jamk.fi

Development and design of a drawing reference web application

Aarni Ylhäinen

Bachelor's thesis
May 2021
Information and Communication Technologies
Bachelor's Degree Programme in Information and Communications Technology

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Description

Author(s)	Type of publication	Date
Ylhäinen, Aarni	Bachelor's thesis	May 2021
		Language of publication: English
	Number of pages: 31	Permission for web publication: x

Title of publication
Development and design of a drawing reference web application

Degree programme
Information and Communications Technology

Supervisor(s)
Manninen Pasi, Niemi Kari

Assigned by

Abstract

The objective was to develop a web application that would solve the following challenges: displaying Google Street View -locations intuitively and make finding specific locations efficient, as well as designing an application UI that caters to simple needs but is expandable. The application was a personal project of the author and would've been used by the author himself. Google Street View had proved to be a great source of art reference material for the author, who, as a result, had been accumulating hundreds of Google Street View -links.

The basic idea of the application was to display the Street View -locations on a map. Categories that showed the contents of the imagery would define the colour of the marker placed on the map, which would help the user distinguish between different locations that were geographically close to each other.

The web application utilized a ReactJS frontend, a Firebase Realtime Database working through a REST API, while using Figma and Inkscape to complete a prototype before developing the application itself.

The result was a working application complete with authentication, categories, a complete UI, albeit with some features that couldn't be completed on time. The map solution ended up working very well when compared to keeping the links in a list. Placing the links on a map proved to be a good way to display the Street View -links, since the author could always remember where the views were from.

Keywords/tags (subjects)
ReactJS, Firebase, REST API, User Interface, User Experience SPA

Miscellaneous (Confidential information)

jamk.fi

Description

Tekijä(t)	Julkaisun laji	Päivämäärä
Ylhäinen, Aarni	Opinnäytetyö, AMK	Toukokuu 2021
		Julkaisun kieli: Englanti
	Sivumäärä 31	Verkkojulkaisulupa myönnetty: x

Työn nimi
Referenssikuvien tallennustyökalun suunnittelu ja kehittäminen

Tutkinto-ohjelma
Tieto- ja viestintätekniikka

Työn ohjaaja(t)
Manninen Pasi, Niemi Kari

Toimeksiantaja(t)

Tiivistelmä

Tavoitteena oli toteuttaa verkkopohjainen sovellus, joka ratkaisisi seuraavat ongelmat: Google Street View -linkkien intuitiivinen näyttäminen ja niiden tehokas etsiminen, sekä laajennettavissa oleva käyttöliittymä. Käyttöliittymän tuli kuitenkin olla pelkistetty ja yksinkertainen. Sovellus oli henkilökohtainen projekti ja tuli kirjoittajan omaan käyttöön. Google Street View on referenssikuvien etsimiseen hyödyllinen työkalu, mutta sivuston oma kirjanmerkkijärjestelmä ei sovellu kuvamateriaalin tallentamiseen.

Sovelluksen perusidea oli näyttää linkit kartalla, sillä koordinaatit sai helposti suoraan Street View -osoitteesta. Linkin sisältöä kuvaavat kategoriat auttoivat eri linkkien toisistaan erottamisessa, erityisesti silloin kun ne olivat hyvin lähellä toisiaan.

Web-sovellus käytti ReactJS-kirjastolla toimivaa frontend-koodia sekä Firebasen Realtime Databasea REST APIa hyödyntäen. Figmaa ja Inkscapea käytettiin käyttöliittymäprototyypin sekä kuvakkeiden luomiseen.

Tuloksena oli toimiva sovellus autentikaatiolla ja kategorioilla, riittävän hyvin suunniteltu käyttöliittymä, vaikka jotkin suunnitelluista ominaisuuksista jäivät tekemättä ajanpuutteen takia. Sovellus päätyi toimimaan hyvin, erityisesti verrattuna Google Mapsin omaan kirjanmerkkityökaluun. Linkkien näyttäminen kartalla oli toimiva ratkaisu, sillä usein Street View -näkymä on helppo muistaa maantieteellisen sijainnin perusteella.

Avainsanat (asiasanat)
ReactJS, Firebase, REST API, Käyttöliittymäsuunnittelu, Käytettävyys, SPA

Muut tiedot (Salassa pidettävät liitteet)

Contents

Figures

1 Introduction

1.1 Background

Finding suitable references for drawing and painting can be hard. Especially when it comes to landscapes and architecture, the things you'll find within a reasonable distance are limited. Using photographs is a possibility, but often the photographer doesn't take panoramas, they consider the lighting and composition and further edit the picture to enhance it. For this reason, while using a certain person's photos as reference, one is confined to the artistic view of the photographer.

