



VAASAN AMMATTIKORKEAKOULU  
UNIVERSITY OF APPLIED SCIENCES

Nguyen Nguyen

# CREATING A MODERN WEB USER INTERFACE USING REACT AND TYPESCRIPT

Technology and Communication

2022

VAASAN AMMATTIKORKEAKOULU  
UNIVERSITY OF APPLIED SCIENCES  
INFORMATION TECHNOLOGY

## ABSTRACT

Author	Nguyen Nguyen
Title	Creating a modern web user interface using React and Typescript
Year	2022
Language	English
Pages	58
Name of Supervisor	Timo Kankaanpää

---

This application is a Software as a Service (SaaS) platform that provides an ecosystem, where firms are connected and can be each other suppliers or clients. The potential from this platform is limitless as the time grows, the list of joined companies will increase, which make the network expand and enhance the ability to support client companies.

This platform is a product from Jakamo Company, the platform itself is quite big so the content of this thesis will be the process of developing one of their applications, the todo-list app, where companies can arrange a todo note both for internal usage or share it with other companies from their connections.

Technologies used for development will be React and Typescript to build the frontend, with Material UI version 5.0.0 for a professional outlook, and PHP with PostgreSQL for the backend and database.

---

Keywords	ReactJS, Typescript
----------	---------------------

## TABLE OF CONTENTS

<b>TABLE OF CONTENTS.....</b>	<b>2</b>
<b>LIST OF FIGURES .....</b>	<b>4</b>
<b>LIST OF ABBREVIATION .....</b>	<b>7</b>
<b>1 INTRODUCTION .....</b>	<b>8</b>
1.1 PROJECT OVERVIEW .....	8
1.2 OBJECTIVES .....	9
<b>2 TECHNOLOGIES USED IN THIS PROJECT.....</b>	<b>10</b>
2.1 REACT.JS.....	10
2.1.1 React Components .....	10
2.1.2 React Virtual DOM.....	11
2.2 TYPESCRIPT .....	12
2.2.1 TypeScript addition type .....	13
2.2.2 Interface .....	14
2.2.3 Generics.....	14
2.2.4 Classes .....	16
2.3 MATERIAL UI (MUI) .....	19
2.4 RECOIL .....	19
2.5 THE BACKEND SIDE .....	21
<b>3 ENVIRONMENT SET UP.....</b>	<b>22</b>
3.1 INTEGRATED DEVELOPMENT ENVIRONMENT (IDE).....	22
3.2 RUNTIME ENVIRONMENT .....	22
3.3 HOW TO CREATE A REACT PROJECT.....	23
<b>4 USER STORIES .....</b>	<b>28</b>
<b>5 USE CASE DIAGRAM .....</b>	<b>29</b>
<b>6 APPLICATION IMPLEMENTATION .....</b>	<b>31</b>
6.1 APPLICATION DEVELOPMENT PROCESS .....	33
6.2 PROJECT STRUCTURE .....	34

6.2.1 Introduction about lerna.js .....	34
6.2.2 Installing CRACO .....	41
6.2.3 Leveraging local environment .....	42
6.3 APPLICATION FLOW .....	44
6.4 APPLICATION ARCHITECTURE .....	45
6.5 APPLICATION USER INTERFACE .....	46
6.6 APPLICATION FILTER SYSTEM .....	47
6.6.1 Using recoil.js to store the data globally .....	47
6.6.2 Basic filter .....	48
6.6.3 Advance filter .....	48
6.6.4 Todo list table .....	49
<b>7 TESTING .....</b>	<b>52</b>
7.1 UNIT TESTING .....	52
7.2 TEST RESULTS .....	54
<b>8 CONCLUSION .....</b>	<b>55</b>
<b>9 REFERENCES .....</b>	<b>56</b>

## LIST OF FIGURES

Figure 1. React component using props example.....	11
Figure 2. React component using states example .....	11
Figure 3. Example of an interface .....	14
Figure 4. Example of interface implementation .....	14
Figure 5. Example of a generics interface .....	15
Figure 6. Interfaces where one is a normal interface and the other one uses generics syntax.....	15
Figure 7. Example of variables extend normal interface and generics interface .	15
Figure 8. Example of a function apply generics declaration and how to call it ....	16
Figure 9. Example of field that has initializers, which runs when the class is instantiated. ....	16
Figure 10. Example of constructor in class. ....	17
Figure 11. Example of a method. ....	17
Figure 12. Example of class implements interfaces. ....	18
Figure 13. Example of class extends base class. ....	18
Figure 14. Example of interface extends class. ....	19
Figure 15. Example of data flow in a unidirectional flow. ....	20
Figure 16. Example of an extension on VSC market. ....	22
Figure 17. Command to initialize the application. ....	23
Figure 18. Command to redirect to working directory. ....	23
Figure 19. Project node_module.....	24
Figure 20. Example of div tag with id of root.....	25
Figure 21. Example of how React render.....	25
Figure 22. Public folder in project structure. ....	26
Figure 23. Example of package.json file .....	27
Figure 24. Use Case Diagram .....	30
Figure 25. Example of project structure . ....	34
Figure 26. Example of command lerna init. ....	35
Figure 27. Example of lerna.json configuration. ....	35
Figure 28. Adding workspaces to package.json. ....	35

Figure 29. Components and todo-list-app packages created by .....	36
Figure 30. Shared component in mono-repo.....	37
Figure 31. Exporting “Header” component from “/components/src/index.tsx”. ..	37
Figure 32. Adding entry file for “components” directory.....	38
Figure 33. Importing “Header” component inside App.tsx from “/todo-list-app/src”.....	38
Figure 34. Using “Header” component inside App.tsx from “/todo-list-app/src”. ..	39
Figure 35. Error when starting the server.....	39
Figure 36. “@jakamo/components” is a package of “todo-list-app”. .....	40
Figure 37. Error that causes server fails to start.....	40
Figure 38. CRACO modification.....	41
Figure 39. React scripts modification.....	42
Figure 40. Docker CLI.....	43
Figure 41. Button options to redirect showing page of the old UI Application....	43
Figure 42. Add new page of the old UI application.....	43
Figure 43. Application’s flow .....	44
Figure 44. Flow chart of how the application get data from database through APIs. .....	45
Figure 45. Application’s architecture.....	46
Figure 46. Application List view page.....	47
Figure 47. Basic Filter .....	48
Figure 48. Advance Filter for company that does not have business unit enabled .....	49
Figure 49. Advance Filter for company that has business unit enabled.....	49
Figure 50. Application table. ....	50
Figure 51. Export format selection. ....	50
Figure 52. Table pagination.....	51
Figure 53. Add new button. ....	51
Figure 54. ListView test folder of the application.....	52
Figure 55. Example of mocking data in test.....	53
Figure 56. Example of testing UI of the application, a snack bar popup whenever the data is successfully or fail to return.....	53

Figure 57. Test files in the application.....	54
Figure 58. All the tests in the application are passed.....	54

**LIST OF ABBREVIATION**

<b>UI</b>	<b>User interface</b>
<b>SPA application</b>	<b>Single-page</b>
<b>HTML Language</b>	<b>Hypertext Markup</b>
<b>Js</b>	<b>JavaScript</b>
<b>Ts</b>	<b>TypeScript</b>
<b>IDE Development Environment</b>	<b>Integrated</b>
<b>HOC Component</b>	<b>Higher Order</b>
<b>TDD development</b>	<b>Test-driven</b>
<b>UML Language</b>	<b>Unified Modeling</b>



## 1 INTRODUCTION

Nowadays, in the web development field, there are several famous and powerful libraries that help user build a web application at ease, such as React, Angular or Vue. But through time React stills keep its position as one of the most popular and well-known JavaScript library. According to statistics from SimilarTech, there are 1,240,949 websites that used React in December 2021, and these include some famous websites that have billions of monthly visits, like yahoo.com with 3.5 billion or discord.com with 1.3 billion /1/. These data reveal that React is a well trusted technology that is used in the development of web application that meet the standard for dynamic and responsive perspective.

Typescript is a strongly typed programming language that builds on JavaScript, also known as a superset of JavaScript. It adds additional syntax to improve coding experience as well as better error handler. The way Typescript works is that it converts the code to JavaScript and runs anywhere JavaScript runs like in a browser, on NodeJs or in the app. The way to implement it is just simply add the library to package.json and change the suffix of the file name from js to ts, only then the file can use Typescript syntax and leverage all the benefits from it.

### 1.1 Project Overview

Providing an organized working flow is one of the most crucial reasons why the to do list exists, especially for those who want to pursuit a professional and efficient working environment. For that specific reason, the to do list app is a must have application among all services provided from a supply chain collaboration for manufacturing industries like Jakamo.

The thesis report will cover the process of creation of a modern and user-friendly interface web application that meets the standards for a dynamic and responsive website. Taking advantage out of a React UI library called MUI, that provides a robust, customizable, and accessible library of foundational and advanced components, enabling the ability the build and develop React application faster /2/.

With this application, users will be able to make to do notes both for internal usage or share it with their connections. Since the main type of customer are companies who registered and already a client of Jakamo, that is why they will have the privileges to inherit the connection with other companies that are also clients of Jakamo and hence they can all share the to do notes to each other.

## **1.2 Objectives**

The main objective of this thesis is to create a modern and dynamic web application that meets the requirements of a fully accessible website, which allows users find it interesting and easy to use. The application itself should be a fully functioning to do list app, with user friendly interface that allow users to make to do notes and save it to their dashboard.

Users can see all of their to do when they first access to the app, there will be a filter system enabling user to find the needed to do faster. The filter functionalities consist of Basic filter and Advance filter, each input users type in will be the parameter for the fetching query to the server. Users can also create new to do and share it with their connections. New to do will be saved to the database of that user and as well as the connection they shared.

## 2 TECHNOLOGIES USED IN THIS PROJECT

### 2.1 React.js

React.js is an open-source JavaScript library that is used for building user interfaces specifically for SPA. React is best known for its reusability of UI components. It was created by Jordan Walke, a software engineer working for Facebook. React first deployment was in 2011 on Facebook's News Feed and on Instagram.com in 2012 /3/.

The robust potential of React is that it allows developers to create a large, fast, scalable, and simple web-based SPA, that can change data without reloading the page. It is far more advanced and faster than normal websites that are written by normal HTML /4/.

