Expertise
and insight
for the future

Thanh Binh Tran

Tooling with React

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

20 February 2020

Author Title Number of Pages Date	Thanh Binh Tran Tooling with React 40 pages 20 February 2020
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Mobile Solutions
Instructors	Petri Vesikivi, Principal Lecturer Nico Rotstein, CTO of Kodit.io

The web has been evolving dramatically in just a few decades. In addition, JavaScript technologies are changing and improving with fast pace. Among them, Angular and React are two of the fastest growing technologies in recent years. This final year project aimed for analyzing the two technologies and choosing the right toolset for develop an internal product for a real estate startup named Kodit.

The thesis introduces key concepts of React and Angular with code examples such as Component, React Hooks, Virtual DOM, Angular Command Line Interface, Angular Service, and so on. Then React and Angular comparison is made based on key characteristics and popularity. The main difference between those two is that Angular is a full-fledged framework and React is a user interface library. With the result of the comparison, the right tool has been picked by Kodit tech team.

The case product for this final year project is named Jorge, which is used by the real estate specialist team at Kodit. Jorge helps to enhance the buying process for the team by providing and analyzing opportunities available on the market. The core technology is React and Flow for type checking. Continuous Integration, Delivery, and Deployment are composed into Bitbucket pipelines, which eases the development process. Moreover, Lean Software Development methodology is used as a guideline.

Jorge is a successful product for Kodit. After using Jorge, the productivity of the specialist team is improved greatly with the result of increasing the number of bought apartments. The application is constantly modified and improved with more requested features such as search functionality and sorting method.

Keywords	React, web development, Angular, component, Lean Software Development, CI/CD

Contents

List of Abbreviations

API	Application Programming Interface
DOM	Document Object Model
CI	Continuous Integration
CD	Continuous Delivery or Continuous Deployment
CLI	Command Line Interface
CSS	Cascading Style Sheets
HOC	Higher-Order Component
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
JSX	JavaScript XML
npm	Node Package Manager
SPA	Single Page Application
URL	Uniform Resource Locator
UI	User Interface
VDOM	Virtual DOM
WWW	World Wide Web
XML	Extensible Markup Language

1 Introduction

Since the first website, the World Wide Web Project, was launched in 1991, the web has been exploding. As of December 2019, there are more than 1.7 billion websites in the world according to internetlivestats.com (Internet Live Stats, 2019). Additionally, in 2019 there are around 4.4 billion active internet users, which accounts for more than half of the world population (Statista, 2019).

Website categories are varied such as blog, ecommerce, education, portfolio, business, entertainment, media, and so on. Nowadays, nearly every company or business has its own website, so that the business looks more professional and legitimate. The company website also acts as a branding tool and a contacting point for the visitors. Furthermore, many companies choose to build their internal tools through web platform instead of desktop because it is easier to access and to develop. Through web platform, the tools can work for every operating system, and users or employees only need a browser to access them.

In the last decade, web technologies evolved greatly in accordance to the exploding of the web. Moving from static websites, there is a need for being more dynamic and more interactive. Modern websites have various interaction with the user and have to display content dynamically. Traditional web technology, which uses HyperText Markup Language (HTML), Cascading Style Sheets (CSS), and vanilla JavaScript, is considered obsolete. Modern web technologies emphasise not only on displaying the content, but also on performance, security, cost and efficiency to develop and maintain.

In the last ten years, web ecosystem has produced many good frameworks and libraries such as jQuery, Ember, Backbone, Meteor, and so on. Among those, React and Angular are two of the most growing projects in recent years (Stack Overflow, 2019).

The final year project explored the two outstanding web technologies Angular and React. Furthermore, the thesis discussed the reason behind choosing one technology over the other to build an internal tool name Jorge for Kodit. Kodit is a real estate startup located in the heart of Helsinki. Its mission is to become the leading instant home buyer in Europe.

Inside Kodit, the specialist team takes charge of searching, analyzing and acquiring suitable apartments. Before Jorge was in use, specialists had to do a lot of manual work; therefore, Jorge was built to enhance the working process of the specialist team. Jorge is a full-stack application, which contains a Python backend and a website in the frontend. Additionally, continuous integration, continuous development, and process development were also discussed in this final year project.

2 Theoretical background

2.1 World Wide Web

In 1989, an engineer of European Center for Nuclear Research (CERN), Tim Berners Lee began to develop a technology called HyperText Markup Language for writing and sharing documents. Furthermore, Tim implemented a communication protocol to manage hyperlinked text documents named HyperText Transfer Protocol (HTTP). These technologies founded the infrastructure for a new information system, the World Wide Web (WWW) (Deitel, et al., 2011).

In 1994, the World Wide Web Consortium (W3C) was established by Tim Berners Lee. The organization primary goal was to carry out the compatibility of web components between different vendors. Nowadays, W3C evolves to be a standards organization for web technologies. W3C standardizes HyperText Markup Language 5 (HTML5), Cascading Style Sheets 3 (CSS3) and the Extensible Markup Language (XML) (Wellens, 2015).

2.2 Frontend ecosystem

Frontend development focuses the client-site i.e. what happens on the browser. The main technologies for frontend development are HTML, CSS, and JavaScript. However, in recent years, the frontend is evolving dramatically that more tools are being invented to handle different parts of the ecosystem.

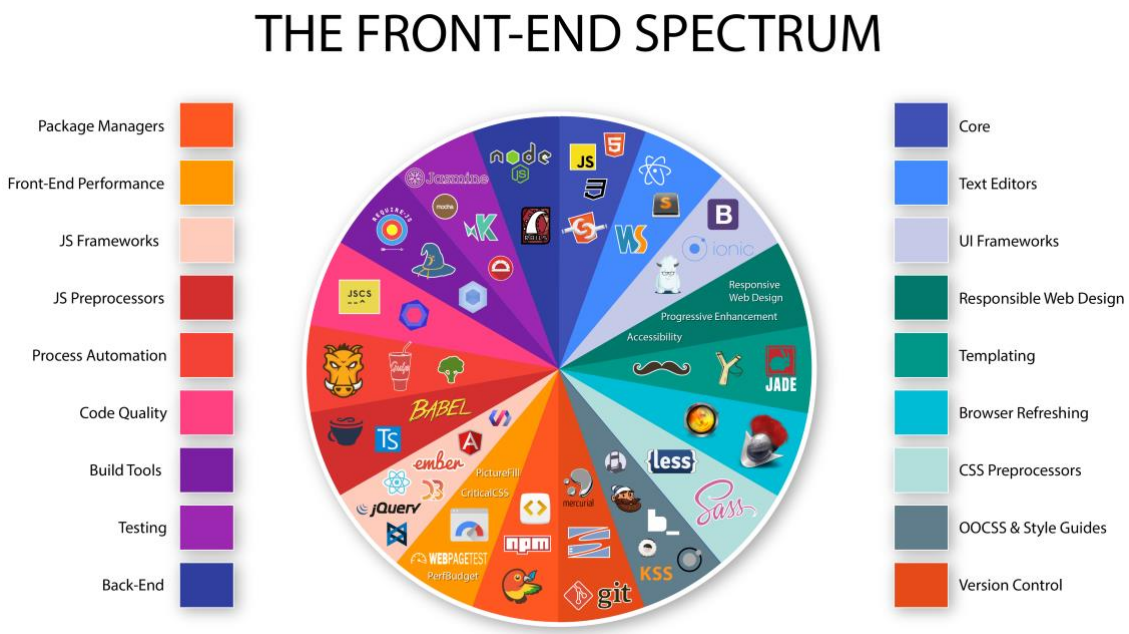

Figure 1 The Frontend spectrum (Pelletier, 2015)

As illustrated in Figure 1, the frontend ecosystem is divided into different sectors, and each sector handles a particular job. Noticeably, in templating for HTML, there are handlebars and Jade. To handle styling, besides the core CSS, there are preprocessors such as Less and Sass. The counterpart preprocessors for JavaScript are CoffeeScript, TypeScript, and Babel. One of the most competitive and evolving sectors is JavaScript libraries and frameworks with prominent representatives such as React, jQuery, Angular, Vue, Ember, D3.
In addition to the core HTML, CSS, and JavaScript, the ecosystem introduces many tools for other aspects. Npm and Yarn are the most notable tools for package managing. Likewise, for code quality, there are ESLint, JSCS, Prettier. Build tool is an essential part in modern web technologies. The most popular build tools are Webpack, Browserify, and Parcel.

2.3 Declarative vs Imperative programming

Eve Porcello and Alex Banks (2017) stated that imperative programming obtains the final results by giving instructions to the application step by step with code. On the other hand, declarative programming structures the application so that it declares the logic rather than define how the logic happens. Take the simple task to make a string to be URL-friendly for example. The following code describes how to gain the result through imperative programming.

```
1.  var string = "This is the midday show with Cheryl Waters";
2.  var urlFriendly = "";
3.
4.  for (var i=0; i<string.length; i++) {
5.    if (string[i] === " ") {
6.      urlFriendly += "-";
7.    } else {
8.      urlFriendly += string[i];
9.    }
10. }
11.
12. console.log(urlFriendly);
```

Listing 1. Manipulate string with imperative approach (Porcello & Banks, 2017)

With imperative approach, every character of the string is looped through and the character is replaced by a hyphen if it is a space. A for loop and an if statements are used to guide the program to accomplish the task. To understand what the code does, it requires a lot of comments in imperative programming. The declarative approach is much simpler to follow:

```
1.  const string = "This is the mid day show with Cheryl Waters"
2.  const urlFriendly = string.replace(/ /g, "-")
3.
4.  console.log(urlFriendly)
```

Listing 2. Manipulate string with declarative approach (Porcello & Banks, 2017)

Examining the code clearly tells the reader what result the program wants to achieve. The program describes that the string should replace space with the hyphen without knowing how the replace function does that.
The code is straightforward and easy to understand using declarative programming. The logic is abstracted to many tiny and well-named functions. Because the code is self-explanatory and easy to follow, it is easier to develop large applications. (Porcello & Banks, 2017)

```
1.  // imperative jQuery
2.  var button = $('<button class="btn red">Change color</button>')
3.  $(button).on('click', function () {
4.    if ($(this).hasClass('red')) {
5.      $(this).removeClass('red')
6.      $(this).addClass('blue')
7.    } else {
8.      $(this).removeClass('blue')
9.      $(this).addClass('red')
10.   }
11. })
12. // declarative React
13. const Button = () => {
14.   const [color, setColor] = React.useState('red')
15.   const handleClick = () => {
16.     setColor(prevColor => prevColor === 'red' ? 'blue' : 'red')
17.   }
18.   return <button className={`btn ${color}`} onClick={handleClick}>Change color
    </button>
19. }
```

Listing 3. Imperative vs declarative programming with web technologies

Listing 3 describes the differences between imperative and declarative approaches in jQuery and React respectively when creating a button that can toggle between red and blue colors. In jQuery approach, the program is instructed step by step from creating a button, to deal with the click handler. In contrast, React declares a button component, in which a color state and a click handler to change the state are defined. The React program does not concern how the button is rendered or how it toggles colors. Not only is the React snippet shorter and easier to understand, the React button version can also be used in multiple places in an application without being declared again.

Declarative programming is becoming the standard of modern web technologies with examples from React and Angular. Imperative programming like using jQuery makes the application difficult to understand and hard to maintain. With declarative approach in modern web technologies, developers can make maintainable and scalable applications.