Google Street View -imagery's purpose is far from artistic; however, it works better as reference material for this exact reason. Street View doesn't consider composition, its only goal is to show locations in an informative manner. This means the artist controls the composition by moving the camera location and direction, as well as zooming in and out. Locations are plentiful, as many countries have mapped most of their cities, and often rural areas too. The potential of street view has been recognized by many people, one of which google has published an article about, namely Bill Guffrey. He had the following to say about his use of Google Maps:

> *"One of the whole points was to be able to travel virtually and paint places that I had not been," he said, adding that he tried to capture the grit of real life and not just the "pretty, pretty scenes" (Heussner, Ki Mae, 2009).*

Figure 1 is an example of using Google Street view as drawing practice, made by the author.

Figure 1. Using Google Street View to practice drawing.

In Figure 2 you can see the setup the author uses for practice, with the reference image on a monitor above the drawing tablet.

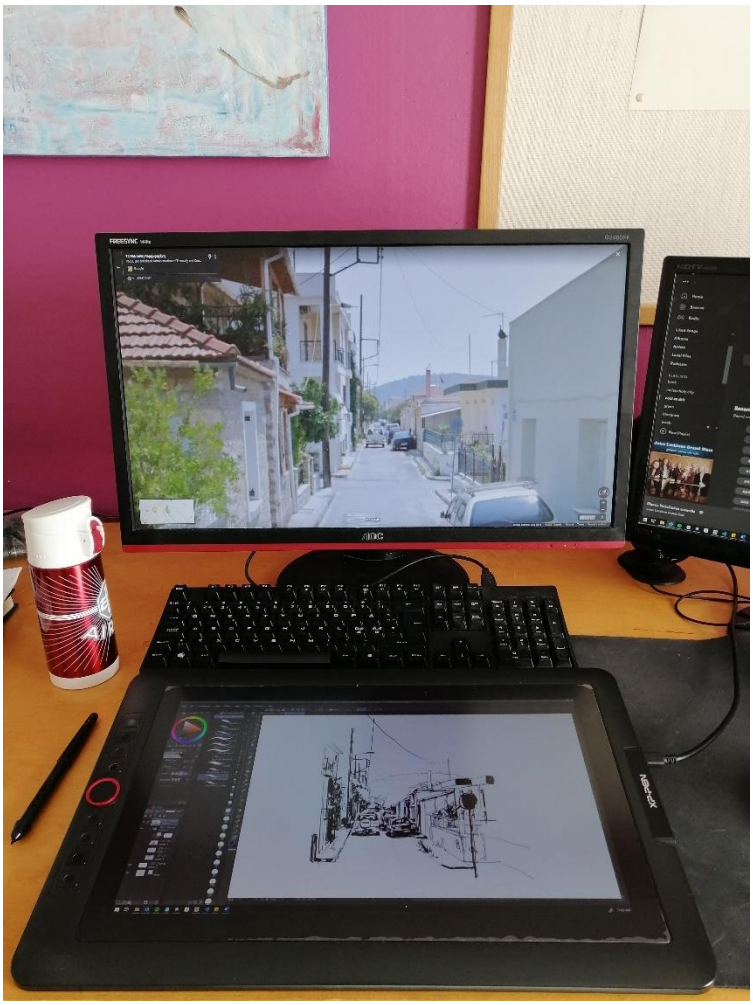

Figure 2. Drawing tablet setup with Google Street View set up as a reference.

The imagery on google maps is skewed by lens distortion, which means the artist must use their imagination and 3-dimensional understanding to adapt the 360 -degree image onto paper. In practice, this means your understanding of 3d spaces improves as you work. An example of counteracting distorted imagery can be seen in Figure 3, on the Toyota Hijet on the left. On Street View it almost seems like a big truck, while in reality it's a remarkably small car.