#### 2.1.1 React Components

One of the crucial factors of React are the React Components. A component is a self-contained piece of code that returns some visual UI representation of HTML, and the core idea here is that these components should be encapsulated. Each component should have its own props and states, and whenever the states changed, the component will be automatically re-rendered according to the logic that has been specified in the component itself, therefore the UI will be updated.

Props and states are two vital elements in React.

- Props are the immutable configuration properties that components use to communicate with each other. By receiving props from parent component and then returning the react element, component can be dynamically reused, therefore save time and efforts.

```

const ExampleComponent = (props) => {
  const { name } = props;
  return <div>Hello {name}</div>;
};

function App() {
  return (
    <div className="App">
      <ExampleComponent name="Pepsi" />
    </div>
  );
}

export default App;

```

Figure 1. React component using props example

- States are the internal mutable properties of the component. Each component will have its own states to store data of the component, when the state changed, the component will be re-rendered. Therefore, states handle the logical action and rendering of the UI. In contradiction with props, states are the properties that cannot be shared to other components, each state should stay inside its component.

```

const ExampleComponent = (props) => {
  const { name } = props;
  const [isShow, setIsShow] = React.useState(false);

  return (
    <div>
      <button onClick={() => setIsShow(!isShow)}>Show</button>
      <h1>{isShow ? `Hello ${name}` : "Please click the button"}</h1>
    </div>
  );
};

```

Figure 2. React component using states example

## 2.1.2 React Virtual DOM

First, we need to understand what DOM is, it is abbreviation for Document Object Model, a cross-platform and language-independent interface that treats scripting language such as HTML or XML as a tree structure.

Virtual DOM is a lightweight copy of DOM, it can do everything DOM can do except Virtual DOM cannot decide what will be shown on the screen. With DOM we have tags like div tag or p tag. On another hand React uses Virtual DOM to create objects like `React.div` or `React.p`, and these objects are what we interact, not DOM or DOM API. That is why the code in React is JSX, a syntax extension to JavaScript, not pure HTML.

The way that Virtual DOM works is, React takes a snapshot of the current situation of the app, right before it applies any alternation to the application. When the app is updated, a Virtual DOM indicated as tree structure will be created, then React will use Diffing Algorithm to compare this snapshot with the Virtual DOM to know exactly which node should be changed. Finally only nodes that are differentiated will be updated to the real DOM and make changes on the screen.

The crucial benefit of using Virtual DOM is, by separating the rendering logic out of DOM, React can be run on multiple environments like React Native or Server-Side Rendering. In fact, Virtual DOM is just a JavaScript object, that is why it can run on every environment as long as that environment enable JavaScript to run. It can be NodeJs or Embedded JavaScript when developing a native application for Web or Mobile platform.

## **2.2 TypeScript**

TypeScript is a programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript and adds optional static typing into the language. TypeScript is designed for the development of large applications and transcompiles to JavaScript. As TypeScript is a superset of JavaScript, existing JavaScript programs are also valid TypeScript program. TypeScript enables developing JavaScript applications for both client-side and server-side execution (as with NodeJs or Deno). There are multiple options available for trans

compilation. Either the default TypeScript Checker can be used, or the Babel compiler can be invoked to convert TypeScript to JavaScript /5/.

One of the reasons that makes TypeScript more popular and becomes a fundamental element for web development is because it makes code more readable and cleaner, improves error handler and easier to maintain. Errors that are easily bypassed are now being revealed by TypeScript, which help reduce effort and time for redundant debugging. In addition, even if the .ts file compilation has error, it can still be generated to .js file on purpose. On another hand, TypeScript provides us a one-of-a-kind code completion as we type in the editor. This is what people often refer to when they mention about tooling in TypeScript.

In order to leverage the power of TypeScript, some of its core knowledge such as Types, Interfaces, Enums, Generics, and Classes should be learnt beforehand. Once we perceive the basics, we can apply it to our project as well as continue to learn the remaining concept.

### **2.2.1 TypeScript addition type**

In JavaScript, we have seven primitive types, like Boolean, Null, Undefined, Number, String, Array, Symbol, and Objects. But TypeScript has provided some of its additional types like Any, Tuple, Void and Enum.

- Any: This is a special type, it is like a work around when we do not know about the type of element that is targeted like a variable or a function parameter, ... Using Any type will let us bypass the type checking of TypeScript and access any properties of that element, like assigning it to (or from) a value of any type or call it as a function.
- Tuple: It is an array, where we will know exactly how many elements are in the array and what type they are.
- Void: It represents the return value of functions which do not return a value. It is the inferred type any time a function does not have any return statements /6/.

- Enum: This type is like a list with a specific range, such as days in a week or colours from a traffic light like red, yellow, and green.

### 2.2.2 Interface

This is an inclusive concept in TypeScript, Interface is a way to declare a type of an object or a function, if the object consistent with the type of interface, then it will pass, else it will report type checking errors.

```
interface userInfo {  
  name: string;  
  age: number;  
}
```

Figure 3. Example of an interface

Figure 3 describe how an interface looks like, and the way to leverage the use of interface is assigning it to an object such as a function parameter as in Figure 4.

```
const User = (user: userInfo) => {  
  return (  
    <div>  
      <h1>  
        {user.name} has age: {user.age}  
      </h1>  
    </div>  
  );  
};
```

Figure 4. Example of interface implementation

### 2.2.3 Generics

One of the ideal ideas in developer world is that everything is dynamic and reusable. TypeScript indeed notice of this and had provided to us a concept called Generics to solve our concern. Generics is a term to indicate that a target such as a variable, interface, class, or a function, does not need to be specified to a single type when instantiated, but instead a type that can work over a variety of types. This concept leverages the reusability of coding, helps reducing effort and time.

```
interface Box<Type> {
  contents: Type;
}
```

Figure 5. Example of a generics interface

Figure 5 displays the syntax of a generics interface. In order to have a deeper understanding about the usage of this generics, we will define two interfaces, one is a normal interface and another one uses generics syntax.

```
interface StringBox {
  contents: String;
}

interface Box<Type> {
  contents: Type;
}
```

Figure 6. Interfaces where one is a normal interface and the other one uses generics syntax

The figure above shows a normal interface where it has a 'contents' attribute that has type as a string and an interface that has the same structure but instead it uses a standard generics syntax.

```
// Normal interface
let boxStr1: StringBox = { contents: "hello" };

// Generic interface
let boxStr2: Box<String> = { contents: "hello" };
let boxNum: Box<Number> = { contents: 25 };
```

Figure 7. Example of variables extend normal interface and generics interface

This figure demonstrates the usage and difference between two type of interface, object 'boxStr1' extends a normal string interface and the other two extend a generics interface. Obviously, we can see that with normal interface, we can only append object to one specific type, on another hand, with generics



interface, we can assign it to whatever type we want such as string or number as shown in Figure 7. Moreover, we can also apply generics for function declaration.

```
function setContents<T>(box: Box<T>, newContents: T) {  
  box.contents = newContents;  
}  
  
setContents<String>(boxStr2, "Hello World!");
```

Figure 8. Example of a function apply generics declaration and how to call it

The function itself has assigned a type T, as well as 'newContents' parameter, on another hand, the parameters "box" leverages the generics 'Box' interface that has the corresponding 'T' type. With this implementation, we can ensure that the parameter that need to be passed will have the correct type.

## 2.2.4 Classes

In ES2015, TypeScript had introduced a full support for "class" keyword, as a superset for JavaScript, it has the type annotations and syntaxes to express the relationships between classes and other type /7/.

Here are some features of the class member:

- Fields: A field creates a public writeable property on a class, it can have an initializer which will be run automatically when the class is initialized.

```
class Point {  
  x = 1;  
  y = 2;  
}  
  
const p = new Point();  
console.log(`${p.x}, ${p.y}`); // 1,2
```

Figure 9. Example of field that has initializers, which runs when the class is instantiated.

- Constructor: Class constructors is where you can add parameters with type annotations, default values or overloads.

```
class Point {
  x: number;
  y: number;

  constructor(x = 1, y = 2) {
    this.x = x;
    this.y = y;
  }
}
```

Figure 10. Example of constructor in class.

- Method: Different from normal function, a function which is an attribute of a class is called a method, it can use all the same type annotation as functions and constructors.

```
class Point {
  x = 10;
  y = 10;

  scale(n: number): void {
    this.x += n;
    this.y += n;
  }
}
```

Figure 11. Example of a method.

Because JavaScript is an object-oriented language, classes in JavaScript can inherit from base classes by using the keyword “implements” and “extends”, but there are some differences between these two.

- Implements: This is used to check if the class meets all the criteria from a specific interface, else it will report some errors. In addition, one class can inherit from multiple interfaces.

```

interface A {
  callA(): void;
}

interface B {
  callB(): void;
}

class Point implements A, B {
  callA(): void {
    console.log("this is A");
  }
  callB(): void {
    console.log("this is B");
  }
}

```

Figure 12. Example of class implements interfaces.

- Extends: This keyword helps class inherit from a base class, including all of the properties and method, and also define additional members.

```

class Animal {
  speak(sound: string) {
    console.log(sound);
  }
}

class Dog extends Animal {
  move() {
    console.log("move a little bit");
  }
}

const milo = new Dog();
milo.move();
milo.speak("woof woof");

```

Figure 13. Example of class extends base class.

There is one particular signature which only feasible in TypeScript, interface can extend classes, whereas it is not possible in other object-oriented languages. In addition, when an interface extends a class, the interface includes all class members (public and private), but without the class' implementation

```
class Point2 {  
  x: number;  
  y: number;  
  
  constructor(x: number, y: number) {  
    this.x = x;  
    this.y = y;  
  }  
}  
  
interface C extends Point2 {  
  z: number;  
}  
  
let point3d: C = { x: 1, y: 2, z: 3 };
```

Figure 14. Example of interface extends class.

## 2.3 Material UI (MUI)

Material UI (version 5.0.0) is a React UI library that is quite ubiquitous and powerful, it provides a variety of pre-defined components that can be used right away or very easy to customize. A robust, open-source library that is well maintained is a crucial element that helps developers bridge the gap to create well architected, modern, and user-friendly applications. Moreover, MUI documentation is quite descriptive and informative, it has been shaped over time after time, that is why it is very easy to perceive by all levels.