2.4 React

2.4.1 Overview

React is a front-end library, which was released and maintained by Facebook in 2013. React focuses on the view layer of the application through creating and managing components. Since its release, React has been building a strong position in front-end community. Moreover, not only is React used for developing web applications, React also target other platform such as native mobile and Virtual Reality. (Dresher, et al., 2018)

2.4.2 Virtual DOM

The Document Object Model or DOM, which is constructed by HTML or XML, is the representation of a webpage, and through which programs or scripts can manipulate the page structure, style, or content (Mozilla, 2019). To modify the DOM, JavaScript uses DOM API, for example document.createElement, document.appendChild, and the process is relatively easy (Porcello & Banks, 2017). However, to efficiently manage the page with DOM API and JavaScript is very complex, especially when building a large application. React is one of libraries that provides the solution with virtual DOM.
According to React documentation (2019), the virtual DOM (VDOM) represents a UI, which is stored in the memory. After the state of the UI changes, the VDOM is synced with the real DOM through reconciliation process. Developers can work with declarative API of React instead of DOM API and React takes care of DOM manipulation and event handling.

2.4.3 React element

DOM elements are used to construct the browser DOM, so are React elements used to make React VDOM. React elements are different from browser DOM elements as they are immutable plain objects and updating objects is much faster than directly updating the DOM using DOM API (Porcello & Banks, 2017).

Figure 2 Relationship between createElement and the DOM element. Copied (Porcello & Banks, 2017)

As demonstrated in Figure 2, properties of React element are added to the DOM tag as attributes, and the child text is added as the Node Text of the DOM element.

2.4.4 JSX

Facebook's JSX Specification (2014) states that "JSX is an XML-like syntax extension to ECMAScript without any defined semantics." JSX is an alternative and recommended way to write React elements. Code written in JSX is transformed to plain JavaScript by a compiler such as Babel. The purpose of JSX is to give a simpler syntax for building a complex DOM tree and to make it more readable like HTML (Porcello & Banks, 2017).

```
1.  // without JSX
2.  React.createElement("ul", {"className": "list"},
3.    React.createElement("li", {"className": "list-item"},"Item text"),
4.    React.createElement("li", {"className": "list-item"},"Item text"),
5.    React.createElement("li", {"className": "list-item"},"Item text"),
6.    React.createElement("li", {"className": "list-item"},"Item text"),
7.    React.createElement("li", {"className": "list-item"},"Item text"),
8.    React.createElement("li", {"className": "list-item"},"Item text")
9.  );
10.
11. // JSX
12. <ul className="list">
13.   <li className="list-item">Item Text</li>
14.   <li className="list-item">Item Text</li>
15.   <li className="list-item">Item Text</li>
16.   <li className="list-item">Item Text</li>
17.   <li className="list-item">Item Text</li>
18.   <li className="list-item">Item Text</li>
19. </ul>
```

Listing 4. React with and without JSX

Listing 4 outlines the difference when writing React using JSX and without it. JSX makes the code cleaner and straightforward to read; therefore, it helps the maintenance job easier. The developer can write deeply complex nested DOM structure without readability because JSX helps composing the code similar to HTML.

2.4.5 Component

A component is a piece, big or small, of the application UI, for example a navigation bar, a header, or a section. Components are made of elements. Breaking down the UI into smaller components helps to develop faster, maintain easier, and be able to reuse the code better (Facebook, 2019).

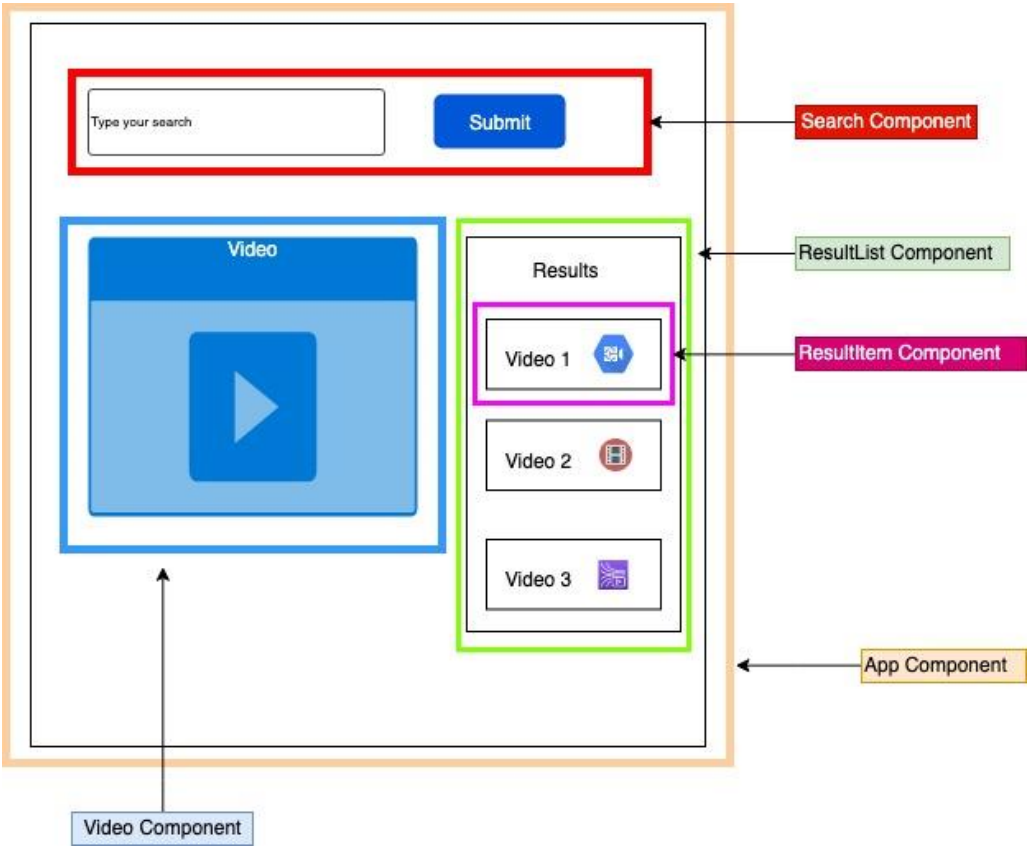

Figure 3 Breaking down UI into components

Figure 3 draws a UI of a simple application which takes a user input and displays video search results. The UI is divided into four big components:

- App Component is whole application.
- Search Component takes user search input.
- ResultList Component displays the list of search results.
- Video Component contains the video player.

Additionally, some components can be split to even smaller components for reusability. For instance, the Search Component consists of a text input and a button, both of which is a component and can be reused in other places of the application.
There are two types of component: functional and class.

2.4.6 Hooks

React 16.8 introduced Hooks, which provide state and other features to functional component. Abramov (2018) stated that building components with Hooks is effortless and more user friendly. React provides various built-in Hooks including useState and useEffect which help declare state and side effects for functional components. All use cases for class component can be covered by Hooks but Hooks also enables code reusability, extractability, and testing with more flexibility. Therefore, Hooks serve as the future vision of React. The case project in this thesis was composed using Hooks for handling state instead of using class components.

In general, there are two strict rules of Hooks:

- Only call Hooks at the top level. Don't call Hooks inside loops, conditions, or nested functions.
- Only call Hooks from React functions. Don't call Hooks from regular JavaScript functions.

2.4.7 Context API

React is a state management library itself and context is one method for doing so. React has a top-down data passing mechanism from parent to child with props which is straight forward, but the mechanism also creates issues like prop drilling. The prop drilling issue occurs when developers must pass or drill props deeply to multiple layers of components, which causes difficulties to refactor. React context helps managing state and sharing data easily without explicitly passing a prop through every level of component tree (Facebook, 2019).

2.4.8 React Router

Historically, website routing is mostly implemented in the server where series of files that represent web pages are stored. When a user navigates to a web page, the server returns the file in that location which is reflected in the browser's location bar.
Modern web technology introduces single page application in which JavaScript loads data and changes the UI without the need of server-side routing. In React ecosystem, React Router is the most well-known solution for client-side routing for React applications (Porcello & Banks, 2017).

2.5 Angular

2.5.1 Overview

Angular is an open source project and a framework released by Google in 2016 for building Single Page Applications (SPA). The Angular framework is fully re-written from AngularJS or Angular 1 by the same team from Google. In addition to web, applications for multiple platforms such as mobile and desktop can be developed with Angular. (Bodrov-Krukowski, 2018)

Angular has a related product named Angular CLI. Angular CLI is a command-line tool developed by the Angular team that helps developers from getting rid of the burden when setting up a new Angular project. A project generated by Angular CLI has necessary building blocks such as development server, test runner, Webpack, linting, and build steps.

2.5.2 Component

A component defines a piece of UI on the screen of an application such as a tool bar with navigation links, a list of items, an input. An Angular component is a JavaScript or TypeScript class decorated by @Component decorator which defines metadata for the component. The metadata can comprise a template to define the view, a selector for invoking the component in a HTML template, style URLs to apply styles to the HTML view, and providers to supply services.

```
1.  @Component({
2.    selector:    'app-hero-list',
3.    templateUrl: './hero-list.component.html',
4.    providers:  [ HeroService ]
5.  })
6.  export class HeroListComponent implements OnInit {
7.  /* . . . */
8.  }
```

Listing 5. Angular component example (Google, 2019)

As illustrated in Listing 5, a @Component decorator is put before the HeroListComponent class definition to supply the component with the necessary metadata such as selector, templateUrl, and providers. Only classes decorated by @Component decorator like above become an Angular component.

2.5.3 Service

In an ideal circumstance, a component should only handle user experience and act as a mediation between the view and the application logic by providing properties and methods for data binding. Additionally, a component can make use of services to handle tasks such as data fetching, input validating, or

console logging. Those services are injectable through dependency injection provided by Angular, so that they are available to the whole application.

Dependency injection is a programming design pattern which allows a class utilizing services outside its scope by injecting them as dependencies. Dependency injection enhances the flexibility, efficiency, testability, maintainability, and modularity of an Angular application (Google, 2019).

2.6 React and Angular comparison

2.6.1 Key characteristics

React and Angular are two of the most powerful technologies for developing the frontend. Angular is a framework which provides a full-fledged toolkit for creating an application. Packages for http request, routing, templating and so on are available within the Angular core framework. One of the biggest advantages of Angular is consistency in different projects because all tools are already built for developers to use and Angular defines how developers structure the project. On the other hand, it is difficult to include a new library into an Angular application because the library must be TypeScript compatible and wrapped into a service in order to communication with components. Moreover, Angular is opinionated which means there is an Angular way for an implementation.

Unlike Angular, React is a library which focuses on the view layer of the application. Because of its concentration on building the UI, React needs other libraries to develop a complete application. Due to that fact, importing and using a third-party library in React is extremely easy through npm. React has a large ecosystem in which provides multiple solutions for different layers of an application such as routing, data fetching, state management. Therefore, React offers flexibility for developers to structure and develop applications.

2.6.2 Popularity

React and Angular are big projects and are widely popular among open-source community. There are many factors for popularity comparison such as trending, usage, community growth and so on.