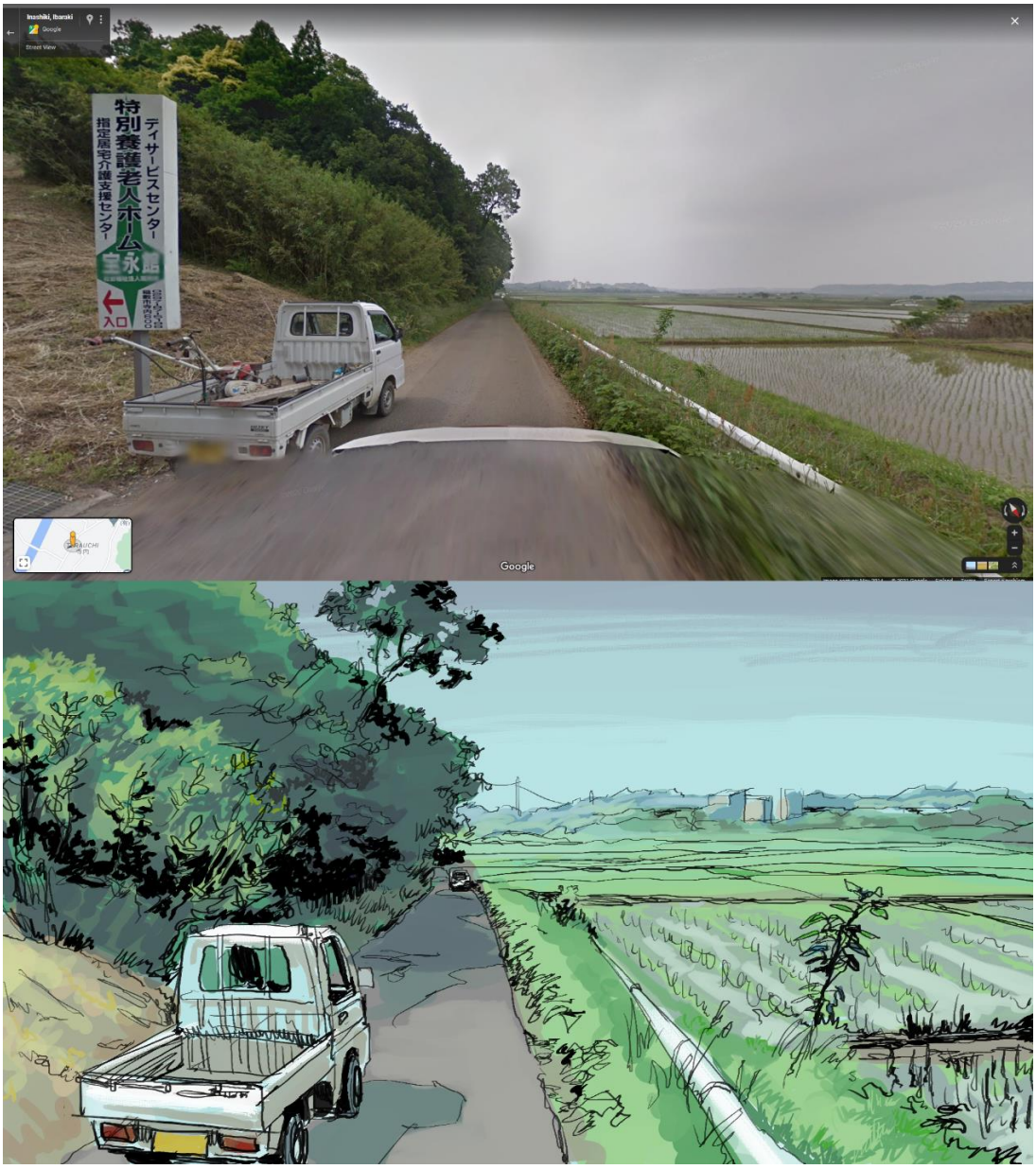

Figure 3. Demonstration of counteracting the lens distortion.

On top of all this you have endless amounts of landscapes, cityscapes, buildings, and vehicles to draw, all from the comfort of your home. Figure 4 is an example of how

Street View can be used to practice: drawing the location from a slightly different view, and freely choosing what to include in the image.

