the most important rule of writing tests, that tests cannot depend on the results of other to function properly. Depending on the results, the engine would mark either green check if succeeded or red cross if failed.

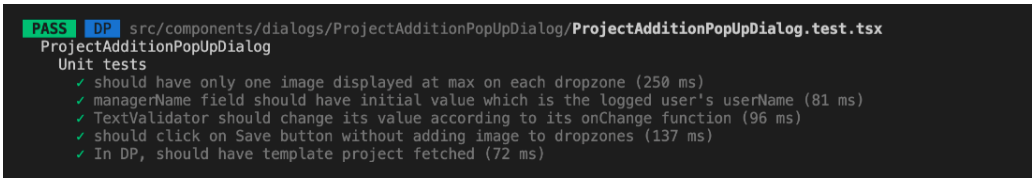

Figure 11. Test cases run successfully

In any test method, the most used setup contains three steps, which are Prepare, Action and Expect. A good practice when writing tests is to clearly separate the steps using empty line. For example, in the first section of Code snippet 4, the Prepare step simply contains a variable to be used in another steps. In most cases, the step consists of more complex setup, such as creating fake database tables, and filling in example data. Afterwards, for the step Action, the wrapper, which is the tested component, in this case a text validator, got called with an onChange event. This updated the text to contain the values provided in the event variable created in the previous step. However, to propagate the change to the virtual DOM, an update method on the component is needed. Finally, in the Expect step, the test verifies whether the component includes exactly the text provided.

```
it("TextValidator should change its value according to its onChange function", () => {
  const event = { currentTarget: { id: "customerName", value: "customer" } };

  /* @ts-ignore */
  wrapper.find("#customerName").at(0).props().onChange(event);
  wrapper.find("#customerName").at(0).update();

  expect(wrapper.find("#customerName").at(0).props().value).toEqual("customer");
});
```

Code snippet 4. Three possible steps in a test case: Prepare, Action and Expect

As with anywhere else in application development, rewriting the same lines of code is wasteful and time consuming hence it should be avoided if possible. This can occur more often for test codes because one setup can be used repeatedly to tests different flows, and scenarios. Not only do setups get regularly reused, but torn down as well. As an example, database tables need to be cleared of after tests to save disk space and memory. Understanding all of this, developers behind the Jest package provided some helper functions to users, called beforeEach and afterEach. They are automatically run before and after each test are executed. In the Code

snippet 5 below, before running any tests, the ProjectAdditionPopUpDialog is always rendered to the virtual DOM and assigned to the wrapper variable inside beforeEach block. Therefore, in both test cases, no duplicate Prepare section is needed.

```
describe("Unit tests", () => {
  let wrapper: ReturnType<typeof mount>;

  beforeEach(() => {
    jest.clearAllMocks();
    act(() => {
      wrapper = mount(
        <ProjectAdditionPopUpDialog
          portalType="SalesPortal"
          projectSaveType={ElementCRUDType.APPEND}
          open={true}
          onClose={dummy}
        />
      );
    });

    wrapper.update();
  });

  it("should have only one image displayed at max on each dropzone", () => {
    expect(wrapper.find("DropzoneArea").get(0).props.filesLimit).toEqual(1);
    expect(wrapper.find("DropzoneArea").get(1).props.filesLimit).toEqual(1);
  });

  it("managerName field should have initial value which is the logged user's userName", () => {
    wrapper.update();
    expect(wrapper.find("#managerName").get(0).props.value).toEqual("tester");
  });
```

Code snippet 5. beforeEach helper function added to test cases

6. CONCLUSIONS

Managing the sales and delivery status of any project is not a simple process for any organization, no matter the size of it. In addition to managing the execution of the project, keeping the communication clear between the admin and clients is a vital part of a successful sale. Moreover, since every single company needs to maintain a high-level secrecy on all their files and data, security is non-negligible. A sales portal helps by creating a single platform with all the needed tools at a reasonable price.

Accordingly, this offers a great opportunity for growth and gaining experience in developing a React application in a professional environment. Not only does it provide a chance to work in a team, consisting of designer, backend developers, but it also allows the developer to work with new and promising technologies, such as Solid POD, SASS, and Enzyme.

As requested by the customers, there are some more work that can still be done to improve the rich text editing usage. First of all, between sessions, the embedded link did not get reapplied and shown in the user interface. In order to fix it, more investigation into the APIs available from DraftJS to debug easily the stored text styling from Solid PODs. Secondly, due to the limited documents and help available with DraftJS usage in the community, the ability to add headings to the text is not fully implemented. By default, the library provides a dropdown with different sizes for headings. However, at the moment, it is not showing any options when clicked on. Finally, one of the features requested and used the most by customers, the ability to copy text from another source and preserve their stylings, is not implemented yet due to the complexity of the library and the lack of well-written documentation. In the future, for this is to be implemented, a deeper investigation into the APIs available and some thorough trial and error work is needed to fully understand and work efficiently with DraftJS.

REFERENCES

1. Yarn Installation. Accessed: 04.04.2022. https://classic.yarnpkg.com/lang/en/docs/install/#mac-stable
2. YarnPkg. Getting started. Accessed: 23.04.2022. https://yarnpkg.com/getting-started
3. JavaScript. Accessed 04.04.2022. https://developer.mozilla.org/en-US/docs/Web/JavaScript
4. React. Accessed 04.04.2022. https://reactjs.org/
5. TypeScript. Accessed 04.04.2022. https://www.typescriptlang.org/
6. Coderslang. 2021.What is Typescript and why should you use it. Accessed 04.04.2022. https://learn.coderslang.com/0056-what-is-typescript-and-why-should-you-use-it/
7. SolidProject. About. Accessed 04.04.2022. https://solidproject.org/about
8. About Jest. Accessed 04.04.2022. https://jestjs.io/
9. Testing React with Jest and Enzyme. Accessed 04.04.2022. https://medium.com/codeclan/testing-react-with-jest-and-enzyme-20505fec4675
10. About Material UI. Accessed 04.04.2022. https://mui.com/
11. About SASS. Accessed 04.04.2022. https://sass-lang.com/
12. About Visual Studio Code. Accessed 04.04.2022. https://code.visualstudio.com/docs
13. About Git. Accessed 04.04.2022. https://git-scm.com/about