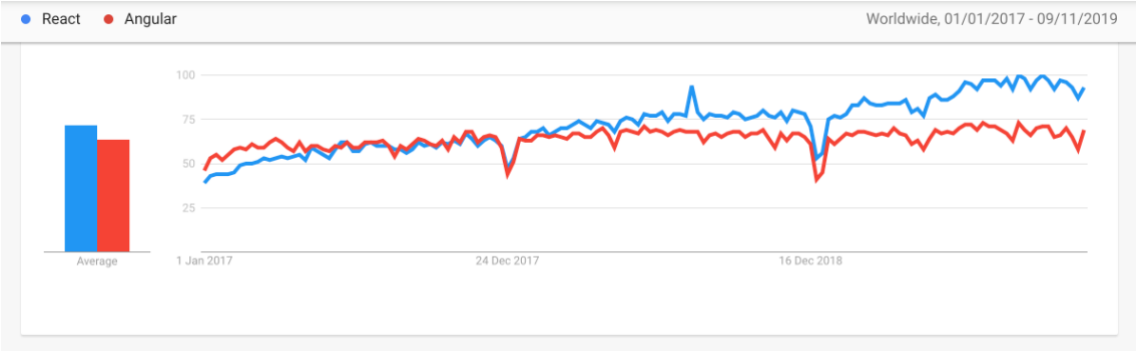

Figure 4 Interest over time React vs. Angular from January 2017 to November 2019 (Google Trends, 2019)

As illustrated in Figure 4, which shows the trending of React and Angular on Google search engine over the past three years, at beginning of 2017 Angular was in front; however, from mid-2017 React has been consistently overcoming Angular. Moreover, React enlarges the gap gradually over time. The trending on Google partially proves that there are more and more people interested in React over time.

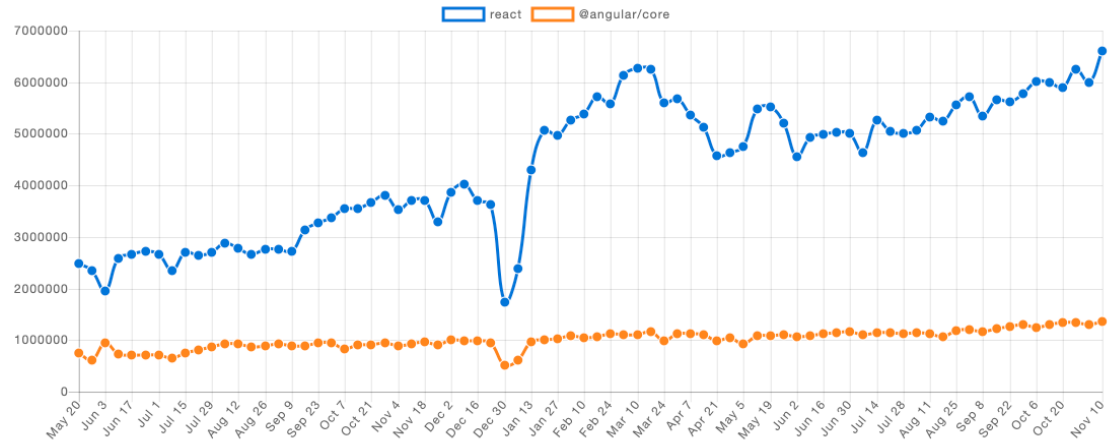

Figure 5 Weekly downloads React vs. Angular from May 2017 to November 2019 (npm trends, 2019)

According to npmtrends.com, the downloads of React and Angular has a significant gap. Over the past two years, Angular was downloaded around one million times weekly, and the number is steadily doubled from 750 000 in May 2017 to 1350 000 in November 2019. On the other hand, from May 2017 React already had more than two million weekly downloads. The number is tripled in two years to more than six million at the end of 2019. The weekly downloads of both Angular and React increase significantly over two years; however, React has shown to have a larger growth trajectory.

2.6.3 Popularity through web survey

Web survey is a powerful tool to measure the attitude of developers toward tools and technologies. Stack Overflow's annual Developer Survey is one of the best surveys for technologies in general.

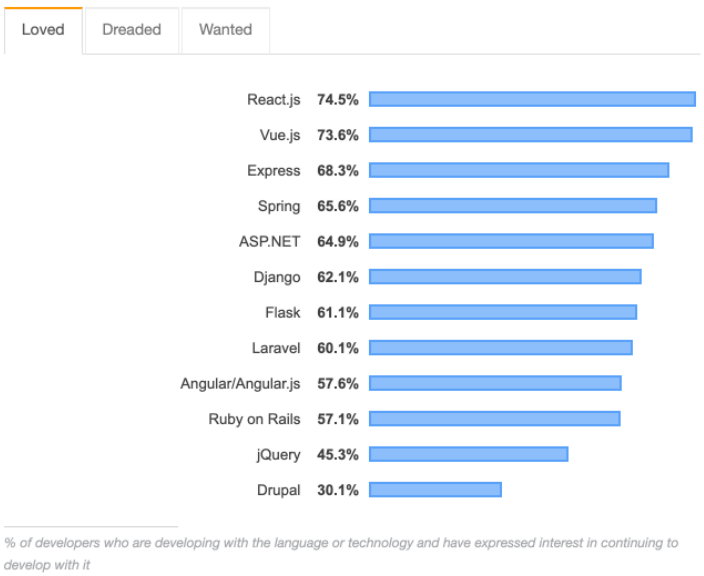

Figure 6 Most loved technologies (Stack Overflow, 2019)

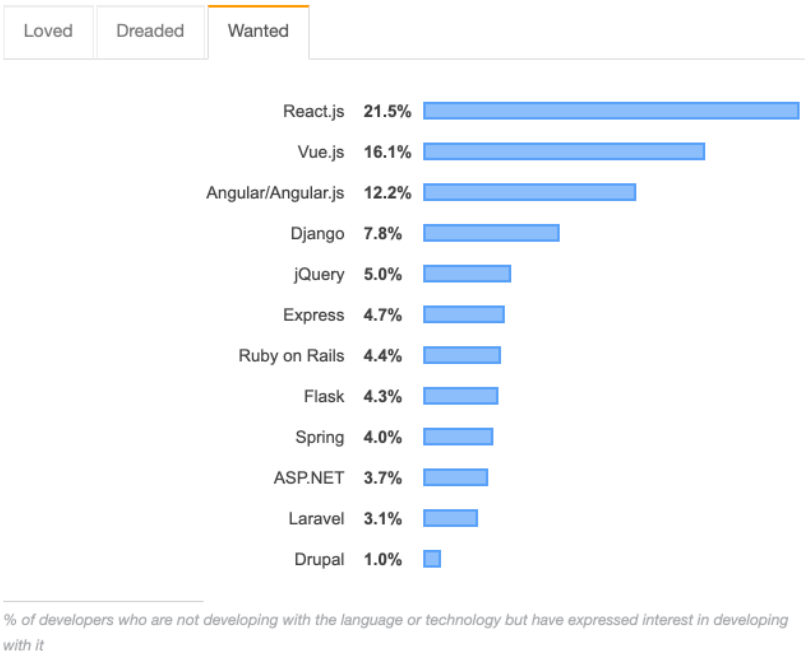

Figure 7 Most wanted technologies (Stack Overflow, 2019)

As shown in Figure 6 and Figure 7, there is a positive tendency toward React with the library tops the most loved and most wanted technologies tables. Three in four developers express a huge interest for React as they want to continue developing using the library. In contrast, only over half of the developers have the same expression for Angular.

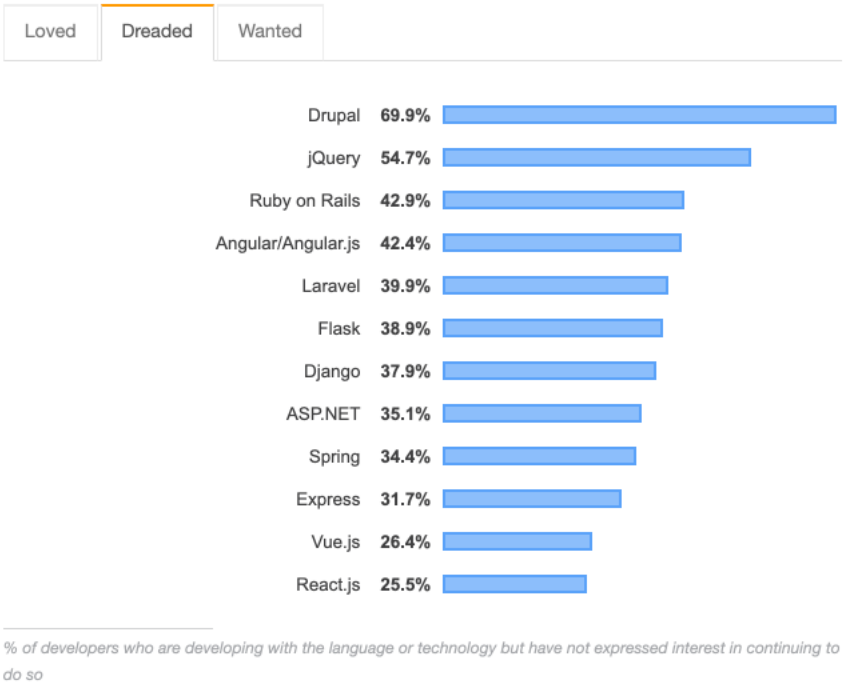

Figure 8 Most dreaded technologies (Stack Overflow, 2019)

In the opposite survey for the most dreaded technologies, Angular are forth in the table with 42.4% of developers consider not to continue using Angular for their future development. The number for React is 25.5% which is much lower.

Stack Overflow surveys shows that developers have a better attitude toward React than Angular. However, Angular is still one of the most wanted technologies along with React and Vue in 2019.

2.6.4 Conclusion

Both React and Angular are extremely famous and widely used technologies for building web applications. However, React is leading in terms of community growing and positive attitude from developers. Kodit, as a startup, firstly wants to develop and release products fast. Secondly, Kodit needs to attract wider range of frontend developers in order to quickly grow. With an easier learning curve, and a larger community, React is a better solution for Kodit than Angular. Therefore, at Kodit, the tech team decided to use React for building internal tools as well as the company website.

3 Application requirements and project setup

3.1 Background

The application was carried out on the request of acquisition specialist team at Kodit and was named Jorge Dashboard or Jorge in short. The job of the acquisition team is to take part in different process such as looking for opportunities, booking inspections, making offers in order to acquire good apartments for Kodit and its investors. Before the Jorge Dashboard, selected opportunities were sent to specialists' emails by the Opportunity Bot every day in the morning. The specialist then went through the email, made call to the respective selling contacts, took notes and handled opportunities by herself. This process led to inefficiency. There was needs for a tool to manage the opportunities more effectively and the Jorge Dashboard was born from that reason.

3.2 Requirements

Jorge Dashboard was developed for in-house users, mainly real estate specialists, who are going to buy apartments for Kodit. The application is a minimal viable product, which replaced the old Opportunity Bot. Jorge is a web application which supports desktop and tablet screen with latest Chrome version. The application did not support mobile initially; however, this may change in the future. Addition to the frontend web application, there is a backend, which handles data processing called Opportunity API.