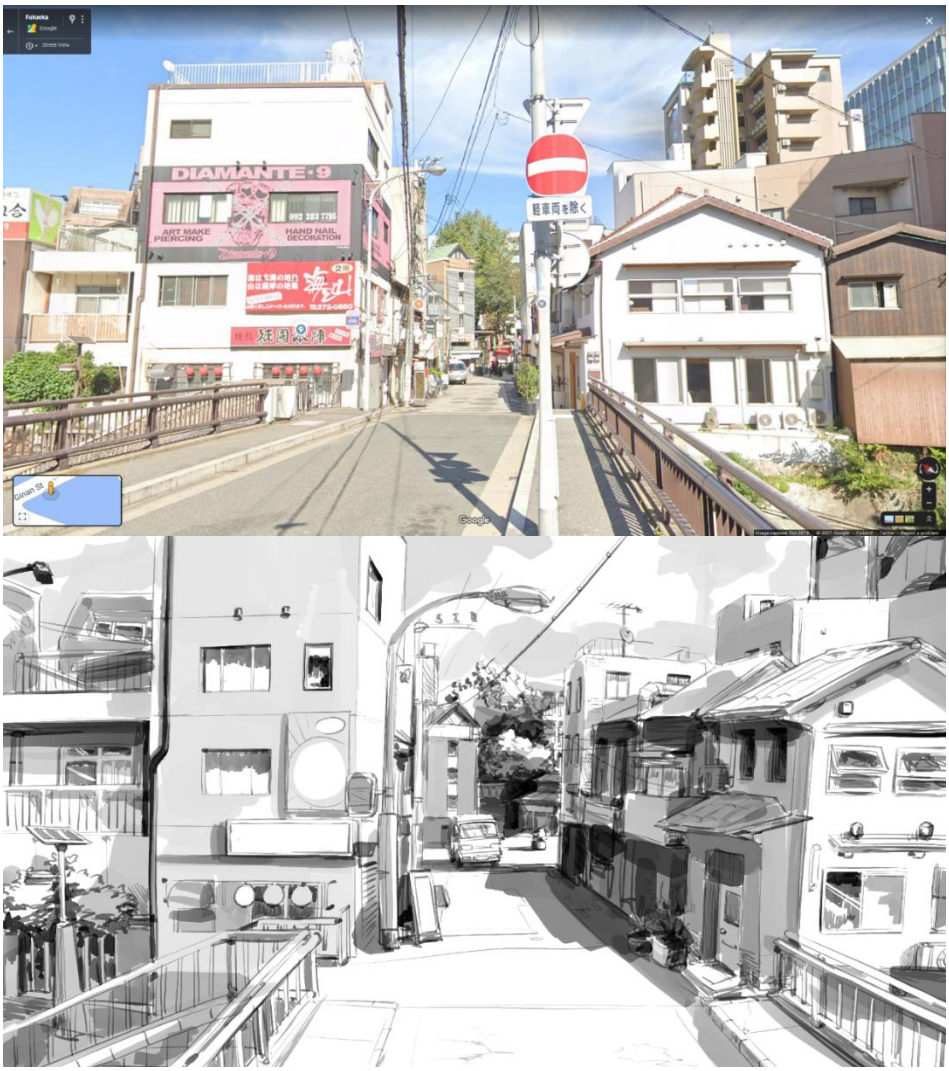

Figure 4. Changing the viewpoint

Different forms of practice can help understanding 3d environments better, as opposed to just drawing what you see in the reference image. Figure 5 was made

based on the street view image, but only from memory.

Figure 5. Drawing a scene from memory

1.2 The Issue

The author started accumulating a large amount of Google Maps -locations and needed a way to store them. A list containing the links and short descriptions of the locations sufficed for a time, but the number of links grew over a hundred. It was clear that it had become an issue, as finding a specific link out of several hundred is not fast nor fun when they’re presented as a list with simple descriptions of the locations.

1.3 The Goal

The goal of was to create a good-looking and easy to use web application that stores and shows Street View imagery using a RESTful API. A user could make an account using their email, or alternatively use their existing Google account. After logging in the users could store Street View links that are stored in a Firebase database and shown on a map. Each user would have their personal data. When adding links, the user could select from a set of different categories. Categories would define the colour of the location's pin, such as landscape, cityscape, or a car. A layer system that allows hiding certain categories might prove useful as well.

There will be places with plenty of pins condensed within a small area, and they may be hard to distinguish. Different categories having different pin colours would help with that. Adding notes to the links would help with links that are close to each other, and in the same category. These notes would show up in a menu or popup of some sort. The popup should also contain a link to the view itself, which is the most important part, as well as a button for deleting the location.

As for comparison with other tools, the result had to work better and faster than Google Maps' own bookmarking function. Said function is hidden away in Google Maps' menu and is quite limited if you want to save a specific composition. While it does show the locations on the map, all the functions are hidden in menus, and the saved location won't have the zoom and direction you pointed the camera towards.

1.4 Research methods

The aim was to answer the following questions: "how to display Google Street View - locations intuitively and make finding certain locations efficient", and "how to design an application UI that caters to rather simple needs but has potential for additional features".

Displaying the links intuitively was especially important, as storing text is easy, but it's harder to make the information easy to find. This was the main problem that led to creating the application, as opposed to managing them in something more akin to Excel sheets. Creating a UI for the initial use was the goal as well, but it also had to be expandable.

2 User Interface and Design

2.1 UI vs UX Design

The roles of UI and UX designers, while related, are very different. The role of a User Experience designer is to analyze the needs of the users and trying to ensure the application meets those needs (Babich, Nick, 2019). UI designers on the other hand concentrate on the concrete visuals, rather than the experience. Their purpose is to use visual design skills in order to turn the UX designers' prototype into a finished product. They're also responsible for making the design responsive. (Babich, Nick, 2019)

According to Babich (2019), UX and UI design are sometimes advertised as a single role, or the roles may be mixed up. They are indeed related in subject, but the practical tasks and required skills are quite different. Both are important however: a great UI designer without a great UX designer may design the interface for a person who doesn't exist, with needs and preferences based on fiction. A great UX designer without the support of UI designers may create great prototypes but the technical part of the presentation may fall short.

2.2 The Importance

A lot of success stories credit their rise largely to User Experience. Companies such as Facebook, Amazon and Google didn't create a completely new concept, they instead

developed software that grew their userbase with good design (Kucheriavy, Andrew 2015).

2.3 Nielsen's heuristics

Heuristic evaluation is a common concept in UI design. It's a process where simple "rules of thumb" are used to examine and measure usability. (Interaction Design Foundation n.d) One example of a heuristic ruleset is Jakob Nielsen's "10 Usability Heuristics for User Interface Design" (Nielsen, Jakob 1994).