## 2.4 Recoil

Normally in a Single Page Application (SPA) like React, most of the time we will encounter a problem called “props drilling”. This is to indicate that with a unidirectional flow technology like React, we will have the situation where the data flow will only go in one direction, e.g., from father component to children component. And if the data we want to share does not have the same common ancestor, we must lift the state up to one specific level where it has the same parent component and can be shared. This process will get more complex as the application grows, which ends up making the application hard to maintain and develop.

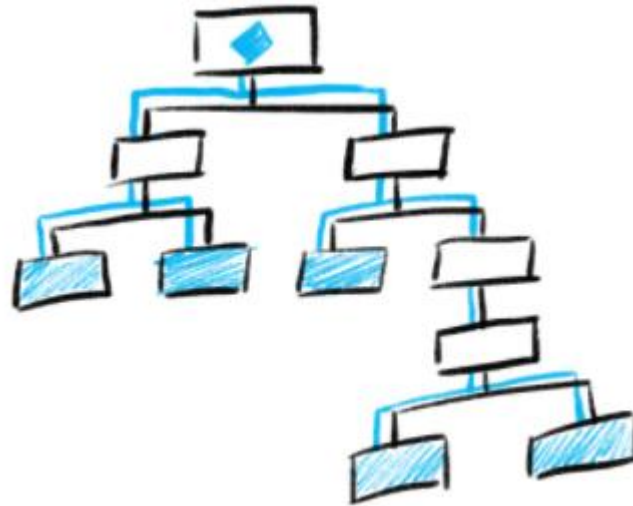


Figure 15. Example of data flow in a unidirectional flow.

To solve this problem, Recoil has been created. It works and thinks like React, some say it is the combination between Redux and Context API, it has the powerful global state management architecture, and it is a library inclusively for React. In Recoil, we have a feature called atoms, they are a unit of state, updatable and subscribable, whenever an atom is updated, the subscribed component is re-rendered with the new value. Atoms can be used in place of React local component state and they can be created at runtime too. In addition, if the same atom is used from multiple components, all those components share their state /8/. All we need to do is wrap the application inside “RecoilRoot” tag from recoil, it is like an HOC where its children can use all the attributes and features in recoil. Then whenever we want to leverage the global state management from recoil, we can specify the hook we want to use in the needed place, and manage the state through “atom”.

There are multiple defined hooks in Recoil that help the state management more efficient, such as:

- `useRecoilState`: This hook subscribes the component to re-render when the state is updated.
- `useRecoilValue`: This hook returns the value of the given Recoil state.

- `useSetRecoilState`: This hook returns a setter function to update the given Recoil state.

These are three main hooks that are widely used across the project, more information can be found at the documentation from RecoilJs official website.

## **2.5 The backend side**

In this project, the deployed and fully maintained server that includes a backend written by PHP and PostgreSQL database will be used. These server-side things are written by another backend developer and leverage the company strategy as well as some specific architectures. But in general, the application use REST apis to communicate with the backend and get data back from the server, using separate endpoints for specific task and feature.

Since this project is a mono-repo project, which means there are multiple of small projects in one big application, the backend side and database need to be developed in a way that isolate important part but still link everything together. Due to the confidential of the company the thesis itself cannot reveal too much about the logic and working flow under the surface.

## 3 ENVIRONMENT SET UP

### 3.1 Integrated Development Environment (IDE)

IDE is a playground, a laboratory of developers where they can have a comprehensive facility to develop software application. IDE provides a source code editor, a debugger and build automation tools, on another hand, it also includes multiple of extensions that act as a handy assistant, such as language snippets, or those that differentiate scopes by adjusting the bracket's colour.

There are a variety of IDEs on the market right now, such as atom, or sublime text, or even notepad is also an option, but overall Visual Studio Code (VSC) is the best IDE among them all. It is not just user friendly with a lot of useful configurations, it also increases working performance of the user by providing a powerful code editor with simple configuration and a massive community that always ready to support and solve any related problem. The VSC market provides a wide range of extension that is extremely helpful and supportive for the coding performance.

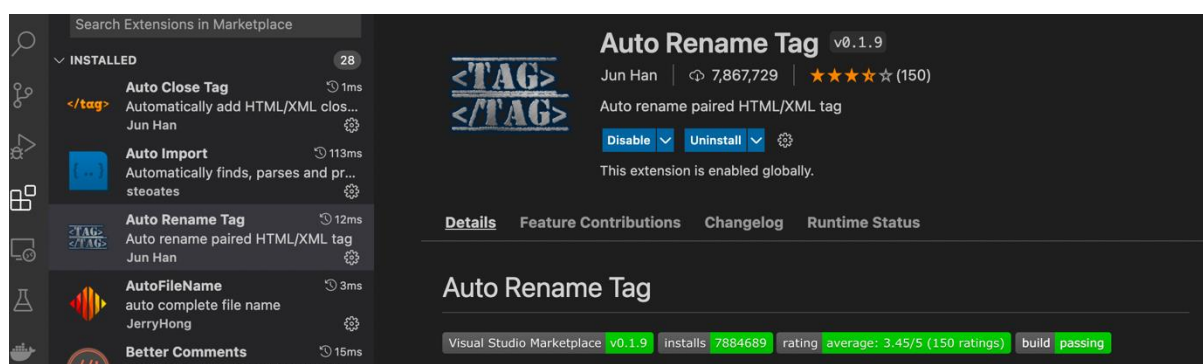


Figure 16. Example of an extension on VSC market.

### 3.2 Runtime environment

In order to write and execute JavaScript we need to install Node, which is an open source, cross platform JavaScript runtime environment that runs on the V8 engine and executes JavaScript outside a web browser. Node lets user use JavaScript to write command line tools and for server-side scripting /9/.

In addition, we will also install npm, which is a package management for node, what it does is it puts modules in a place where node can find them and resolve dependencies confliction intelligently. Npm is also used to install dependencies that are needed in the development process. We can visit Node official website at <https://nodejs.org/en/> to download both node and npm.

### 3.3 How to create a React project

Create React App (CRA) is a best way to start building a react single-page-application, it allows you to use the latest JavaScript features, provides a nice developer experience and optimize the app for production. CRA simply creates a frontend build pipeline and leverages the usage of webpack and babel so that it is compatible with any backend /10/.

To create a new react application, all we need to do is run some commands.

First is the command to initialize the application

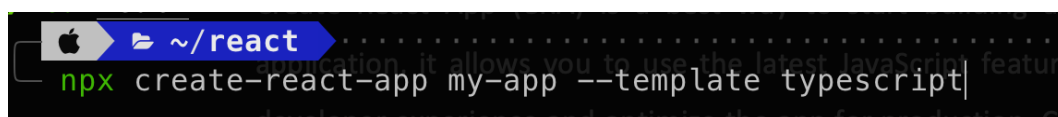
A terminal window with a dark background. The top bar shows an Apple logo on the left and a folder icon followed by the text '~ / react' on the right. The terminal prompt is a green character. The command entered is 'npx create-react-app my-app --template typescript'.

Figure 17. Command to initialize the application.

Note that npx is not a typo error but it is a package runner tool that comes with npm 5.2+.

After everything is installed and set up, redirect to that folder and open VSC

A terminal window with a dark background. The top bar shows an Apple logo on the left and a folder icon followed by the text '~ / react' on the right. The terminal prompt is a green character. The command entered is 'cd my-ap|'.

Figure 18. Command to redirect to working directory.

Here is the project folder structure, it consists of three main folders and some crucial files initially, the amount of file and folder can be increase in the future as the application grows.



- `node_modules`: this folder is where all the packages and dependencies live, every time we install new packages, the source code of those will be stored in here for us to use in the application.

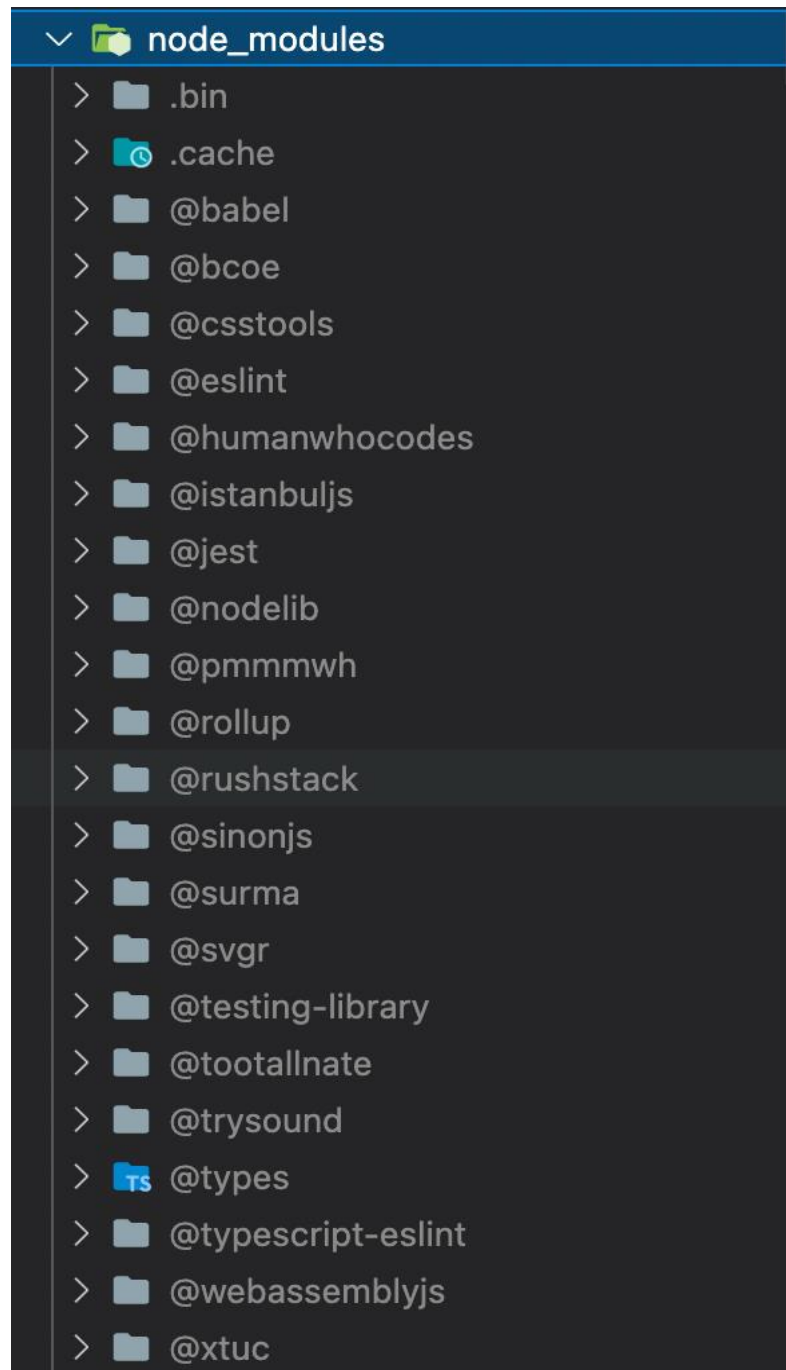


Figure 19. Project `node_module`.