Jorge Dashboard needs to show the list of opportunities from Opportunity Bot, in which new ones come every morning. A specialist can click to discard an opportunity in the list and select a discarding reason. An opportunity can be marked as contacted when the specialist makes the call and books the inspection. Qualified opportunities can be assigned to a specialist and she take care of the opportunity. Furthermore, the specialist should have the ability to add notes to an opportunity such as "Called several times and didn't reply yet." Finally, Opportunities stay in the main unassigned opportunities list until they are assigned or discarded.

3.3 Architecture

The Jorge Dashboard replaced the Opportunity Bot emails with the mission to ease the buying process of the specialist team. It comprised of two parts: frontend application and backend APIs. I developed the React web application while another developer was in charge of the backend. The application flow is illustrated in Figure 9.

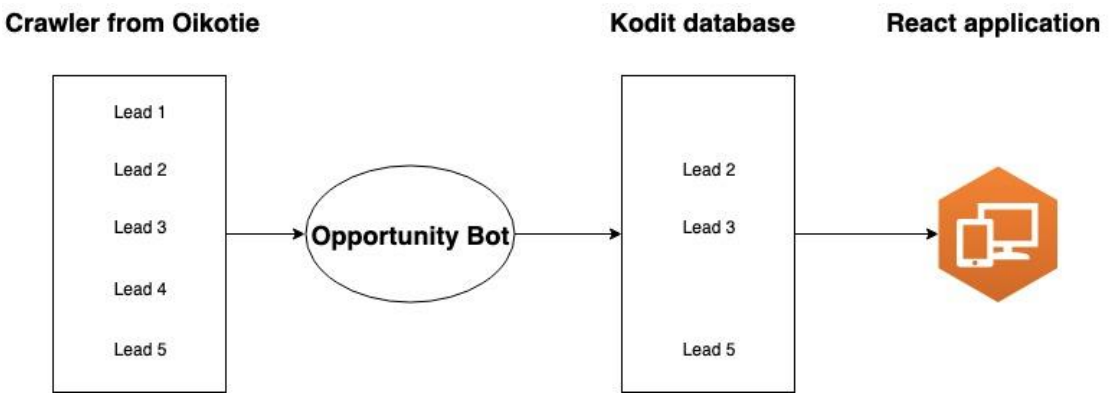

Figure 9 Jorge Dashboard application flow

Firstly, crawler scrapes information of new leads from Oikotie every day. Then the Opportunity Bot picks out the new leads and filters them as shown in Figure 9. Leads are selected based on various criteria, such as apartment size, postal code, Kodit target area, upcoming renovation, condition, and so on. Moreover, Kodit machine learning model generates predicted price for those selected leads, which are then persisted to Kodit opportunity database. Finally, the backend exposes APIs for the React application to consume.
Jorge Dashboard displays relevant information about the leads. With each lead, a specialist can perform different actions, for instance assign the lead to oneself, mark the lead as handled, discard the lead with a reason, leave a note. Additionally, the application provides filters for respective actions being unassigned, assigned, handled, discarded.

3.4 Tools and Technologies

3.4.1 Create React App

Create React App, which is a tool developed by Facebook helps developers to concentrate on coding and developing applications because it bootstraps all necessary build tools needed for an React application. Build tools such as ESLint, Babel, Webpack and so on are configured so that they work seamlessly under the hood without mismatching versions. Moreover, if there is a need for more customized or advanced configuration, Create React App allow user ejecting and directly modify config files (Facebook, 2019). Create React App is a right tool for this project since I can start developing at the beginning without any tooling configurations.

3.4.2 Continuous Integration, Delivery, and Deployment

Nowadays software requirements and technologies are changing frequently, so that companies need to adjust themselves to the needs, develop and deliver products rapidly. The old methods of delivering software, in which customers have to wait more than half a year for a new version of a software to be released and the company gathers the feedback and evaluates the software after that is outdated. The software needs to evolve continuously to meet the customers' expectations. Therefore, it is compulsory for companies to adapt an agile way in a software development cycle to cut short the feedback time and to

deliver the software continuously (Virmani, 2015). To advance the software development and delivery without decreasing quality, continuous practices namely Continuous Integration (CI), Continuous Delivery (CD), and Continuous Deployment (CD) are used. The relationship between mentioned continuous practices is described in Figure 10.

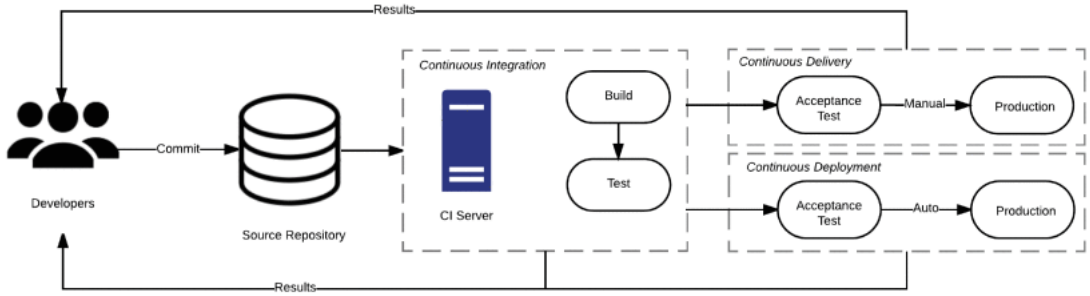

Figure 10 The relationship between continuous integration, delivery and deployment (Copied from (Virmani, 2015))

Continuous Integration requires developers to commit code change to the central repository and validate the code behavior frequently, for example multiple times per day. Each time a commit happens, the CI server triggers the tests and makes a production build. After passing the tests and quality checks, Continuous Delivery ensures the build is deployed automatically to staging, a production-like environment. In staging environment, the software can be tested by users and feedback is generated faster, which reduces deployment risks and costs. In addition, as sketched in Figure 10 Continuous Deployment is similar to Continuous Delivery, but the practice goes deeper and deploys the application automatically and continuously to production environment. (Shahin, et al., 2017)

3.4.3 Static type checking with Flow

JavaScript is a dynamic-typed programming language, which performs type checking at runtime. JavaScript implies the type of variables; whereas, in static-typed languages, the type of a variable has to be declared when it is constructed. Even though it is very convenient to write a program in dynamic-typed JavaScript, it is not optimal. The program can be compiled even if type errors exist and cause buggy behaviors (Kasireddy, 2016). The final year project used Flow, an open-source project released by Facebook, as a type checker.

3.5 Development Process

At Kodit, the team applies Lean Software Development methodology in the development process. Lean Software Development methodology is adopted from lean product manufacturing principles, with which Honda and Toyota successfully utilized for their automobile production process in the 1980s. According to Tom and Mary Poppendieck in their book Lean Software Development: An Agile Toolkit (2003), there are seven principles in lean development:

- Eliminate waste. Waste is everything which do not bring value for the customer, for example: implementing a useless feature, switching tasks, miscommunication, and so on.
- Amplify learning. Software development is a learning process in which the final results are complex and usually have variations. Amplify learning is the best approach for improving a software development environment.
- Decide as late as possible. Decisions are based on fact and not speculation; therefore, delaying decisions until a predictable future is valuable.
- Deliver as fast as possible. Delaying decisions capability is not possible without speed. Delivering fast brings reliable feedback and helps the team to learn more.
- Empower the team. Developers know best the details of the software, thus letting them take part in the technical decision process is essential for success. The core idea is to respect people and acknowledge their work
- Build integrity in. Software needs to adapt to changes and maintain its usefulness. Attributes of software with integrity are usability, maintainability, adaptability, and extensibility.
- See the whole. The product of the software system is bigger than the sum of its components.

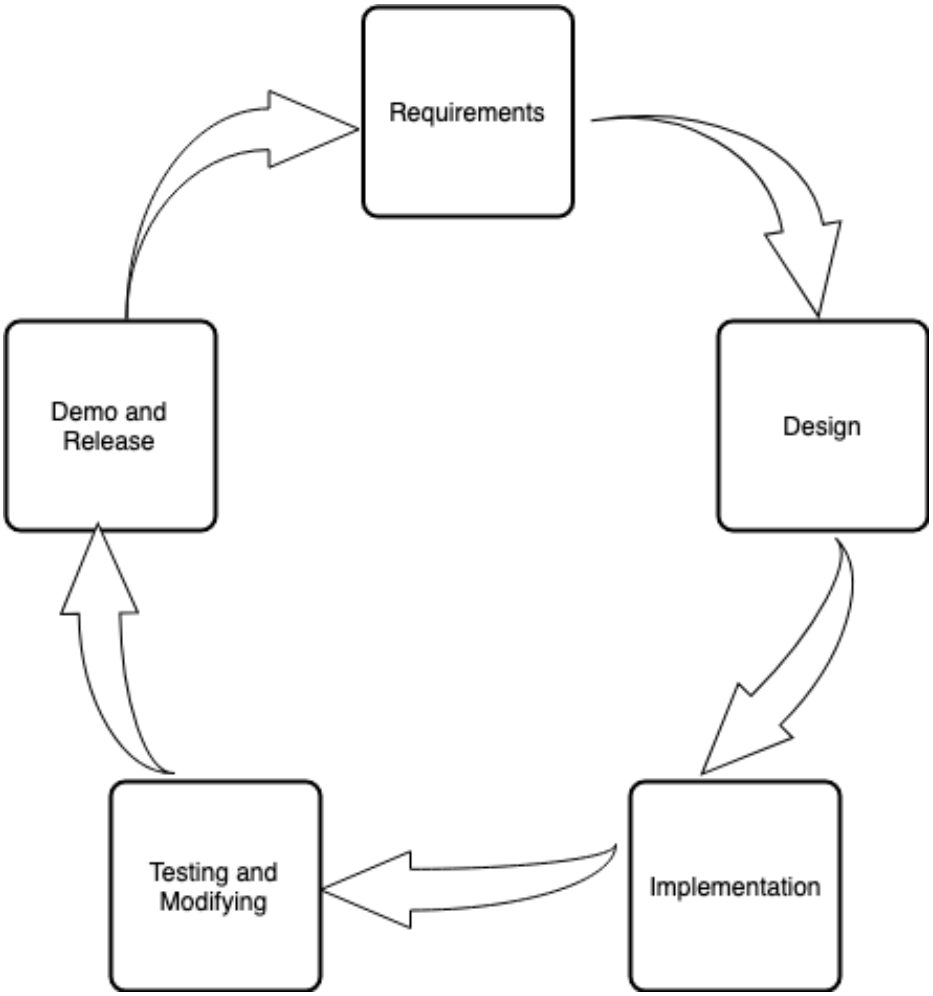

Figure 11 Jorge sprint development cycle

As seen in Figure 11, the Jorge project was developed by two-week sprint cycles. At the beginning of a sprint, there was a sprint planning meeting, in which all the parties set up requirements for new features. Before that, the head of product and the specialist team verified the new features for their feasibility and usability due to not developing impractical feature according to the Lean software development principles. Then the design for new features was prepared by the head of product. With all the requirements and design in place, the developer could implement the features and deploy them to the staging environment for the specialist team to test. If there was a need to modify a feature, the development made the change accordingly. At the end of the sprint, there was a demo day to showcase the new features to the company. The demo time was an opportunity for questions relating to the new features. The new version of the application was released to production and a new sprint can start after that. A The retrospect happened at the beginning of the sprint planning meeting, in which each features or goals of the previous sprint were briefly discussed.