- Jakob Nielsen's (1994) first rule is "visibility of system status". It's about keeping the user informed of the state of the application, using various UI elements.
- The second rule is about the concepts being understandable: *"The design should speak the users' language. Use words, phrases, and concepts familiar to the user, rather than internal jargon."*
- The third rule involves giving the user a sense of control. Having the ability to easily undo actions gives users a sense of safety for example.
- According to the fourth rule and somewhat related to the second one, the design should use familiar elements. Having industry standard icons, such as the burger icon for side menus on mobile, is important in making the application easy to use.
- Before error handling and -messages, the developer should concentrate on error prevention.
- Rather than recalling the documentation, UI elements should be recognized without prior knowledge of the UI.
- The UI should have shortcuts that speed up the application use of an experienced user, which are hidden from a novice user. These can be customizations, keyboard shortcuts etc.
- The visual design should focus on the essentials.
- Error messages should be easily identifiable as error messages, tell the users what went wrong without technical terms, and offer a solution to the problem.
- While the application should need as little explanation as possible, it is sometimes unavoidable. According to the tenth and last rule, documentation should be available.
  (Nielsen, Jakob 1994)

2.4 Responsiveness

Responsiveness is very important in creating a web application which is supposed to be a simple tool. Responses to inputs must be as fast as possible, and the application's UI must make it clear that the user's inputs have been received. Akamai (2017) estimated in a study that a company can lose a 7% of their sales on their web store if the loading times increase by 100 milliseconds. That's a huge hit for a tiny problem

and shows that a swiftly working application is important to people, even if they don't necessarily notice the difference.

3 Technologies

3.1 Prototyping and graphic design

3.1.1 Figma

Figma is a collaborative design tool. Developers can work on projects and see each other's changes in real time. It can be used for several different uses, UX- and UI and graphic design being common examples. It's similar to a lot of other design tools on the market, but what sets it apart is the collaborative nature of it. (Figma API Documentation n.d)

3.1.2 Inkscape

Inkscape is a free open-source vector graphics editor. It's possible to export web-ready vector files quickly and efficiently, similar to Adobe Illustrator. (Inkscape overview 2021) The application was announced to be in development as early as 2003, and had emphasis on community-oriented development, as well as having a small core with extension support for additional features (Harrington, Bryce 2003).

3.2 React.js

3.2.1 Basic information

React is an open-source JavaScript library, used for creating the frontend side of single-page and mobile applications (React documentation n.d). Work on React was started in 2011, by Facebook. One of the reasons React exists is to literally react to changes in the data by only updating the components with the render method, instead of reloading the page. (Hunt, Pete 2013)

3.2.2 JSX

When writing React.js applications, you aren't working with html and JavaScript separately. Defining UI elements React uses JSX syntax; an extension of JavaScript that looks a like a combination of html elements and JavaScript. According to the React.js documentation, everything is JavaScript at its core; instead of separating markup and code React.js combines them. (React documentation n.d) The use of JSX makes the file structure very clear, and also makes it easy to implement code in the frontend.

3.2.3 Props and State

State is the way to manage the state of the application. It can store data retrieved from the backend, the application's status, or any other relevant information. (React documentation n.d)

Props and state are regular JavaScript objects. They have one key difference: state is managed by and within a component, and props get passed to the component. (React documentation n.d.) Props can also make the child component update as it gets new props, if for example the props are tied to the higher component's state.

When thinking about the component structure, developers may decide on a higher "container" component that is stateful, meaning it contains the application state. Most components within the stateful ones are stateless components that get props from their parent component, containing data to display or possibly references to functions in the container. (React.js stateless vs stateful 2019)

3.2.4 Components

The React.js UI is based on Components. They let the developer slice the application into smaller pieces that are reusable and can either manage their own state or get it from a component higher in the order. (React Documentation n.d) Components are imported into the code the same way as other resources and can be declared the same way any other JSX element can.

3.3 Redux

3.3.1 Basics

Managing application state over different parts and pages of the application can be challenging. Redux makes this easier, the idea behind it is having a central store for the application state, which you can access and modify in any react -component you connect to it (Why Use React Redux n.d).

Redux isn't part of React, but they are often used together using react-redux. Managing React state in a growing application can become a complicated process, while Redux store can be modified with the same actions from anywhere within the application. Figure 6. shows how this works on a conceptual level.