- `public`: this folder is where the `index.html` file, which represents the whole application located. In `index.html` file, there is a `div` tag looks like this.

```
<div id="root"></div>
```

Figure 20. Example of div tag with id of root.

It is called a root DOM node because everything inside it, will be managed by React DOM. React application usually only have a single root DOM node, and in order to render a React element into a root DOM node, pass both to ReactDOM.render().

```
ReactDOM.render(  
  <React.StrictMode>  
    <App />  
  </React.StrictMode>,  
  document.getElementById('root')  
);
```

Figure 21. Example of how React render.

- src: this is the folder where we specify and declare our components. The folder structure will vary based on different company's convention and architecture, but overall, we can put everything that are used to build the application in here.

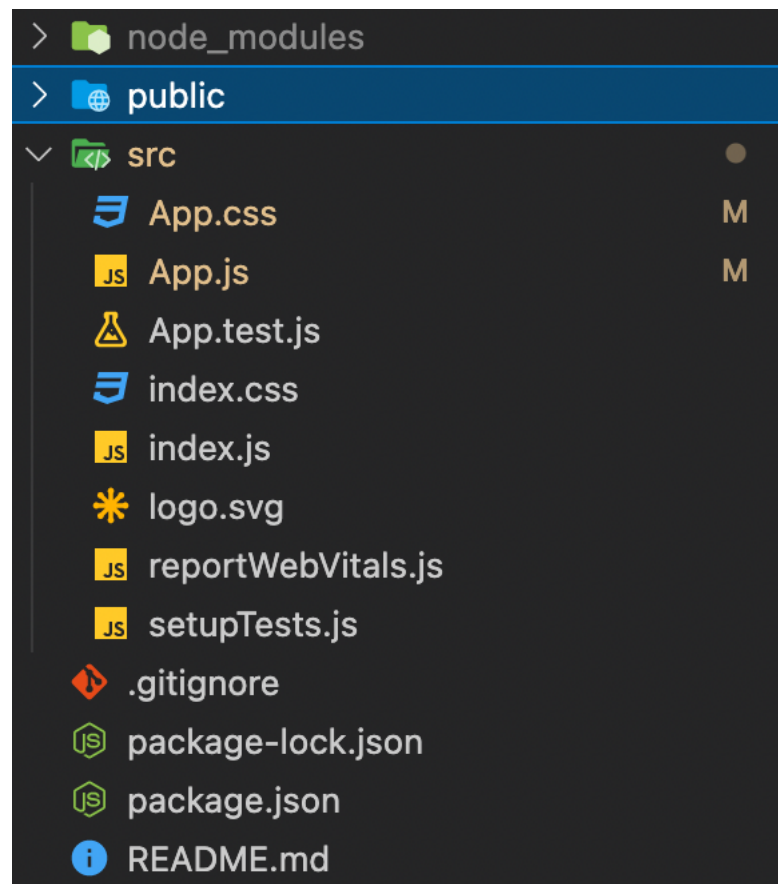


Figure 22. Public folder in project structure.

- .gitignore: this file will store everything that we do not want to include in our commit and be pushed to git.
- package.json: this is the file where we specify all the dependencies with their version and some scripting command to start, test or build the project. We can also declare the application information as well as other configurations for our application.

```
"dependencies": {
  "@testing-library/jest-dom": "^5.16.1",
  "@testing-library/react": "^12.1.2",
  "@testing-library/user-event": "^13.5.0",
  "react": "^17.0.2",
  "react-dom": "^17.0.2",
  "react-scripts": "5.0.0",
  "web-vitals": "^2.1.4"
},
  > Debug
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
```

Figure 23. Example of package.json file .

- package-lock.json: this is the file where we store the version of every dependency that we use in our application. This is to ensure that when working in a team, every collaborator get the same developing environment.
- README.md: this file is the descriptive instruction of how other user can use and develop this app.

## 4 USER STORIES

- As a user: I want the application can create a todo for me and show every todo in an easy-to-read format, so I can efficiently organize my work.
- As a developer: I want the application has a well-structured table that showing todo, with a pagination so user can easily interact with their data.
- As a developer: I want the application has a filter system, from basic to advance, so that the user can easily find their needed information from the app.
- As a tester: I want all the test cases are covered and successfully passed, so that the application works flawlessly.

## 5 USE CASE DIAGRAM

A use case diagram in UML is the primary form of system/software requirements for a new software program underdevelopment. It demonstrates the interactions between user and the underdevelopment software. Use cases specify the expected behaviour (what), and not the exact method of making it happen (how). It can be showcased as both textual and visual representation (i.e., use case diagram). A use case diagram should be simple, it should only summarize some of the relationships between use cases, actors and systems. In addition, it should not show the order of which step are performed to achieve the goal of each use case /11/.

In the UML diagram, use cases are displayed as oval shapes and user is denoted as a stick figure. In Figure 24, a use case diagram for this project is showed. According to the figure, when user is in ListView page, he/she can create a new todo, leverage the use of filter system, which consists of Basic Filter and Advanced Filter. Moreover, there is also a table of todo in ListView page, and as the figure has showed, user is capable of sorting the order of todo in the table as required. On another hand, with user that has Business Unit enabled, he/she will additionally has the Business Unit filter in the filter system and can sort the business attribute in their todo. Finally, every user can open the VisualView page when they are in ListView page.

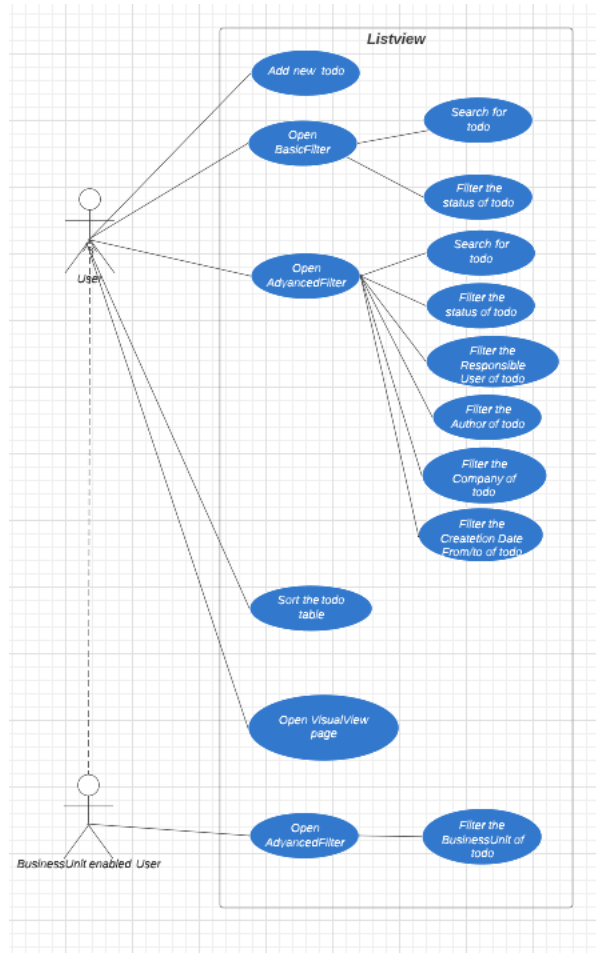


Figure 24. Use Case Diagram

## 6 APPLICATION IMPLEMENTATION

The main goal of this application is to create a to-do list where companies can create and share with each other. A to-do list application has been an important app since it helps people increase their working efficiency and productivity. It not just plays the role as a scheduler that support people arranging their upcoming works, but it is also a reminder that reminding people about tasks that still need to be done.

Here are some requirements for the application:

### REQUIREMENTS FOT THE APPLICATION TABLE.

REQUIREMENTS	PRIORITY (1 BEING THE HIGHEST)
THE APPLICATION SHOULD BE WRITTEN BY REACT/TYPESCRIPT.	1
STYLING OF THE APPLICATION SHOULD BE HANDLED BY MATERIAL UI (MUI).	1
EVERY COMPONENT IN THE APP SHOULD BE TESTED WITH UNIT TESTING, USING JEST AND REACT TESTING LIBRARY.	2
THE APPLICATION NEEDS TO HAVE A "LIST VIEW" PAGE THAT HAVE A FILTER SECTION AND A TABLE THAT SHOWS ALL THE TODO ITEMS.	1
THE FILTER SECTION INCLUDES "BASIC FILTER" AND "ADVANCE FILTER".	1



THE TABLE CAN BE SORTED ALPHABETICALLY FROM A-Z OR VICE VERSA BY CATEGORY.	2
THE TABLE HAS PAGINATION THAT ALLOW USER TO CHOOSE A SPECIFIC NUMBER OF DISPLAYED ITEMS AND CAN BE CLICKED TO NEXT PAGE TO SHOW THE REMAIN ITEMS.	2
IN BASIC FILTER, THERE WILL BE A SEARCH BAR TO SEARCH TODO ITEM	1
IN BASIC FILTER, THERE WILL BE THREE BUTTONS TO FILTER THE TYPE OF THE TODO ITEM	1
IN ADVANCED FILTER, THERE WILL BE AN AUTOCOMPLETE TO FILTER THE AUTHOR OF THE TODO ITEM	1
IN ADVANCED FILTER, THERE WILL BE AN AUTOCOMPLETE TO FILTER THE RESPONSIBLE USER OF THE TODO ITEM	1
IN ADVANCED FILTER, THERE WILL BE AN AUTOCOMPLETE TO FILTER THE COMPANY OF THE TODO ITEM	1
IN ADVANCED FILTER, THERE WILL BE A DROPDOWN TO FILTER THE STATUS OF THE TODO ITEM	1
IN ADVANCED FILTER, THERE WILL BE A TEXT FIELD TO SEARCH FOR A TODO ITEM	1
IN ADVANCED FILTER, THERE WILL BE A TEXT FIELD TO SEARCH FOR TAG OF THE TODO ITEM	2

IN ADVANCED FILTER, THERE WILL BE A DATE RANGE PICKER TO PICK A CREATION DATE - FROM OF A TODO ITEM	1
IN ADVANCED FILTER, THERE WILL BE A DATE RANGE PICKER TO PICK A CREATION DATE - TO OF A TODO ITEM	1

This application is built based on the stack that has been introduced in Chapter 2. The application's plan is made based on careful discussion, following developing specifications and company's convention.