3.6 Project setup

3.6.1 Initialize project

Firstly, Create React App tool needs installing to the local machine with the following command:

```
npm install -g create-react-app
```

Listing 6. Command to install Create React App using npm

With Create React App available in the machine, the following command was used to initialize Jorge Dashboard project with zero configuration:

```
create-react-app jorge-dashboard
```

Listing 7. Create Jorge Dashboard project with create-react-app

The jorge-dashboard directory was created in the location where the command run with a React application boilerplate structure. Create React App provides many useful commands to work with the application, for example npm run start to start the development server, npm run test to run all tests for the application.

3.6.2 Linter and formatter

In order to have a universal code style between developers and not to have unnecessary conflicts in the code base, the project used ESLint as a linter, Prettier as a formatter, and Husky to run pre-commit scripts. The goal of ESLint is to ensure consistent code and to avoid bugs by identifying and reporting JavaScript patterns which is defined by a set of rules (ESLint, 2019). Prettier is described as “an opinionated code formatter”, which supports various programming languages such as JavaScript, HTML, CSS, YALM (Prettier, 2019). Jorge Dashboard React application forced Prettier to use ESLint configuration rules when possible, so that the tools did not overlap. Additionally, to guarantee all developers to have the same code style when committing the code to the central repository, the project utilized Husky to run a script which checked and fixed all ESLint rules and formatted the code using Prettier for changed files before a developer can run the commit command.

3.6.3 Deployment pipelines

To begin with, a bitbucket-pipelines.yml was created at the root directory of the repository for CI/CD tasks. Pipelines are run on Bitbucket Cloud environment inside a Docker container. Jorge's pipelines were defined as below:

```
pipelines:
  default:
    - step:
        name: Build & Lint
        image: koditio/bb-node8-aws:1
        caches:
          - node
        script:
          - npm install
          - npm run build
    - step:
        name: Deploy to staging
        deployment: staging
        trigger: manual
        image: koditio/bb-node8-aws:1
        caches:
          - node
        script:
          - npm install
          - npm run deploy:staging
    - step:
        name: Deploy to production
        deployment: production
        trigger: manual
        image: koditio/bb-node8-aws:1
        caches:
          - node
        script:
          - npm install
          - npm run deploy:production
```

Listing 8. Jorge bitbucket-pipelines.yml

As shown in Listing 8, three steps were constructed for Jorge:

- Build and Lint: test to run the build script. This step is run automatically.
- Deploy to staging: install all dependencies and deploy the project to the staging environment. This step is triggered manually.
- Deploy to production: the same with Deploy to staging step but for production environment. This step is also triggered manually.

3.6.4 Project structure

The structure of the project was divided into sub folders which have their own functionality.

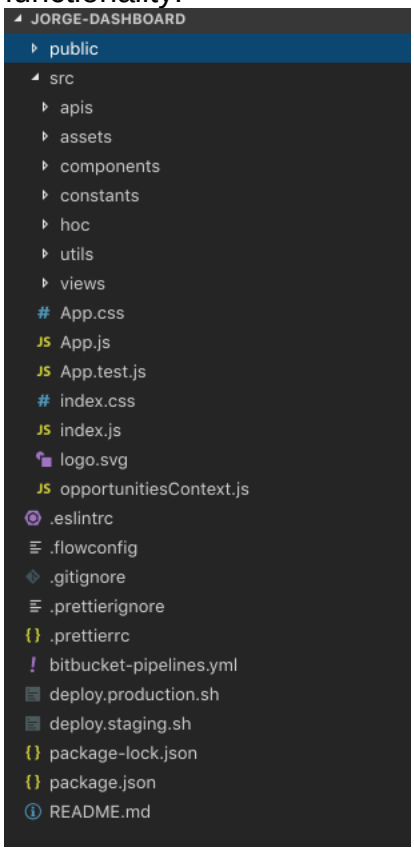

Figure 12 Jorge project structure

As observed in Figure 12, configuration files were placed at the root directory of the project. There are two main sub folders: public and src. The public folder consisted of Kodit icon images and an entry index.html, which was generated by Create React App. Third party library scripts, fonts, and stylesheets can be located there. The src directory made up the main source code of the application.

At the base level, index.js file is the entry of the application, which renders the App component defined in App.js file. App component constructed all routes of the application. Inside the src directory, elements of the application were broken down into sub folders:

- apis: logic of network request
- assets: static assets such as images
- components: reusable React components
- constants: static types, opportunity status, area postcodes
- hoc: Higher Order Components
- utils: utility functions
- views: pages of the application

4 Application implementation

4.1 API client

Network requesting is a critical part in Jorge application because it is the communication bridge between the frontend and the server. Network operations are handled by an API client which typically contains four methods: GET, POST, PUT, and DELETE. Jorge Dashboard only displays and updates the resources, hence GET and PUT methods were implemented.

```
1.  export const get = async (
2.    url: string,
3.    headers: Object = getDefaultHeaders()
4.  ) => {
5.    const response = await fetch(url, {
6.      headers,
7.      mode: 'cors',
8.    })
9.    if (!response.ok) {
10.     throw new Error(`HTTP error, status = ${response.status}`)
11.   }
12.   return response.json()
13. }
14.
```

Listing 9. API client – get function

The API client is described in Listing 9. Both get and put functions take the request endpoint URL as the first parameter and the optional headers as the last one. The default headers handle configurations such as the Content-Type and the Authorization, which attaches the token from the browser's localStorage to authorize the request. Additionally, the put function accepts the body parameter, which holds the data to update the resource. The functions throw an error if the request fails and return the response JSON if the request succeeds. In order to manage the project efficiently, resources are separated into several files, comprising:

- opportunityResources.js: all resources related to opportunity leads
- insightResources.js: lead images from Oikotie, price prediction
- miscResources.js: geocoding resource

4.2 Authentication with Higher Order Component

The authentication service was provided by another application from the same domain. When logged in, user information, which includes email, name, token, and its expiry were stored in the LocalStorage. Because Jorge and the authorization service provider were hosted on the same domain, Jorge can access all the information in the LocalStorage.

```
1.  const withAuth = WrappedComponent => {
2.    class WithAuth extends Component {
3.      state = {
4.        authenticated: false,
5.        user: null,
6.      }
7.
8.      componentDidMount() {
9.        if (isAuthenticated()) {
10.         this.setState({
11.           authenticated: true,
12.           user: getUser(),
13.         })
14.       } else if (process.env.NODE_ENV === 'production') {
15.         window.location.href = `${window.location.origin}/?redirect=jorge/`
16.       }
17.     }
18.
19.     render() {
20.       const { authenticated, user } = this.state
21.
22.       return authenticated ? (
23.         <WrappedComponent {...this.props} user={user} />
24.       ) : (
25.         <div>Not authenticated</div>
26.       )
27.     }
28.   }
29.
30.   WithAuth.displayName = `WithAuth(${getDisplayName(WrappedComponent)})`
31.
32.   return WithAuth
33. }
```

Listing 10. withAuth HOC

The project used the withAuth HOC to handle authentication. The function checked the availability of the token and its expiry. If both requirements pass, the user is logged in. According to Listing 10, if the user is authenticated, the withAuth function return the input component with user information as a prop. Otherwise, the user is redirected to the authentication service, which is the root domain with a redirect parameter to signal the redirection to Jorge application after the user logs in.

4.3 Using context to share data

As discussed in section 2.4.7, context is a method to share data across a React application. Jorge used context API as a global data store, which then can be passed to different views or components.

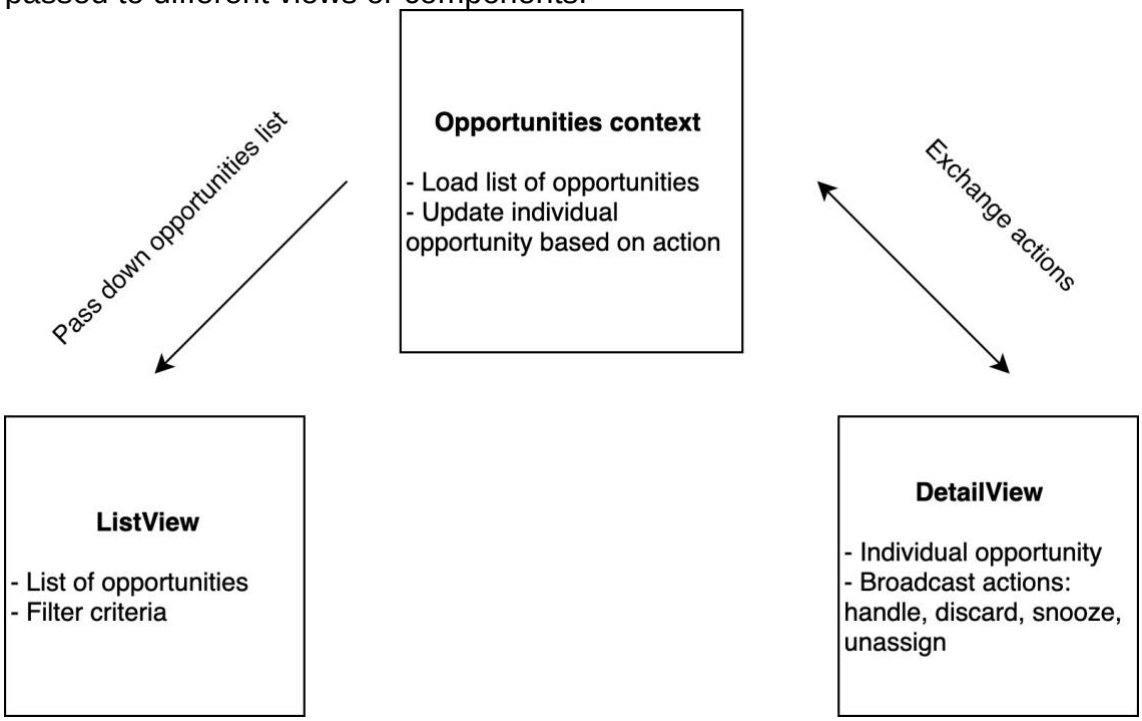

Figure 13 Opportunities context

As depicted in Figure 13, Jorge has a central store, the opportunities context which manages the list of opportunities as well as its individual. The list is passed down to ListView or the main Dashboard, so that opportunities are reflected in the UI. Moreover, the DetailView at the same time is able to update individual opportunity by triggering actions.

```
function useOpportunities() {
  const context = useContext(OpportunitiesContext)
  if (!context) {
    throw new Error(
      'useOpportunities must be used within a OpportunitiesProvider'
    )
  }
  const { opportunities, setOpportunities, error, loaded } = context
  const updateSingleOpp = (
    id: string | number,
    updateOptions: {
      status: string,
      assigned_to: string,
    }
  ) => {
    setOpportunities(opps =>
      opps.map(opp => {
        if (Number(opp.opp_id) === Number(id))
          return { ...opp, ...updateOptions }
        return opp
      })
    )
  }

  return {opportunities, setOpportunities, updateSingleOpp, error, loaded}
}
```