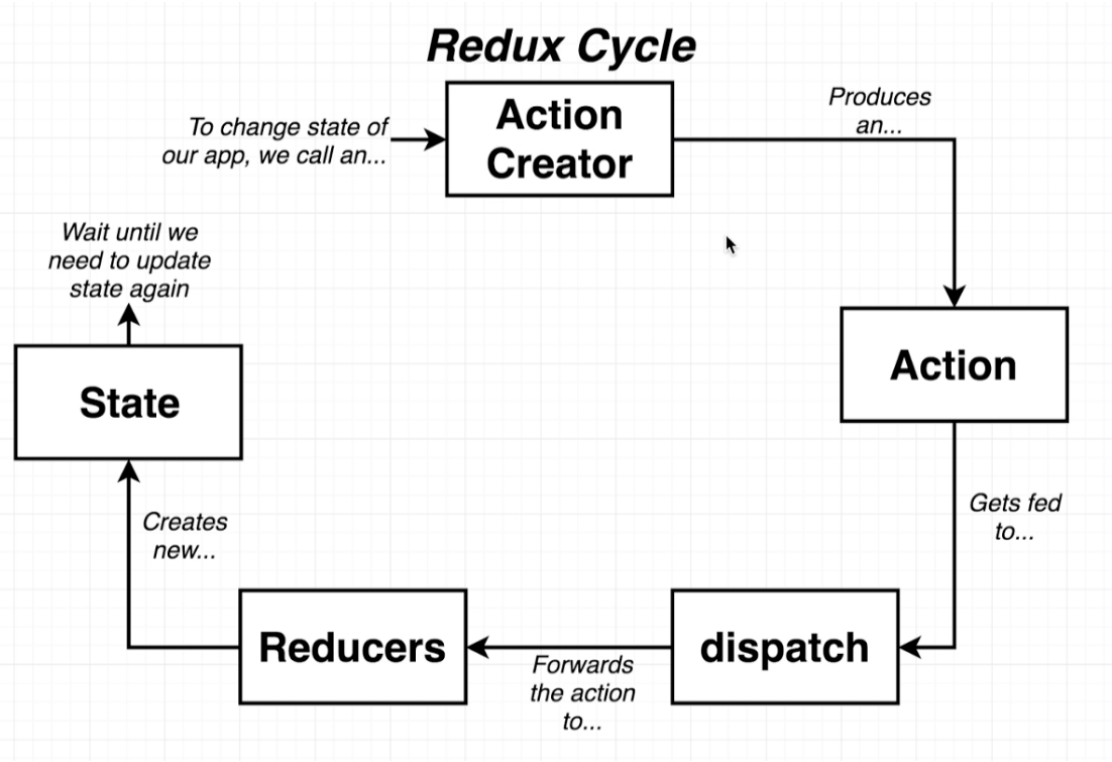

Figure 6. Redux cycle (Hamedani, Mosh 2020)

3.3.2 Redux Thunk

You can only do synchronous updates in a plain Redux store. Thunks let you add asynchronous logic, which is helpful in running code after making http -requests. Simply put, “A thunk is a function that wraps an expression to delay its evaluation”. (Redux-thunk Github n.d.) Redux Thunk is one of the simplest thunks available. It adds literally only fourteen lines of code, that allow the developer to delay functions within Redux.

3.4 Axios

A http client handles communicating with the server. Axios is a common choice for web applications that use Node.js. It’s simple to setup by just installing the node -package and making a small config file, and automatically transforms response data into JSON for ease of use (Axios Github n.d.).

3.5 Firebase

3.5.1 General Information

Firebase is a platform for creating databases for the web and mobile, developed and maintained by Google. Its features include built-in hosting, authentication, different database types and cloud functions. (Firebase documentation n.d)

3.5.2 Realtime Database

The Realtime Database is one of the database options on Firebase. It allows real-time synchronization between the client and user, and while there are some things the developer should know before setting it up, it’s possible to set up a database quickly and start building the application itself immediately. The Realtime Database uses JSON, which means it’s easy to parse through the data. It can be used via a REST API, by using the Realtime Database URL as a REST endpoint. (Firebase documentation n.d)

4 Implementation

4.1 Premise

The application started as a side project, not a thesis. The author made a rough prototype of this tool before and chose to use Firebase and React.js. Firebase is easy to use, free to use for personal projects and lets a frontend developer concentrate on the frontend. React.js on the other hand allows updating the view easily, which was required to make the application work smoothly.

4.2 User Interface and design

4.2.1 Starting out

The first thing to make sure of was a solid User Interface design, before starting to build the application itself. It made designing the architecture under the hood, for example React.js components structure, significantly easier.

There was no need to make the application work on mobile devices. The intended use of it is on a personal computer or laptop, with a monitor big enough to show the reference image properly. It would, however, have to scale well regardless for different monitor sizes.

The author's friend had the idea of placing the links as markers on a map. This solved a major part of the problem, as long as there would be a way to have different colors of links to identify the link with something more than just its location. The user usually searches for locations with specific kinds of views in mind and remembers the general whereabouts of the view.

The next problem was how to differentiate links that are relatively close to each other. This is solved by having a set of well-defined categories, which represent the

type and purpose of the view. The author had been using Google Maps as drawing reference for a while already, so it was easy to define the required categories.

4.2.2 Prototyping in Figma

Figma was used to make a simple wireframe of the application. Figma can also be used for creating interactive prototypes, but for the scale of the application it wasn't necessary. The author chose the software because he was interested in learning it, even though collaborative features weren't needed. Figma proved to be a solid UX design tool. It also supported importing Inkscape vector images, so there was no need to create placeholders for the prototype. As seen in Figure 7, it was easy to create the whole UI in the application's web version.

The design required walking the line between clear visuals and overdesigning a simple application. It had to be decent to look at, but not contain anything extra. This is the difference between illustration/interfaces and creating personal art projects: the design must be concise and not get in the way of the usability, while art can look flashy.