## 6.1 Application development process

In order to have a good developing architecture, the process of creating the application must follow these specific steps.

1. Ignite the idea of the feature, is it a recreation of the old feature, or it is a demand from customer.
2. Create an issue on organization's Github, make it as descriptive as possible and state out all the requirements.
3. The assigned developer will create a mock-up for that feature, if it a whole new page or an UI component.
4. A meeting with Product Owner will occurs in order to review the mock-up and agree on the final layout of the new page.
5. When everything is approved, then the creation of the new feature is started. Make sure to move to a separate branch for that feature and naming the branch following company's convention.
6. After the component is finished, then unit testing should be implemented next. Every aspect should be covered and the best way to cover everything is to make the test case first. (Sometimes the tests are made before implementing the code, following TDD process)

7. All test cases should be passed before making a pull request. Team members shall be requested to review the code carefully before merge the branch to main branch.
8. Move on to another issue and follow above steps.

## 6.2 Project structure

This project is implemented as a mono-repo application, which means there will be multiple small projects represent the application services inside one big repository. And in order to set up a mono-repo project, we will need to use lerna.js, a tool to manage JavaScript projects with multiple packages.

This mono-repo project will have a structure like

```
jakamo-frontend
├── packages
│   ├── components
│   │   └── (shared components that can be used for other application)
│   └── todo-list-app
│       └── (components that are only used for todo-list-app)
```

Figure 25. Example of project structure .

### 6.2.1 Introduction about lerna.js

Nowadays, large and complex monolithic application is getting more difficult to maintain and develop as the time go by and the application grows. That is why splitting large code bases into multiple smaller, separate and independent versioned packages becomes common and popular, some companies decide to move the entire infrastructure to a microservice platform, but that will be extremely hard and require a lot of time and effort. Another solution is converting the application to a mono-repo project. Here are steps to set up a mono-repo project:

1. How to set up Lerna

First run “lerna init”, this will create a lerna.json file with some basic lerna configuration along with a package.json file and also a “packages” folder where all the packages will be located in here.

```
> lerna init
lerna notice cli v4.0.0
lerna info Updating package.json
lerna info Creating lerna.json
lerna info Creating packages directory
lerna success Initialized Lerna files
```

Figure 26. Example of command lerna init.

In lerna.json file, “npmClient” and “useWorkspaces” needs to be added so it will use yarn workspaces.

```
lerna.json > ...
1   {
2     "packages": [
3       "packages/*"
4     ],
5     "npmClient": "yarn",
6     "useWorkspaces": true,
7     "version": "0.0.1"
8   }
9
```

Figure 27. Example of lerna.json configuration.

And in package.json, we add this line

```
},
"workspaces": [
  "packages/*"
],
```

Figure 28. Adding workspaces to package.json.

This is to indicate that we want to include everything inside “packages” folder. After that we move into “packages” folder and use “create-react-app” to create inner packages which represent inner applications. In this

project, we will create “components” package and “todo-list-app” package.

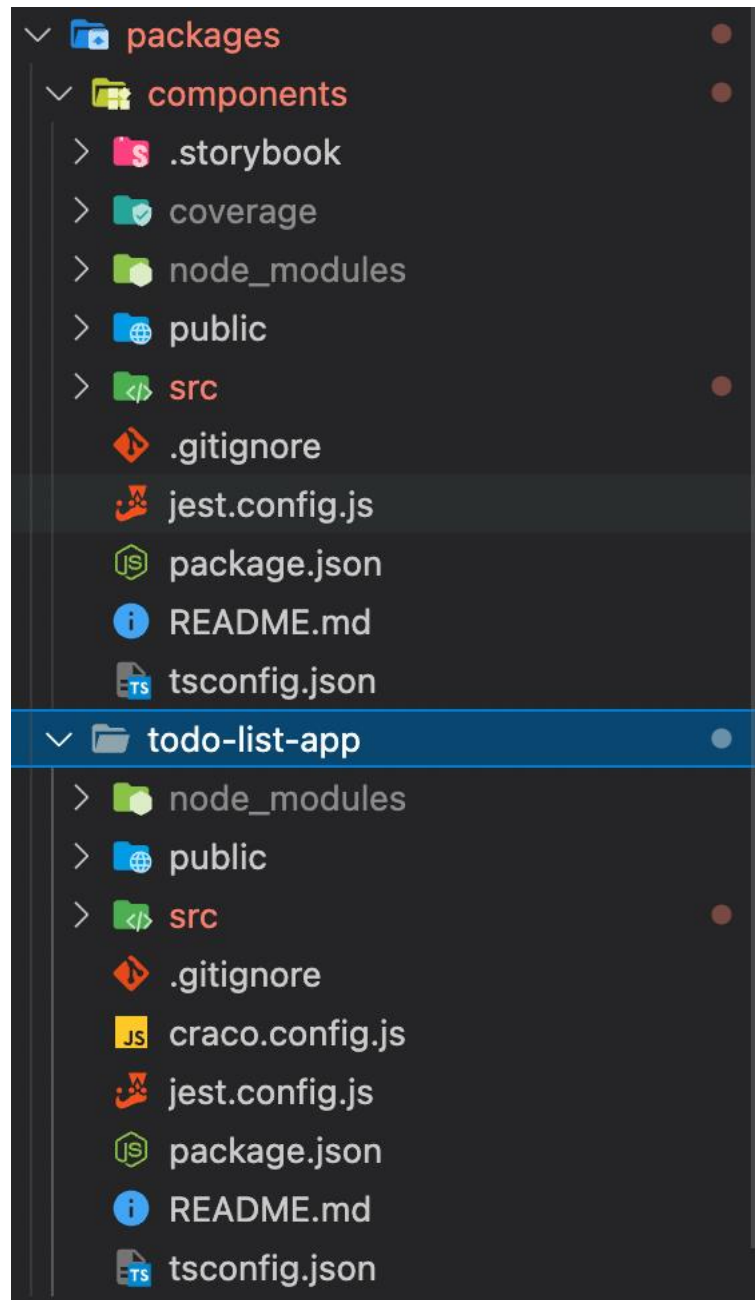


Figure 29. Components and todo-list-app packages created by

“create-react-app” inside “packages” folder.

Then inside todo-list-app, we redirect to package.json file and add a prefix to “name” attribute, from “todo-list-app” to “jakamo/ todo-list-app”, and so for the package.json inside “components” and root directory, from

“components” to “jakamo/components” and from “root” to “jakamo/root”. The purpose of renaming is that we can reference our packages inside our application and other packages with these names.

## 2. Create shared component in mono-repo

First, in “components” directory we create a “components” folder, this is where we locate all the shared components that are used across the application.

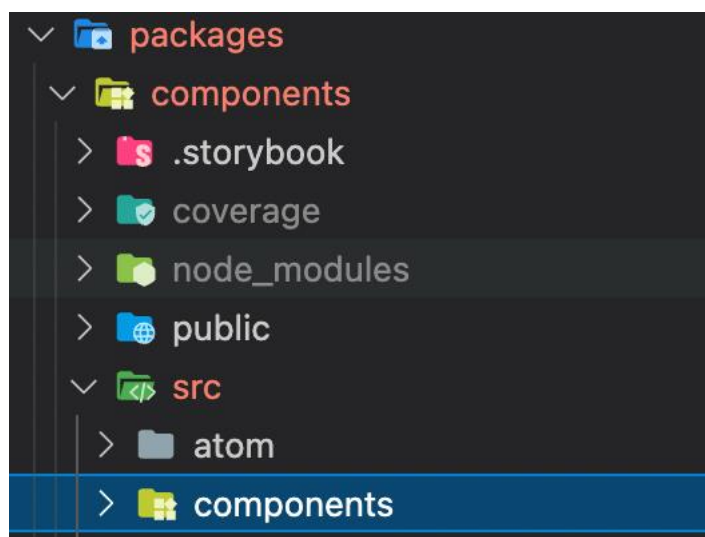


Figure 30. Shared component in mono-repo.

Then whatever component is created in “components” folder could be use in another folder like “todo-list-app” folder. For example, we will create a “Header” component inside “components” folder, export it from index file and import it inside “App” components from “todo-list-app”.

After finishing the “Header” component, we need to remove everything inside “/components/src/index.tsx” and replace with this line of code from Figure 30.

```
export { default as Header } from "./components/Header/Header";
```

Figure 31. Exporting “Header” component from “/components/src/index.tsx”.

Additionally, there is one more thing we need to do before we can export “Header” component, is that we need to define the entry file of “components” directory. We can do that by adding this attribute “main”: “./src/index.tsx” inside package.json inside this directory.

```
{  
  "name": "@jakamo/components",  
  "version": "0.1.0",  
  "main": "./src/index.tsx",  
}
```

Figure 32. Adding entry file for “components” directory.

Then we can import and use “Header” component inside “todo-list-app”.

```
import { Header } from "@jakamo/components";
```

Figure 33. Importing “Header” component inside App.tsx from “/todo-list-app/src”.

```

const App: FC<Props> = () => {
  return (
    <RecoilRoot>
      <ThemeProvider theme={JakamoMainTheme}>
        <SnackbarProvider
          autoHideDuration={5000}
          maxSnack={4}
          anchorOrigin={{ ...
        >>
      </SnackbarProvider>
      <Router>
        <Container className="App">
          { /* use additionalButton props for additional button */ }
          <Header
            title="to-do lists"
            listViewHref={"/todolist/"}
            visualViewHref={"/todolist/visual"}
          />
          <ListView />
        </Container>
      </Router>
    </ThemeProvider>
  </RecoilRoot>
);
};

```

Figure 34. Using “Header” component inside App.tsx from “/todo-list-app/src”.

Then we open the terminal and redirect to “todo-list-app”, as soon as we start the server, we will notice there is an error called “Failed to compile”.

```

./src/App.tsx
Module not found:

```

Figure 35. Error when starting the server.