Listing 11. useOpportunities custom Hook

Jorge utilized a custom Hook to manage the opportunities context. Listing 11 shows that the useOpportunities Hook handled the OpportunitiesContext object by firstly throws an error if the context usage is not inside the OpportunitiesProvider (the OpportunitiesProvider will be discussed next). Then the Hook provided a function, updateSingleOpp for the Detail view to update its current opportunity. The updateSingleOpp finds in the list of opportunities in the context the opportunity which needs updating, then it updates the opportunity with provided options. Finally, updateSingleOpp calls setOpportunities function from the context to update the new list of opportunities. React will update the UI of any component that utilizes the opportunities from the context when setOpportunities function is called.

```
function OpportunitiesProvider(props: Object) {
  const [opportunities, setOpportunities] = useState([])
  const [loaded, setLoaded] = useState(false)
  const [error, setError] = useState(null)

  useEffect(() => {
    const fetchOpportunities = async () => {
      try {
        const data = await getOpportunities()
        data.sort((a, b) => {
          return new Date(b.ad_date) - new Date(a.ad_date)
        })
        setOpportunities(data)
        setError(null)
        setLoaded(true)
      } catch (e) {
        setError(e.message)
        setLoaded(true)
      }
    }
    fetchOpportunities()
  }, [])

  const value = useMemo(() => {
    return { opportunities, setOpportunities, loaded, error }
  }, [opportunities, loaded, error])
  return <OpportunitiesContext.Provider value={value} {...props} />
}
```

Listing 12. OpportunitiesProvider

OpportunitiesProvider has the responsibility to load the list of opportunities and to provide necessary values for its consumers. As written in Listing 12, the React useState Hook was used to store the state such as opportunities list, loaded status, error message. Moreover, the opportunities fetching was implemented inside a useEffect Hook, which performs side effect for functional components. Loaded, error, and opportunities states are set according to the fetching status. For example, the opportunities and loaded states are set when the fetching succeeds, otherwise the error state is set if there is a failure.

```
<Route
  path="/opportunities"
  render={props => (
    <OpportunitiesProvider>
      <Dashboard user={user} {...props} />
    </OpportunitiesProvider>
  )}
/>
```

Listing 13. Usage of OpportunitiesProvider

The OpportunitiesProvider wraps any component which uses its context values. As shown in Listing 13, the Dashboard or ListView component is wrapped by the OpportunitiesProvider, so that the component has access to the opportunities list and other values in the context. The usage of context values will be discussed in subsequent parts of the thesis.

4.4 Routing

There are two routes in Jorge Dashboard application:

- /opportunities for Dashboard or ListView
- /opportunities/:id DetailView

Listing 13 in the previous section introduced the declaration of the first route in the App component. The declaration of a Route simply provides the path where the user navigates in the browser and the component to be rendered.

```
1.  <Route
2.    path={`${match.url}/:id`}
3.    render={routeProps => <Detail user={user} {...routeProps} />}
4.  />
```

Listing 14. /opportunities/:id route

The second route, which is the route for showing individual opportunity detail is declared inside the Dashboard component of the first route because it is a nested route. A nested route is where the component can be rendered additionally to its parent. In Jorge application, when the user clicks on an individual opportunity, the Detail view for that opportunity is displayed and the Dashboard view remains visible. The implementation allows the user continuing browsing the opportunities list where she left before she enters the current opportunity Detail view.

4.5 Dashboard view

The Dashboard is the entry point of the application after the user is logged in. It shows the list of opportunities and provides different filters for users.

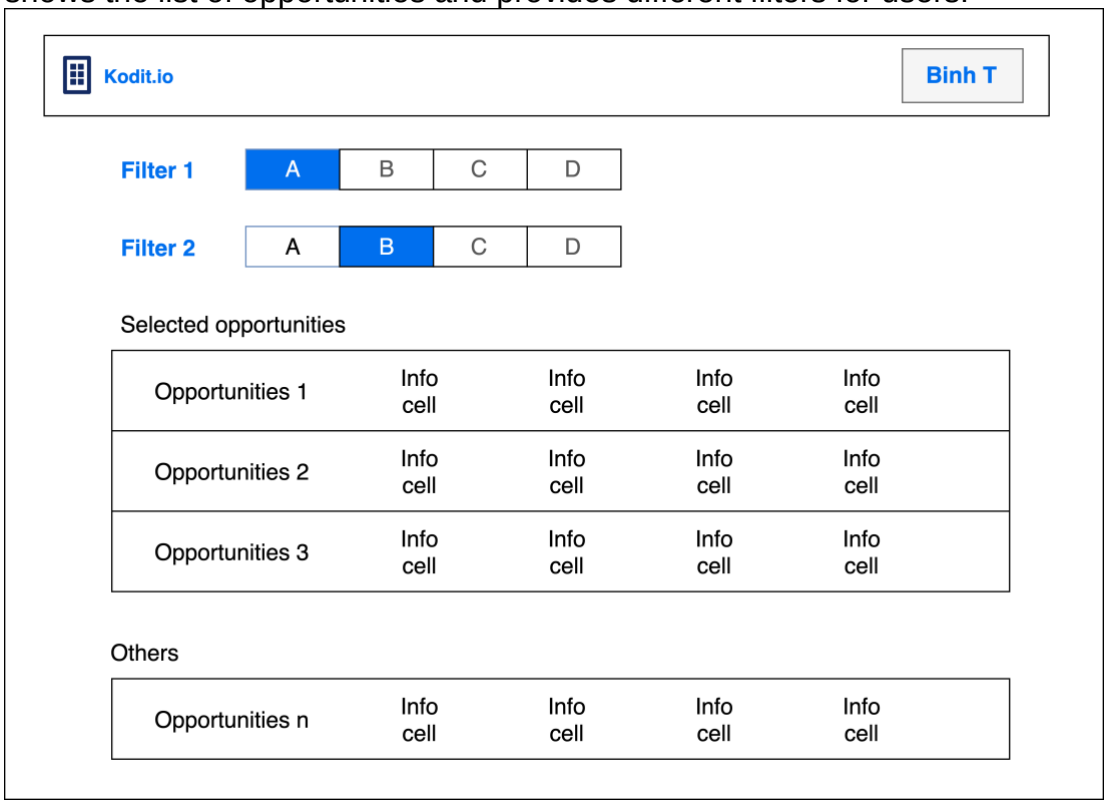

Figure 14 Dashboard view blueprint

Figure 14 displays the draft mockup about how the entry view of Jorge application looks like. The Dashboard view is composed of a navigation bar which displays the user's name, filter sections, and the opportunities lists. The opportunities lists are presented according to the two filters above. Each opportunity consists of information of the apartment such as address, size, condition, built year, price and so on.

4.5.1 Data and filters

The data of the Dashboard view was passed from the Opportunities context. To access the data, the Dashboard view made use of the useOpportunities custom Hook as described in the following Listing.

```
1.  const Dashboard = ({ user, match }: Props) => {
2.    const [filterStatus, setFilterStatus] = useState(STATUS_UNASSIGNED)
3.    const [filterArea, setFilterArea] = useState(_getDefaultArea())
4.    const { opportunities, error, loaded } = useOpportunities()
5.  ...
```

Listing 15. Declare filters' initial values and opportunities data

In addition to opportunities data, the useOpportunities Hook provided two variables, error and loaded so that the Detail view could reflect them accordingly to the UI. Listing 15 also declares the initialization of the two filters: filterStatus and filterArea.

```
type Opportunity = {
  opp_id: number,
  created_at: string,
  assigned_to: string,
  ad_date: string,
  address: string,
  href: string,
  condition: string,
  built_year: number,
  . . .
}
```

Listing 16. Opportunity data model

The Opportunity data model in Listing 16 is self-explanatory. There are notable attributes: opp_id, assigned_to, address, status, and selected. First of all, the opp_id attribute is the unique identifier and is used to load full data for each opportunity. The status as the name suggested is the status of the opportunity, including unassigned, assigned, handled, discarded, and snoozed. It is operated with the assigned_to attribute to make the status filter. The selected attribute defines the two list of opportunities selected and others. Finally, the address attribute shows the location of the apartment and is used for the area filter.
The status filter was designed that besides unassigned status, which does not belong to any specialists, the opportunities are filtered by the status and the assigned_to attribute. For example, specialist A has handled opportunities A, D, and G. When he clicks to the tab handled in the status filter bar, only his handled opportunities show, and other specialists' opportunities are filtered out. Furthermore, opportunities are divided into areas by postcode. There are six areas HKI 1, HKI 2, Vantaa +, Espoo +, Tampere, and Turku.

4.5.2 React-virtualized to render large list

At the time the list of opportunities was originally implemented, a problem occurred. When a user switched between filter tab, the list took a long time to render due to the amount of DOM elements the browser had to destroy and create in accordance with thousands of opportunities. After researching different options, the author decided to utilize React -virtualized library to solve the issue. React-virtualized provides components for rendering large list efficiently. Virtualized is an approach that only renders visible rows in a list. The project used List, AutoSizer, CellMeasurer components to render the list of opportunities. List, as the name implied, is the container for the opportunity list. Each row of the list is wrapped with a CellMeasurer component. The CellMeasurer measures the cell's content in advance, before the cell is rendered when visible. Finally, the AutoSizer wraps the content and equips the dynamic height and width for each row.

4.6 Detail view

When the user clicks on an opportunity in the Dashboard, the Detail view is open in the form of a modal. Additionally, the URL in the navigation bar of the browser changes thanks to the nested route.

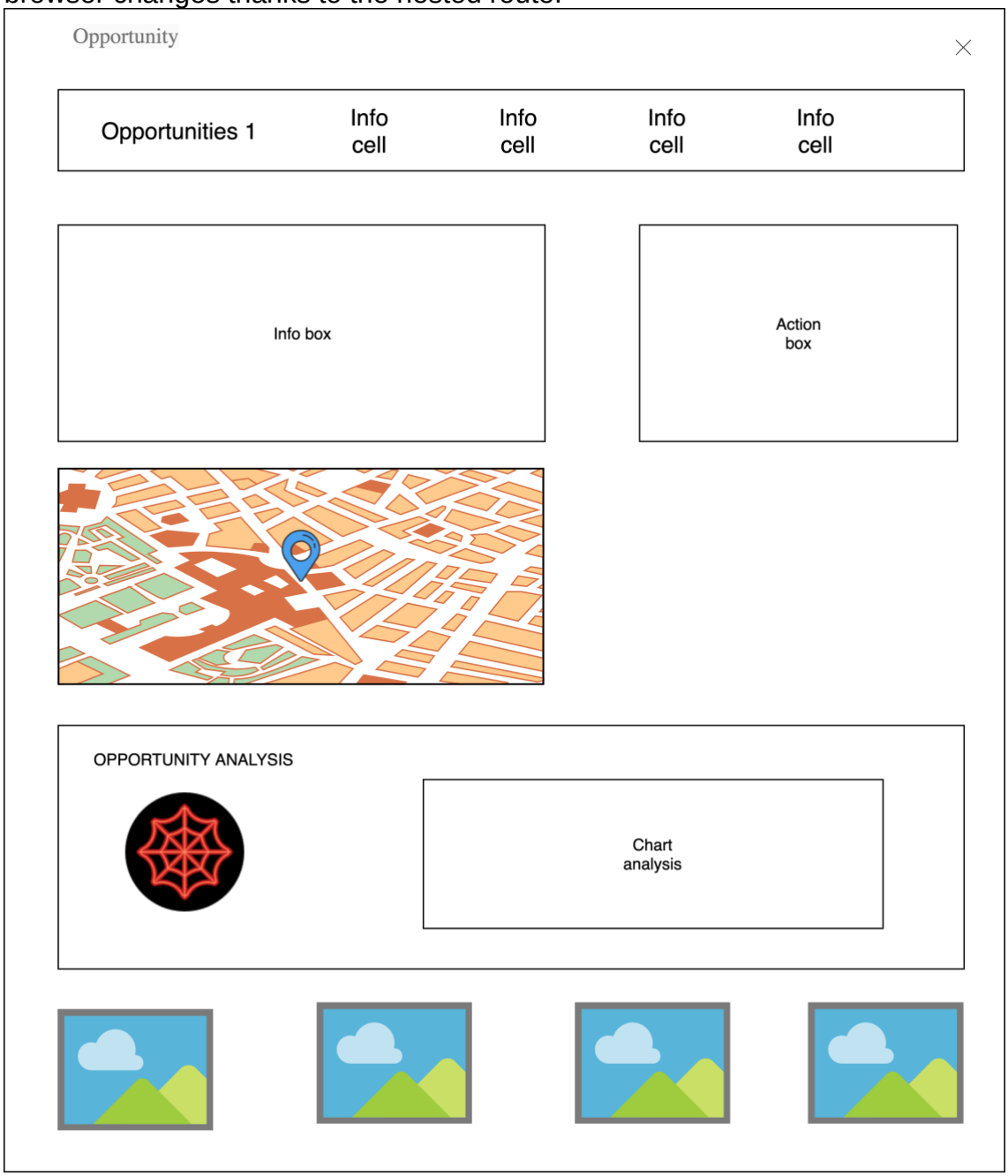

Figure 15 Detail view blueprint

As illustrated in Figure 15, the Detail view composes of the header which is the card in the Dashboard view, the info box and action box, a map, a chart with analysis, and images of the apartment.