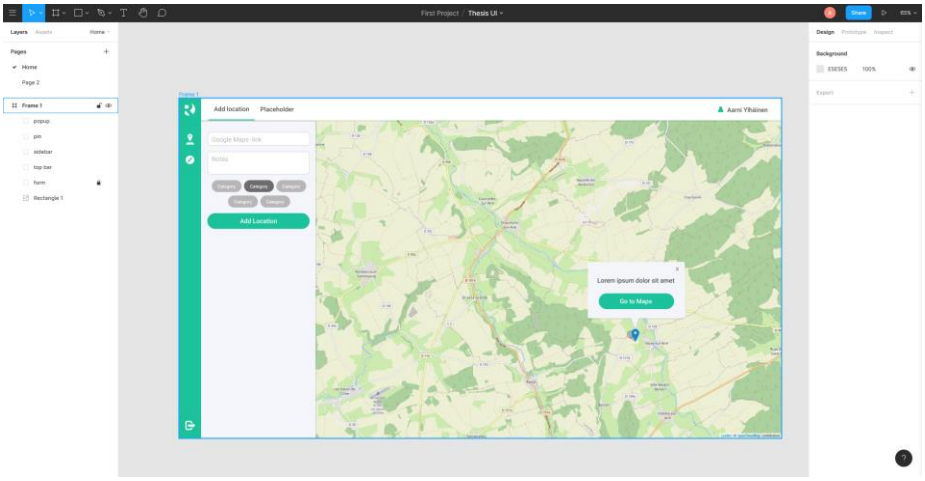

Figure 7. Figma UI Prototype

The icons were made in Inkscape and exported as web-ready vector files. They were imported straight into both Figma and the React.js UI itself.

4.2.3 From prototype to application UI

Translating the design into React.js was not a problem either. The sizes of the elements came directly from Figma for example. The React.js UI can be seen in use in Figure 8. Sizes for the elements are the same, and the icons are the exact same ones, imported as Inkscape vector images.

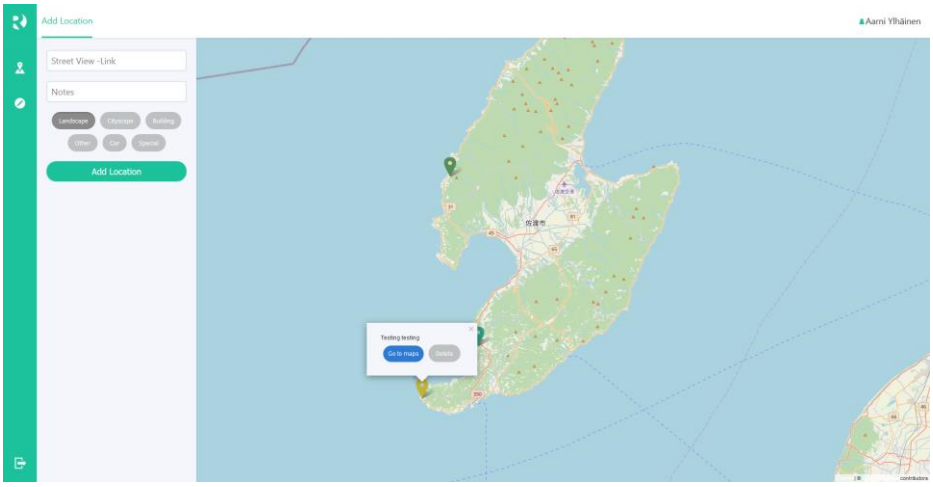

Figure 8. The UI design implemented in the web application

According to Nielsen (1994) the system status of the application must be clear. A notification was necessary to tell the user if the location was successfully added. This is handled through a component that shows up for only a moment and shows a message. (which is passed into the component by props) When the location is successfully added, it reads “location added”.

4.2.4 Choosing the colours

It was important to keep the UI clean and easy on the eyes but appealing at the same time. A bluish green seemed appropriate: According to Munro (2019), green feels natural, safe, and fresh to people. A significant portion of the images saved on the application will be landscapes with a focus on nature or countryside, so green fits the

bill. It also forms a decent-looking analogous palette with the green of the default OpenStreetMap land tiles, as well as the blue of the oceans, while still clearly separating the tones of the UI and the map tiles. Shifting the green towards blue makes a calm impression as well, ideal for a drawing tool.

The chosen green is used sparingly on the UI as an accent colour, somewhat according to the 60-30-10 rule. According to Munro (2019) this means that 60% of the design is in the primary color, 30% the secondary one and 10% of it uses the accent colour. The accent colour is mostly used to mark significant actions on the UI, such as submitting forms and the navigation component.

4.2.5 Icons

Vector images make webpages lighter, and they scale better than rasterized images. The menu icons, map pins and their shadows were made in Inkscape. The only conventional image in the application is the login page background.

4.3 React frontend

4.3.1 React.js Component Structure

At first there were three stateful components: one handling the authentication and showing a login and signup form, one for the map view and one for the random Street View -tool. The form with which the user adds the locations ended up becoming a stateful component as well after implementing Redux.