This is because “todo-list-app” does not recognize “components” package. That is why we need to go back to “component” folder and open the terminal in order to add it as a dependency to “todo-list-app” package. Lerna support us to do this just by typing this command in the terminal at root level, “lerna add @jakamo/components --scope=@jakamo/todo-list-



app". This command indicates that we want to add "components" package as a dependency to the "todo-list-app" package. And now if we check the package.json file in "todo-list-app", we will see "@jakamo/components" as a dependency.

```
"dependencies": {  
  "@craco/craco": "^6.2.0",  
  "@date-io/date-fns": "1.x",  
  "@emotion/react": "^11.4.1",  
  "@emotion/styled": "^11.3.0",  
  "@jakamo/components": "^0.1.0",  
}
```

Figure 36. "@jakamo/components" is a package of "todo-list-app".

And now if we start the "todo-list-app" server, we will see that it resolves the problem but the application still fails to start because of this error.

```
Module parse failed: Unexpected token (6:5)  
File was processed with these loaders:  
* ../../node_modules/@pmmmwh/react-refresh-webpack-  
plugin/loader/index.js
```

Figure 37. Error that causes server fails to start.

This error happens because we are using tsx and jsx code inside out "components" package and using that inside "todo-list-app" package, but "create-react-app" configures webpack to use bubble loader for the code inside the project, but in this case, it does not use the loader for the code that we are importing from another package. So, what we need to do is to use bubble loader not only for "todo-list-app" but also for the code from "components" package.

The way to do this is to modify the webpack configuration, and since we are using “create-react-app”, we can use a package called “CRACO”.

### 6.2.2 Installing CRACO

CRACO is a create react app configuration override, it allows us to modify different configuration including the webpack. All we need to do is create “craco.config.js” file and create our modification there.

In order to install CRACO, we need to open the terminal at “todo-list-app” level since we only need CRACO here right now and run “yarn add @craco/craco -D”, this command will help us installing CRACO as a dev dependency.

After it has been installed, we are going to open “craco.config.js” file, and add our modification.

```
const path = require("path");
const { getLoader, loaderByName } = require("@craco/craco");

const packages = [];
packages.push(path.join(__dirname, "../components"));

module.exports = {
  webpack: {
    configure: (webpackConfig, arg) => {
      const { isFound, match } = getLoader(
        webpackConfig,
        loaderByName("babel-loader")
      );
      if (isFound) {
        const include = Array.isArray(match.loader.include)
          ? match.loader.include
          : [match.loader.include];
        match.loader.include = include.concat(packages);
      }

      // The following makes so that we output single bundle instead of multiple chunks
      webpackConfig.optimization = {
        runtimeChunk: false,
        splitChunks: {
          chunks(chunk) {
            return false;
          },
        },
      };
      // Output information for the single bundle
      webpackConfig.output = {
        path: path.join(__dirname, "build"),
        publicPath: '/',
        filename: "todo-list-app.bundle.js"
      };
      return webpackConfig;
    },
  },
};
```

Figure 38. CRACO modification.

What we are doing inside this config file is that we are importing “getLoader” and “loaderByName” functions from CRACO, and then we will define a “packages” array and inside that is the path to all the packages we want to use

bubble loader. So besides “todo-list-app” package, we also want to use it on “components” package, that is why we push path to “components” package into this array. Finally, the config file will export an object and because we want to modify the webpack configuration, we will configure webpack properties and export it. So, everything that are exported inside the webpack scope, are the modification we want to make.

In order to apply changes that we made, we also need to make some small adjustments for the react-script inside package.json too. Since we want to start the server inside “todo-list-app”, we will redirect to package.json file in this directory, move to scripts section, and update the commands to this.

```

"scripts": {
  "start": "craco start",
  "build": "run-script-os",
  "build:mac": "craco build 66 mkdir -p ../../public/bundles 66 cp build/todo-list-app.bundle.js ../../public/bundles/todo-list-app.bundle.js",
  "build:windows": "craco build 66 if not exist ..\\..\\public\\bundles mkdir ..\\..\\public\\bundles 66 copy .\\build\\todo-list-app.bundle.js ..\\..\\public\\bundles\\todo-list-app.bundle.js",
  "build:default": "craco build 66 mkdir -p ../../public/bundles 66 cp build/todo-list-app.bundle.js ../../public/bundles/todo-list-app.bundle.js",
  "watch": "watch 'yarn build' ./src",
  "test": "react-scripts test --env=jsdom",
  "eject": "craco eject",
  "lint": "eslint src --ext .ts,.tsx,.js,.jsx",
  "prettier": "prettier --write src"
},

```

Figure 39. React scripts modification.

By updating the existing calls, we are now can leverage the CRACO CLI and apply our modifications.

### 6.2.3 Leveraging local environment

In order for the application working properly, we need to start the server. The whole backend has been dockerized into an image, so we need to start docker and run the image. We can do that by using docker CLI. In this application, the image that need to be run is “jakamo” image, with docker CLI, the process is much simpler, we do not need to use the terminal and type in docker command, we just need to press the play button on the row of the selected image, and that is all.

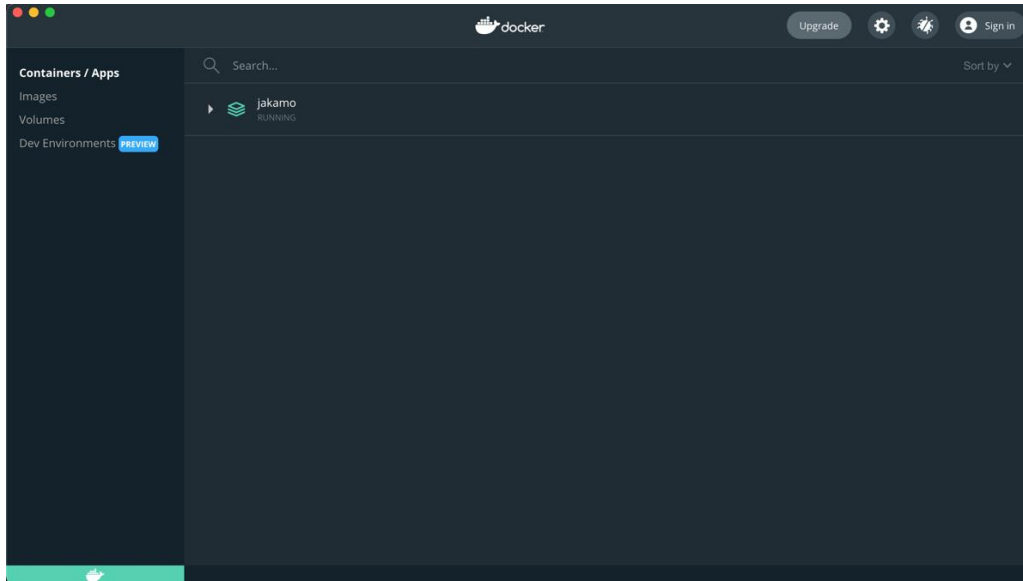


Figure 40. Docker CLI

Due to the fact that this is a recreation of the already working application, the development process is closely related to the old UI application, e.g. in order to create a new todo item for testing purpose, we need to add it from the old UI application. To add new item, we need to choose the “ADD NEW” button.



Figure 41. Button options to redirect showing page of the old UI Application.

**NEW TO-DO LIST**

**SHARING \***

Owner company  
c2  
Please select the business unit.

**STATUS**

Change status automatically to Completed when all tasks have been completed.

**BASIC INFO**

Title \*

Description

**USERS**

Author jaa tee  
Responsible users   
jaa tee, c2

**TAGS**

Write a tag

Figure 42. Add new page of the old UI application.

After being redirected to the “Add new” page, user needs to fill in all the requirement text fields and then click “Save” button, that is how a new todo item is added. By the time this thesis is being written and presented, the new backend for add new todo item still in developing process, therefore the new frontend cannot be created and used to create new item directly.

### 6.3 Application flow

Initially, when starting to use the app, user need to create a new todo by clicking the add button, then all todos will appear on the table at Listview page. User can interact with the table by moving to next/previous page via the pagination or find the todo via the filter system.

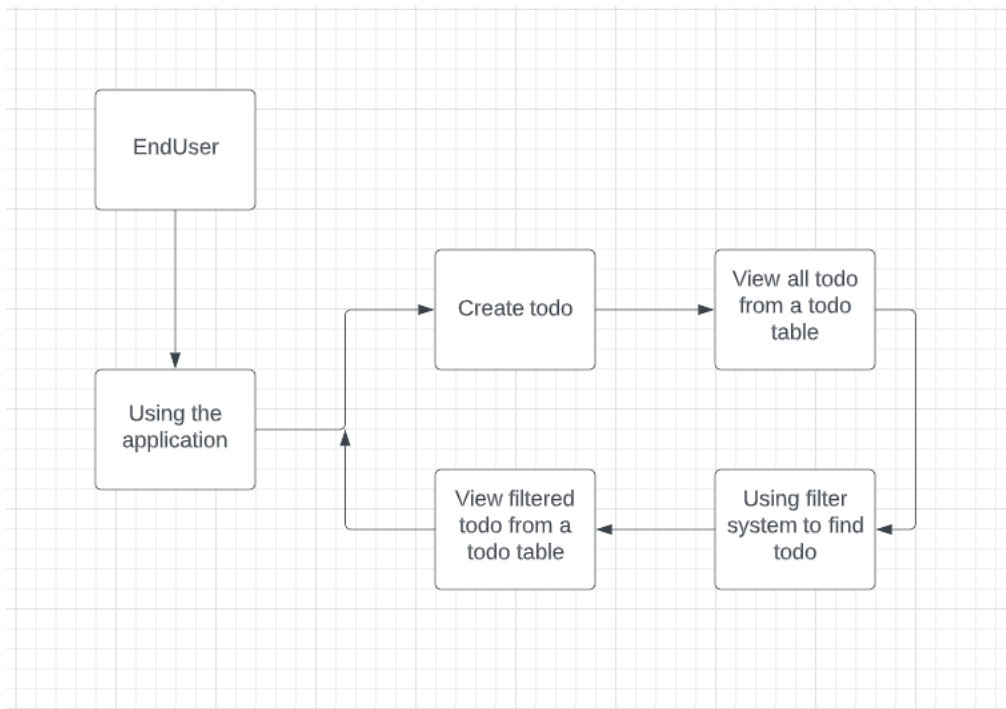


Figure 43. Application's flow

Additionally, the application use REST API to connect to a backend that includes different modules like to-dos, users and companies. Every time client makes an action, the application will make a request to the database and return the requested data