4.6.1 Fetching data

The Detail view used useState Hook to store the opportunity data, and useEffect Hook to perform fetching.

```
const [opportunity, setOpportunity] = useState(_initialOpp)
const [loaded, setLoaded] = useState(false)
const [error, setError] = useState(false)
const { updateSingleOpp } = useOpportunities()
const [adImages, setAdImages] = useState([])

useEffect(() => {
  let didCancel = false
  const fetchOpportunity = async () => {
    setLoaded(false)
    setError(false)
    try {
      const opp = await getOpportunity(match.params.id)
      if (!opp[0]) {
        throw new Error('Opportunity not found')
      }
      if (!didCancel) {
        setOpportunity(opp[0])
        setError(false)
        setLoaded(true)
        getAdImages(opp[0].href).then(setAdImages)
      }
    } catch (e) {
      if (!didCancel) {
        setError(e.message)
      }
    }
  }

  fetchOpportunity()

  return () => {
    didCancel = true
  }
}, [match.params.id])
```

Listing 17. Fetching opportunity data

Listing 17 describes how the useState and useEffect were applied to the Detail view. Besides opportunity state, loaded, error, and adImages states were also stored by useState Hook to reflect the state of the component to the UI. Moreover, updateSingleOpp function from the Opportunities context was declared to be employed by actions in action box.
In the useEffect Hook, firstly the getOpportunity function from the opportunityResources operates to fetch the current opportunity data. Then the opportunity, loaded, and error states are set according to the fetching status. If the opportunity fetch succeeds, the getAdImages function from the insightResources fetches the opportunity images subsequently.

4.6.2 Info box and action box

The info box simply displays additional information that are not available in Dashboard view such as floor, balcony, lot ownership, maintenance fee, built year, and renovations. The renovations are divided into two categories: past renovations and future renovations.
The action box handles notes, contacted flag, and status actions including discard, assign, handle, snooze, and reset to unassigned.

```
1.  const updateCurrentOpportunity = ({
2.    status = opportunity.status,
3.    notes = opportunity.notes,
4.    discardedReason = opportunity.discarded_reason,
5.    contacted = opportunity.contacted,
6.  }) => {
7.    const assigned_to = status === STATUS_UNASSIGNED ? '' : user.email
8.    const _reason = status === STATUS_DISCARDED ? discardedReason : ''
9.
10.   updateOpportunity(match.params.id, {
11.     status,
12.     assigned_to,
13.     notes,
14.     discarded_reason: _reason,
15.     contacted,
16.   })
17.     .then(response => {
18.       const updatedOpp = response[0]
19.       updateSingleOpp(match.params.id, { ...updatedOpp })
20.       setOpportunity(updatedOpp)
21.     })
22.     .catch(e => {
23.       console.error('Error update current opportunity ', e.message)
24.     })
25. }
```

Listing 18. Handle actions with updateCurrentOpportunity function

The Detail view passes the updateCurrentOpportunity function, which is defined in Listing 18, to its child component for handling actions. The function takes an option argument, which contains one or all attributes such as status, notes, discardedReason, contacted. Based on the status, the function also sets the assigned_to variable accordingly. If the status is unassigned, the assigned_to is reset to an empty string; otherwise, it is set to the email of the current user. After getting all the attributes, the updateOpportunity from the opportunityResources is used to update the database. Then updateSingleOpp updates the current opportunity to the list of opportunities in the Opportunities context, so that the Dashboard view can be reflected correctly.

4.6.3 Map component

The Map component utilized react-leaflet library from OpenStreetMap to handle rendering the map. The coordinates of an opportunity are fetched using the same technique when fetching an opportunity.

```
1.  const [coords, setCoords] = useState({ latitude: null, longitude: null })
2.  const [mapLoaded, setMapLoaded] = useState(false)
3.  const [mapError, setMapError] = useState(null)
4.
5.  useEffect(() => {
6.    const fetchCoords = async () => {
7.      setMapLoaded(false)
8.      setMapError(null)
9.      try {
10.       const response = await getCoordinates(address)
11.       setCoords({
12.         latitude: response.latitude,
13.         longitude: response.longitude,
14.       })
15.       setMapError(null)
16.       setMapLoaded(true)
17.     } catch (e) {
18.       setMapError(e.message)
19.       setMapLoaded(true)
20.     }
21.   }
22.
23.   fetchCoords()
24. }, [address])
```

Listing 19. Fetch opportunity coordinates

As shown in Listing 19, the useState and useEffect hooks are applied to handle state and fetching of the coordinates. The getCoordinates from miscResources fetches the coordinates with a given address. The states are set accordingly to the fetch status. The coordinates are then passed to the Map component, which renders the map, the marker, and the address information subsequently.

4.6.4 Analysis component

The Analysis component consists of a radar chart named OMG, and a static description for attributes in the chart. The OMG chart displays two lines of data: the scores and the thresholds. The score is retrieved from the Details view and the threshold is always five (5) for every attribute. The project employs react-apexcharts library to render the radar chart. Due to confidentiality of the data, the attributes and the OMG chart are not pictured in the thesis.

4.7 Unit testing

Unit testing is an important part in software development because it helps developers to prevent errors, maintain the code base easier, and be able to refactor with confidence. Nevertheless, developers often neglect writing test, which leads to unstable code base and unpredictable errors.

In Jorge, all the utility functions had unit tests. Furthermore, functions for transforming data and making calculation were also covered.

```
1.  test('Construct URL Param', () => {
2.    const testParam = {
3.      a: 'foo',
4.      b: 2,
5.      c: null,
6.    }
7.    expect(constructURLParam(testParam)).toEqual('a=foo&b=2&c=null')
8.
9.    const geocodingParam = {
10.     country: 'finland',
11.     address: encodeURIComponent('Erik Bassen tie 12'),
12.   }
13.   expect(construcURLParam(geocodingParam)).toEqual(
14.     'country=finland&address=Erik%20Bassen%20tie%2012'
15.   )
16.
17.   const oppsParam = {
18.     days_before: 90,
19.   }
20.   expect(construcURLParam(oppsParam)).toEqual('days_before=90')
21.
22.   const insightParam = {
23.     url: 'https://asunnot.oikotie.fi/myytavat-asunnot/espoo/14959971',
24.   }
25.   expect(construcURLParam(insightParam)).toEqual(
26.     'url=https://asunnot.oikotie.fi/myytavat-asunnot/espoo/14959971'
27.   )
28. })
```

Listing 20. Unit test for constructURLParam utility function

Listing 20 describes test cases for the constructURLParam function which forms a query string from an object. The function is used to form the API URL for many resources; thus, test cases are prepared accordingly. Take geocoding whose API needs two parameters: country and address for example, the returned value of the construcURLParam function has to be exact a query string which contains country and address with their corresponding values. If the returned value mismatches the expected output, the test will fail, and latter assertions stop executing.

5 Result

The outcome of the project is the Jorge application which is used by the real estate specialist team at Kodit. Figure 16 and Figure 17 show the Dashboard view and the Detail view respectively.