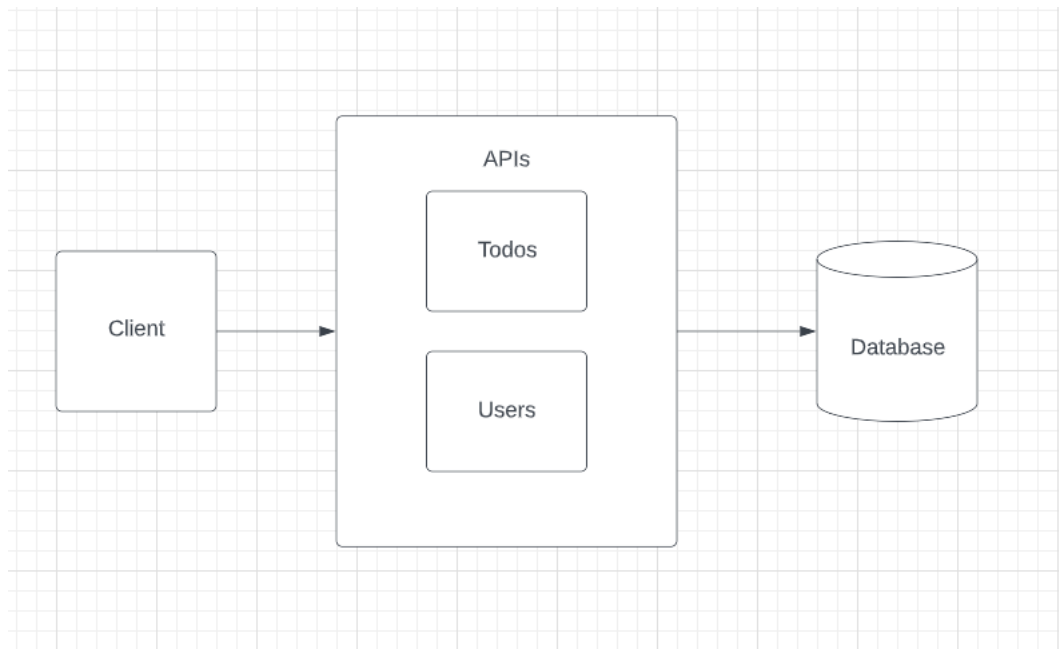


Figure 44. Flow chart of how the application get data from database through APIs.

## 6.4 Application architecture

Due to the confidential problem, the Component node will not be specified and only be called as Component.

The diagram below shows the architecture of the application. Component is where user make interaction with the app, then the state will be stored in atom from recoil, by using `setRecoilValue`. Then the atom will be used in `ListView` component, which is the component that make request to the server, atom will be passed as request's parameter.

After the data have been returned from the server to `ListView` component, it will be passed to atom again and ready to be used across the application, any component that need the atom, can import `useRecoilValue`, and use it inside the component.

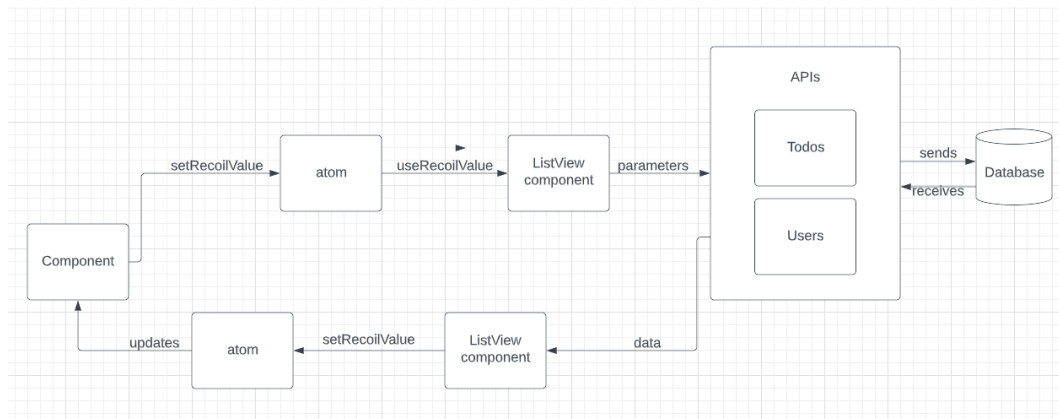


Figure 45. Application's architecture.

## 6.5 Application user interface

There will be multiple pages in this application, but the main page will be "List view" page, this page main features will include filter system and todo-table. The filter system helps users quickly find the todo they need whereas the todo-table will show the todos that being fetched directly from the server.

The styling method used in this application is mainly leveraging MUI components combine with some scss styling. In addition to build better user experience, the animation of the application is built with a framework called "framer motion", with this framework, we can make the component in the app appears smoother.

**TO-DO LISTS** LIST VIEW VISUAL VIEW

**BASIC FILTERS** **ADVANCED FILTERS**

Search ALL MY RESPONSIBILITIES CREATED BY ME

**ALL** **SENT** **RECEIVED** EXPORT AS

CREATION DATE ↑	EDIT TIME	TITLE		STATUS	OWNER	SHARED WITH
2021-09-20 06:49:02	2021-10-19 06:21:25	todo 1		In progress	pepsi co	coca cola
2021-09-20 06:49:30	2021-09-20 06:49:30	todo 2		In progress	pepsi co	coca cola
2021-09-20 07:44:19	2021-09-20 07:44:19	todo 3		In progress	pepsi co	coca cola
2021-09-30 11:07:02	2021-09-30 11:07:02	todo from coca		Completed	coca cola	pepsi co
2021-09-30 11:11:50	2021-09-30 11:11:50	todo2 from coca		In progress	coca cola	pepsi co

+

Figure 46. Application List view page

## 6.6 Application filter system

In todolist application, there will be two filter mode, Basic Filter and Advance Filter, and as the name implies, Basic Filter will only require some simple inputs, in order to quickly find the todo, and on the other hand the Advance Filter will have more detailed inputs with advance logic that help users find the todo more precisely.

### 6.6.1 Using recoil.js to store the data globally

There was a discussion between members in a team to decide which technology should be used to manage the state globally, whether context api, redux, redux tool kit or recoil. Each of them has their own pros and cons, but the chosen option should be based on the compatible of the technology with the application, and recoil is the final decision.



With recoil, process to store data globally is much simpler due to its clear and simple syntax, every needed actions are provided as the form of hooks.

### 6.6.2 Basic filter

There are two columns in this filter, in the left column there will be the search input where user will fill in keywords of the todo they want to find, and in the right columns, there will be a set of three buttons to filter the type of the todo, the buttons type will be “All”, “My responsibilities”, and “Created by me” gradually.

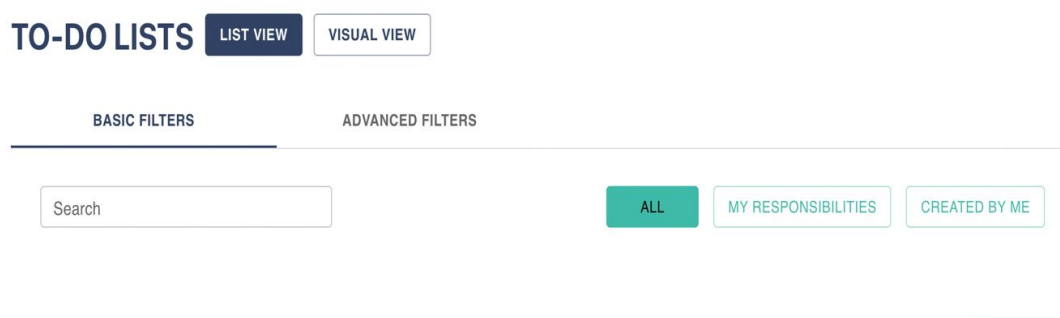


Figure 47. Basic Filter

The way that this filter system works is that whenever user type something in the search bar, or click any button, it will send the request to the server with the corresponding parameters that are recorded from user selections.

### 6.6.3 Advance filter

In Advance Filter, the inputs are divided into three columns, the first column includes autocompletes of “Responsible user”, “Author”, and “Company”. Moreover, with companies that has “Business Unit” enabled, there will be an additional autocomplete to filter out the business unit also.

**TO-DO LISTS** LIST VIEW VISUAL VIEW

**BASIC FILTERS** **ADVANCED FILTERS**

Responsible user	Status All	Search
Author		Tags
Company		Creation Date - From
		Creation Date - To

Figure 48. Advance Filter for company that does not have business unit enabled

**TO-DO LISTS** LIST VIEW VISUAL VIEW

**BASIC FILTERS** **ADVANCED FILTERS**













Responsible user	Status All	Search
Author		Tags
Company		Creation Date - From
Business Units		Creation Date - To

Figure 49. Advance Filter for company that has business unit enabled

The second column includes a “Status” dropdown that help users filter out the status of todo easily. Finally, the third column includes a “Search” and “Tags” text field for user to fill in keyword of the item they want to find, then the date picker in order to find the date range of the items.

#### 6.6.4 Todo list table

The most important part of the application is the display todos table, in this table, user can find all the todo items, or the filtered todo items if any filter option exists. The table itself has the sorting functionality with pagination which help user easily use and manage the todo items.

ALL	SENT	RECEIVED	EXPORT AS			
CREATION DATE ↑	EDIT TIME	TITLE		STATUS	OWNER	SHARED WITH
2022-02-07 08:12:05	2022-02-07 08:12:05	softdrink union with coca	  	In progress	c2	coca cola
2022-02-07 08:13:50	2022-02-07 08:13:50	softdrink union with pepsi	  	In progress	c2	pepsi co
2022-02-07 08:23:08	2022-02-07 08:23:08	jaatee union with sprite	  	Completed	c2	sprite
2022-02-07 08:53:25	2022-02-07 08:53:25	completed todo	  	Completed	c2	

Rows per page 5 ▾ 1-4 / 4 < >

Figure 50. Application table.

As Figure 44 has shown, the table also has its own filtering system too, there are three tabs on the left corner that let user to choose the type of item to be shown, whether the type is “Sent”, “Receive” or both of them. In the right corner there is a dropdown button that let user export the display todo table to either PDF format or XLS format. When use click the button, it will trigger a function to export the table and its data with the corresponding format. However right now this function is not yet being implemented.