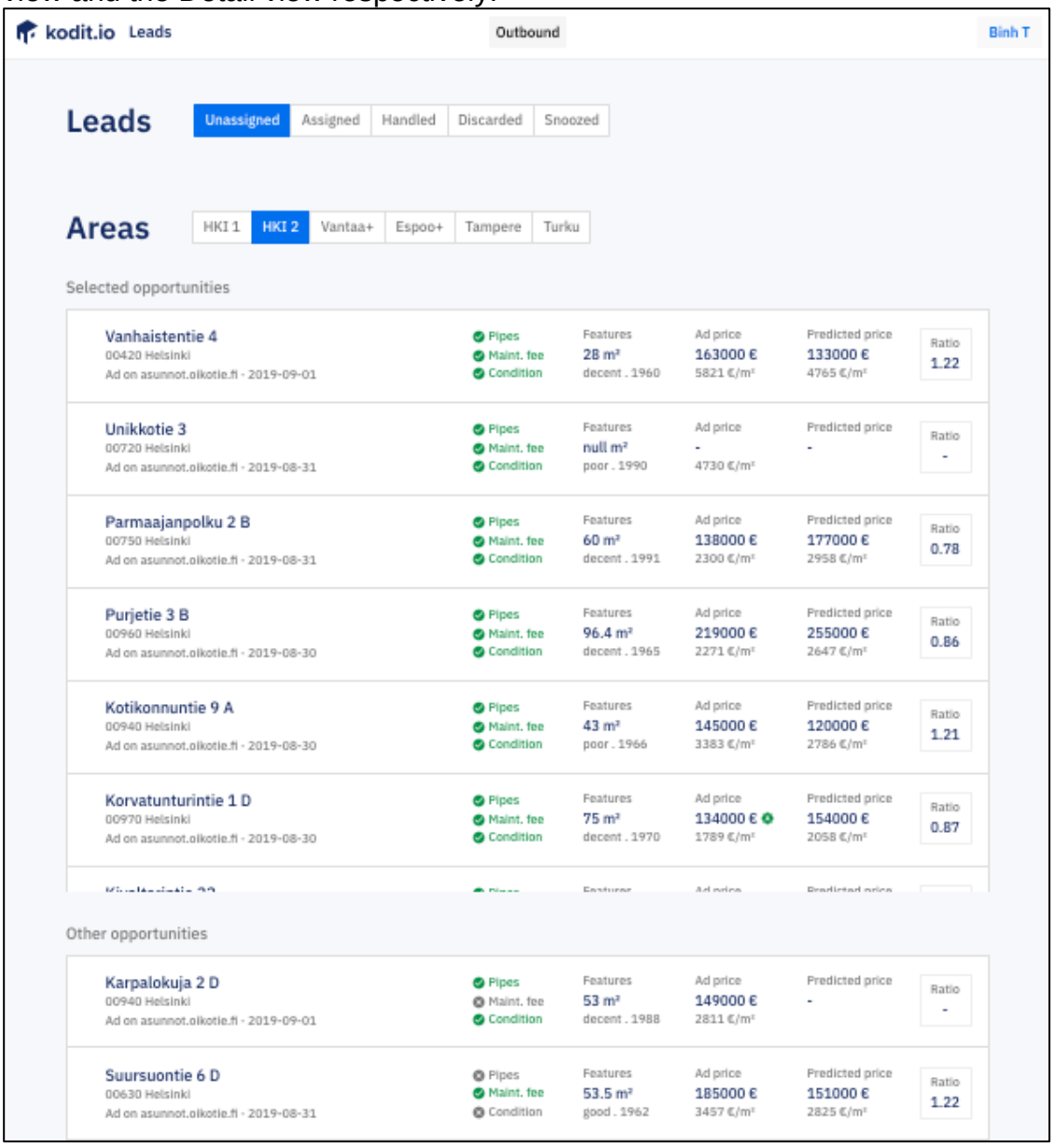

Figure 16 Dashboard view

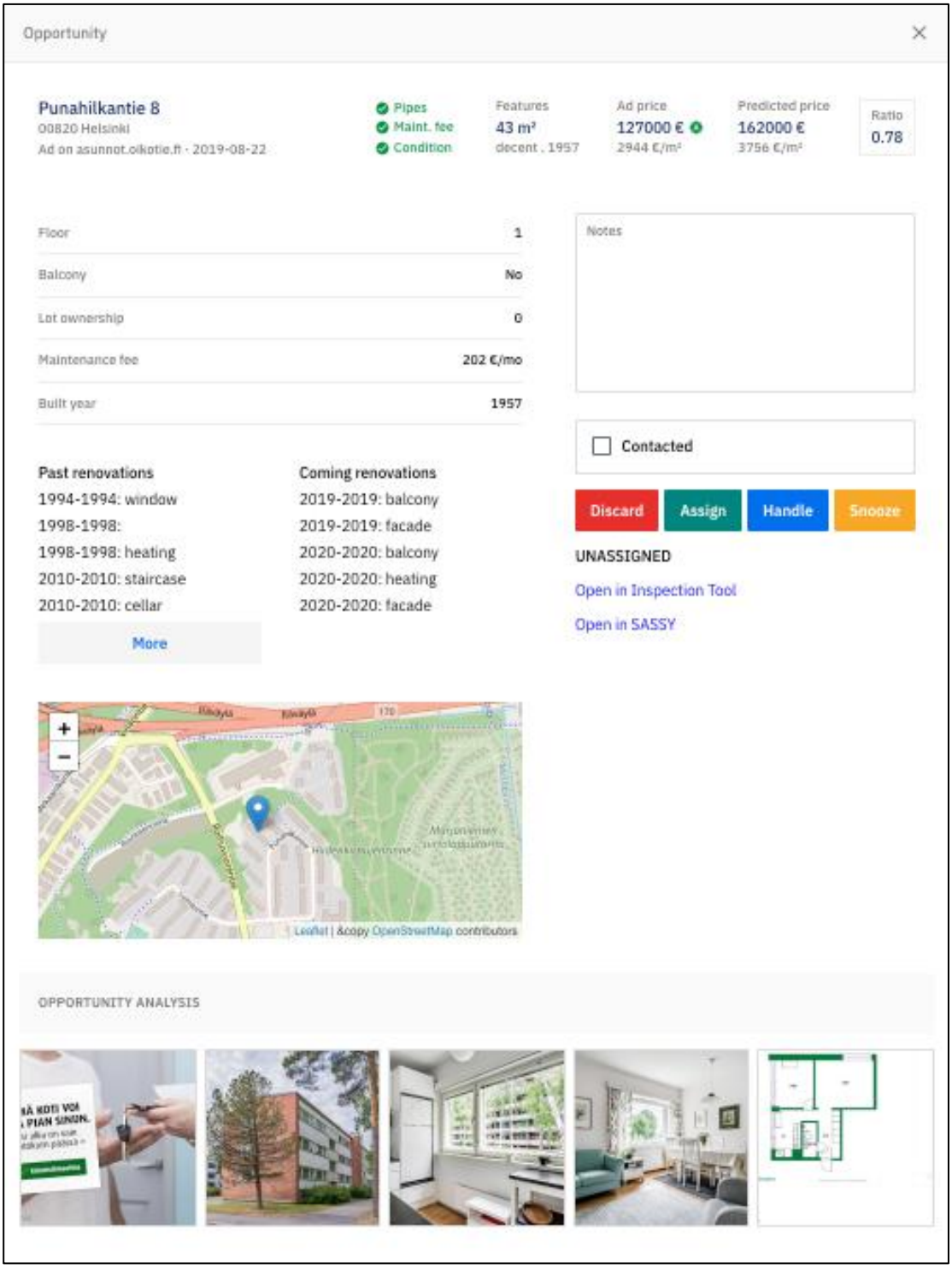

Figure 17 Detail view

The application is very simple to use. A specialist is greeted with the list of opportunities in the Dashboard when she enters the application. The area filter is preselected based on the last visit. The specialist can move around the status tab to check her workload. To enter the Detail view, which is illustrated in Figure 17, the user needs to click on the desire opportunity. The Detail view provides all relevant information of the opportunity such as size, built year, renovations, images, location on the map, and so on. The user can change the status and assign the opportunity to herself by clicking the respective action buttons.

Moreover, the specialist can take a note and mark the opportunity as contacted if she made a call to the seller.

Jorge Dashboard was given good feedback from specialists during the first month in use. It is said that Jorge is a really good and easy way to analyze the opportunities and get inspections. Jorge is a huge upgrade from the Opportunity Bot email. In addition, Jorge improves the productivity of the specialist team dramatically with the number of acquisitions increased by 30% in the first month since it was launched.

This is the early version of the application. More functionalities and improvements will be added constantly as the application is used more by specialists. For example, requested features are search functionality, different sorting methods, more related analysis. Furthermore, when Kodit enters a new market, Jorge needs updating to be compatible with the new data structures and requirements.

6 Conclusion

The goal of this final year project was to select the optimal toolset for Kodit between the two leading front-end technologies Angular and React, and to implement the Jorge Dashboard application. The study suggests that React was chosen because of its fast-growing community, easier learning curve compared to Angular, talent attraction, and suitability for quick development and release cycles.

Although adopting React was a right decision, there is still room for learning and improvement when developing internal tools at Kodit. At the beginning of the Jorge project, Flow was used as a type checker. However, over the development course of the project, Flow has shown not to be the best tool to do the job. The Kodit tech team eventually decided to move on to use TypeScript because TypeScript has a better syntax and integration with source code editors such as VS Code. Furthermore, TypeScript has a much larger community, so the project is constantly updated and improved.

Moreover, continuous integration, delivery, and deployment proved to be vital for fast development and release. The study also encourages further reading on Lean Software Development methodology. The methodology is a key factor for Kodit development process.

Last but not least, Jorge was a success project for Kodit. The tool proved to be useful for the specialists' team as well as for the company as a whole. After Jorge, Kodit tech team has the experience using React for developing and scaling current and future products. There is still a lot to learn, but the future is looking bright for React at Kodit.

7 References

Facebook, 2019. *Create React App - Set up a modern web app by running one command..* [Online]
Available at: https://facebook.github.io/create-react-app/
[Accessed 12 6 2019].
Virmani, M., 2015. *Understanding DevOps & bridging the gap from continuous integration to continuous delivery.* Pontevedra, Spain, IEEE.
Shahin, M., Babar, M. A. & Zhu, L., 2017. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access ,* 5(IEEE), pp. 3909 - 3943.
ESLint, 2019. *Getting Started with ESLint.* [Online]
Available at: https://eslint.org/docs/user-guide/getting-started
[Accessed 24 06 2019].
Prettier, 2019. *What is Prettier?.* [Online]
Available at: https://prettier.io/docs/en/index.html
[Accessed 24 06 2019].
Poppendieck, T. & Poppendieck, M., 2003. *Lean Software Development: An Agile Toolkit.* 1st ed. s.l.:Addison-Wesley Professional.
Facebook, 2019. *React.* [Online]
Available at: https://reactjs.org/
[Accessed 22 July 2019].
Facebook, 2014. *JSX Specification.* [Online]
Available at: https://facebook.github.io/jsx/
[Accessed 27 07 2019].
Porcello, E. & Banks, A., 2017. *Learning React.* s.l.: O'Reilly Media, Inc..
Mozilla, 2019. *Introduction to the DOM.* [Online]
Available at: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction
[Accessed 27 07 2019].
Abramov, D., 2018. *Making Sense of React Hooks.* [Online]
Available at: https://medium.com/@dan_abramov/making-sense-of-react-hooks-fdbde8803889
[Accessed 06 08 2019].
Kasireddy, P., 2016. *Why use static types in JavaScript.* [Online]
Available at: https://www.freecodecamp.org/news/why-use-static-types-in-javascript-part-1-8382da1e0adb/
[Accessed 02 09 2019].
Bodrov-Krukowski, I., 2018. *Angular Introduction: What It Is, and Why You Should Use It.* [Online]
Available at: https://www.sitepoint.com/angular-introduction/
[Accessed 15 10 2019].
Google, 2019. *Introduction to components.* [Online]
Available at: https://angular.io/guide/architecture-components
[Accessed 21 October 2019].
Google, 2019. *Dependency Injection in Angular.* [Online]
Available at: https://angular.io/guide/dependency-injection
[Accessed 31 10 2019].
Google Trends, 2019. *React, Angluar, Explore - Google Trends.* [Online]
Available at: https://trends.google.com/trends/explore?date=2017-01-

01%202019-11-09&q=%2Fm%2F012l1vxv,%2Fg%2F11c6w0ddw9
[Accessed 12 11 2019].
npm trends, 2019. *react vs angular.* [Online]
Available at: https://www.npmtrends.com/react-vs-angular
[Accessed 14 11 2019].
Stack Overflow, 2019. *Stack Overflow Developer Survey 2019.* [Online]
Available at: https://insights.stackoverflow.com/survey/2019
[Accessed 14 11 2019].
Dresher, T., Zuker, A. & Friedman, S., 2018. *Hands-On Full-Stack Web Development with ASP.NET Core.* BIRMINGHAM - MUMBAI: Packt Publishing.
Deitel, H., Deitel, A. & Deitel, P., 2011. *publisher logoInternet & World Wide Web: How to Program.* Fifth Edition ed. s.l.:Prentice Hall.
Wellens, P., 2015. *Practical Web Development.* s.l.:Packt Publishing.
Pelletier, J., 2015. *The Front-End Spectrum.* [Online]
Available at: https://medium.com/@withinsight1/the-front-end-spectrum-c0f30998c9f0
[Accessed 8 12 2019].
Internet Live Stats, 2019. *Total number of Websites.* [Online]
Available at: https://www.internetlivestats.com/total-number-of-websites/
[Accessed 11 12 2019].
Statista, 2019. *Global digital population as of October 2019.* [Online]
Available at: https://www.statista.com/statistics/617136/digital-population-worldwide/
[Accessed 27 01 2020].