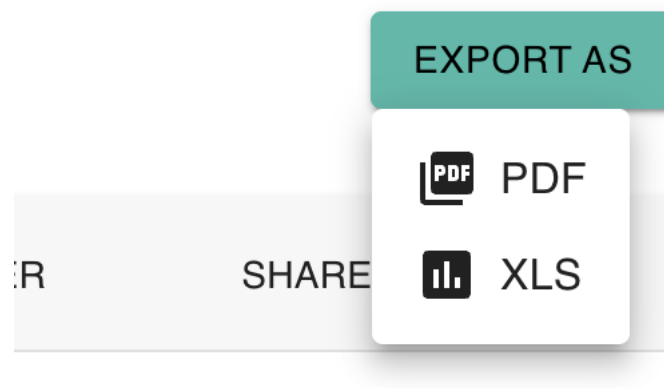


Figure 51. Export format selection.

We can also see from Figure 44, the table also have a pagination at the bottom of the table. It allows user choose the display items on one page and forward to next page if there are still remain other item.

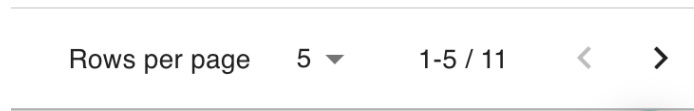
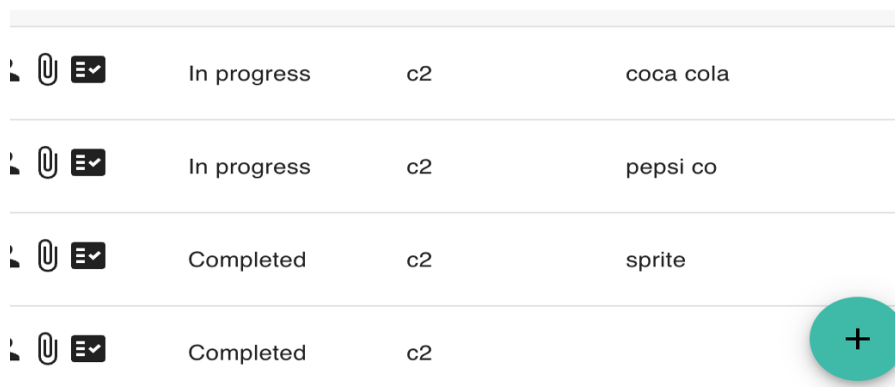


Figure 52. Table pagination.

There is also a button to add new todo item, the button is sticky, which means it will scroll along with the user's scrolling in order to create a friendly user experience.

A screenshot of a table with four rows. Each row starts with a small icon of a person, a paperclip, and a list icon with a checkmark. The rows contain the following data: "In progress", "c2", "coca cola"; "In progress", "c2", "pepsi co"; "Completed", "c2", "sprite"; "Completed", "c2". To the right of the table, there is a green circular button with a white plus sign (+) inside, which is highlighted with a soft shadow. This button is sticky and remains in the same relative position on the screen as the user scrolls through the table rows.

In progress	c2	coca cola
In progress	c2	pepsi co
Completed	c2	sprite
Completed	c2	

Figure 53. Add new button.

## 7 TESTING

### 7.1 Unit testing

This application is following the unit testing approach, which means whenever a new feature is implemented, it needs to be tested before merging. This way the application will always works as expected and avoid unrelated technical errors.

The purpose of testing is the test itself must act as a user behaviour to make sure that every feature in the application works appropriately. Debugging process is also highly used in order to trace out if there are any bug or unnoticed warnings, then the solution can be quickly applied to fix the problem.

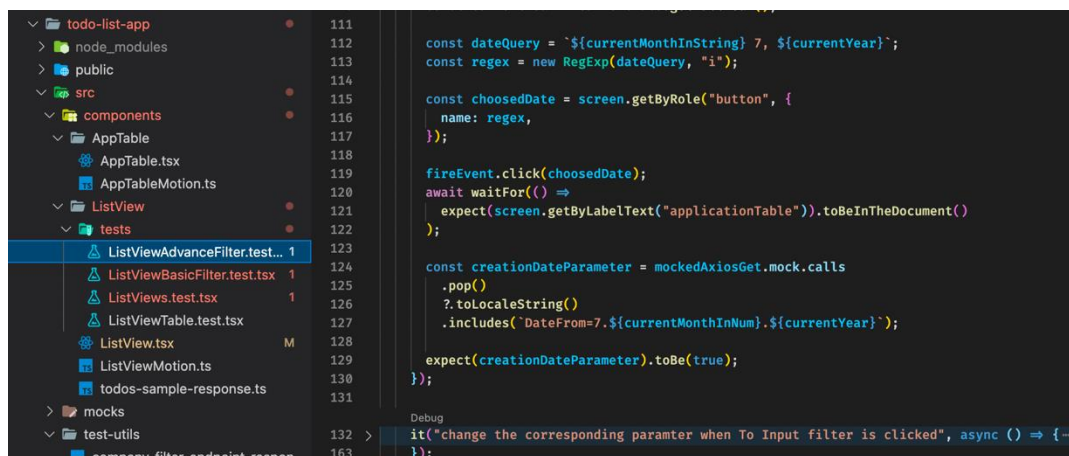


Figure 54. ListView test folder of the application.

Isolation in testing is also a highlighted criterion, the test needs to be independent from external factors so that the problem can be easily found without unnecessary consideration. E.g., for fetching test, initially the fetched data need to be a mocked data, which mean when the test is mimicking the fetching process, it is not actually make a request to the server, but instead getting back the mocked data. With this approach, developer or tester can easily exclude the idea of inappropriate format from the data.

```

jest.mock("axios");
const mockedAxios = mocked(axios);
const mockedAxiosGet = mocked(mockedAxios.get);

```

Figure 55. Example of mocking data in test.

Another test was the UI test, the application must follow “Mobile first” convention, means that it needs to have a friendly UX/UI on all platform and different screen. The requirements from this kind of test are that the application needs to display all the components and features on the UI. This help developer and tester avoid missing small feature that are needed in the application.

```

describe("when the page is render", () => {
  it("should pop up success toast when the data is successfully returned", async () => {
    mockedAxiosGet.mockResolvedValue(response);
    render(<ListView />);
    await waitFor(() =>
      expect(screen.getByLabelText("applicationTable")).toBeInTheDocument()
    );

    const successfullyToast = await screen.findByText(
      "successfully implemented"
    );

    expect(successfullyToast).toBeInTheDocument();
  });

  it("should pop up error toast when the data is failed to return", async () => {
    mockedAxiosGet.mockRejectedValue(errorResponse);
    const { getByTestId } = render(<ListView />);

    await waitForElementToBeRemoved(() => getByTestId("loading"));

    const errorToast = await screen.findByText("something wrong");

    expect(errorToast).toBeInTheDocument();
  });
});

```

Figure 56. Example of testing UI of the application, a snack bar popup whenever the data is successfully or fail to return.

## 7.2 Test results

As the application is confidential, revealing too much detail about the test cases and results are prohibited, that is why all the test cases will be hidden and the thesis will only exhibit the final result of the application when all the test are run and passed.

```
PASS src/components/ListView/__test__/ListViews.test.tsx
PASS src/components/ListView/__test__/ListViewBasicFilter.test.tsx
PASS src/components/ListView/__test__/ListViewTable.test.tsx (6.078 s)
PASS src/components/ListView/__test__/ListViewAdvanceFilter.test.tsx (13.133 s)
```

Figure 57. Test files in the application

```
Test Suites: 4 passed, 4 total
Tests:       27 passed, 27 total
Snapshots:  0 total
Time:        13.905 s
Ran all test suites.

Watch Usage: Press w to show more.[]
```

Figure 58. All the tests in the application are passed

## 8 CONCLUSION

As the time grows, JavaScript has proved its huge value toward the web development field and React has become the most powerful and high demand front end framework worldwide. React offers a sustainable and easy to improve application, yet still assure a modern and user friendly UX/UI due to the large amount of implicitly React's packages that provides reusable and modern premade component.

This thesis has demonstrated process of creating a todo list application that is used as a real product from a professional company and follows strictly with the convention of application development. All the requirements from the company for the ListView of this todo list were met and the application itself is ready for production. The real challenge from this project was to make all the logic from the filter system works appropriately and smoothly whenever there are any input or parameter from the user. In addition, the challenge also included building the folder structure for the whole project, whether divided a piece of code into a separate component or put functions into utils folder to make the code dry and clean.

Finally, the main goal of this thesis is to efficiently apply React knowledge to build a modern and dynamic web application that can works closely with the external sever and successfully fetches data from database. Moreover, an addition criterion is the application must meet the demand for a modern and user-friendly application that give user enjoyment when using it.



## 9 REFERENCES

/1/ Similartech. React js. Accessed 12.12.2021

<https://www.similartech.com/technologies/react-js>

/2/ Mui. Accessed 1.1.2022

<https://mui.com/>

/3/ Wikipedia. React js. Accessed 1.1.2022

[https://en.wikipedia.org/wiki/React\\_\(JavaScript\\_library\)](https://en.wikipedia.org/wiki/React_(JavaScript_library))

/4/ Pandit, Nitin. 2021. What And Why React.js. Accessed 2.1.2022

<https://www.c-sharpcorner.com/article/what-and-why-reactjs/>

/5/ Wikipedia. TypeScript. Accessed 3.1.2022

<https://en.wikipedia.org/wiki/TypeScript>

/6/ Typescriptlang. Handbook. Function. Accessed 8.1.2022

<https://www.typescriptlang.org/docs/handbook/2/functions.html#void>

/7/ Typescriptlang. Handbook. Classes. Accessed 20.1.2022

<https://www.typescriptlang.org/docs/handbook/2/classes.html>

/8/ Recoiljs. Accessed 6.2.2022

<https://recoiljs.org/docs/introduction/core-concepts#atoms>

/9/ Wikipedia. Node js. Accessed 11.2.2022

<https://en.wikipedia.org/wiki/Node.js>

/10/ Reactjs. Accessed 12.2.2022

<https://reactjs.org/docs/create-a-new-react-app.html>

/11/ What is Use Case Diagram. Accessed 11.4.2022

<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/>