The author had some ideas for further features to the application: searchable list view, as well as a feature that finds view based on randomized coordinates. It was clear that the components used would have to be reused elsewhere. Containers for the menus were made in a way that they render the children from props, the elements that are inside the Component element. What the component shows would then be defined in a bigger, stateful component and updated dynamically.

Figure 9. shows the component structure, with some smaller components left out, more specifically, UI elements that only display some props.

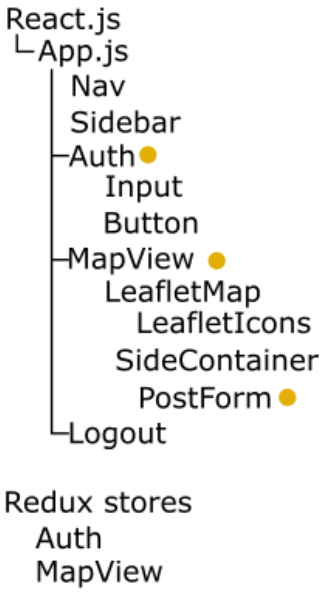

Figure 9. component structure, components that are connected to a redux store or stores are marked with a yellow circle.

A package called React Router is used to navigate the different parts of the application: there had to be a separate URL paths for the login and the application itself. This is managed by React Router: if the user isn't logged in (if the token isn't found in the Redux store) the application will automatically redirect to the login -path.

4.3.2 Leaflet and OpenStreetMap

Integrating Google Maps could've been possible, but the same coordinates work on different map tiles and systems. For simply placing pins on a map, a free alternative such as OpenStreetMap is a possibility.

Leaflet makes the use of OpenStreetMap -based maps easier. It's an open-source JavaScript -library for adding interactive maps to web applications. The version of

Leaflet used is a React.js -package called React-Leaflet. It isn't exactly the same as Leaflet, but remakes Leaflet layers into React.js components. (Leaflet introduction 2021)

4.4 Setting up Firebase

4.4.1 Requests

GET and POST -requests are required to login, get location information and modify the database. These are handled through Redux Thunk, utilizing Axios to make the requests. Figure 10 shows an example of this, in the delete request of a specific location.

```
export const removeLocation = (locationId, token, uid) => {
    return dispatch => {
        axios.delete('/places/' + uid + '/' + locationId +'.json?auth=' + token)
            .then((response) => {
                dispatch(fetchLocations(token, uid));
            })
            .catch((error) => {
                dispatch(removeLocationFail(error));
            });
    };
}
```

Figure 10. Example of a request to the database (deleting a location)

4.4.2 Firebase structure

The structure of the Firebase -database can be seen in Figure 8. All users have their own section which is named after their Firebase id. Locations are loaded only from the user's own section when loading them in the application. Apart from the link itself, the type of the location is saved, and so are the coordinates, so that the link doesn't have to be parsed from the coordinates after it's been saved. The structure can be seen in Figure 11.

Figure 11. Firebase Realtime Database's structure

The author thought the application needed to include authentication, so the default rules had to be changed, as seen in Figure 12. Unauthenticated users should stay on the login/signup page, and the rules were changed so they could not send any requests until they were authenticated. In any request to the server the authentication is checked by adding the token to the query url (Firebase documentation n.d).

```
{
  "rules": {
    "places": {
      ".read": "auth != null",
      ".write": "auth != null"
    }
  }
}
```

Figure 12. Firebase rules

Users are authenticated through a POST -request with an object containing the user's login or register -information. Firebase returns an object containing the user's information, which the application has to save. The most important thing Firebase gives is the security token. Essentially, it's added to any request made to the backend, and has a set expiration time, which is exactly one hour by default.

If the developer wants users to be able to stay logged in, they must add a function that requests a new token before the old one is about to expire. For that, again, you

make a POST -request with the user's information, this time with the refreshToken from the first time the user logged in.

4.5 Redux store

4.5.1 Splitting the store into two

The Redux store was split into 2 separate stores: one for the map view and one for the authentication. The token, refreshtoken, displayName and other information from the Firebase user account would be stored in the Authentication -store, and location-related information in the other one.

4.6 Authentication

The user is authenticated through a POST request to the correct endpoint, with the application's API key. The response to a successful POST request on Firebase is an object containing the relevant information: for instance, when using the login -endpoint the response contains the user's information; including the token, refreshToken, displayName and so on. These pieces of user information can then be stored in the Redux store and used within the other REST API requests within the application, such as getting the locations.

Searching for suitable spots on Maps takes time, and the default expiration time of an idToken on Firebase is only one hour. A separate function to refresh the idToken with a refreshToken had to be created to keep the user logged in.

4.7 Adding & handling locations

When the user changes the contents of the input fields of the post form, functions save the inputted contents (hopefully valid links and notes) to the PostForm -function's state.

When trying to add a location, a function named submitClickedHandler tries to extract the coordinates from the link, as seen in Figure 13.

```
submitClickedHandler = (event) => {
    event.preventDefault();
    const locationData = {
        link: this.state.location,
        type: this.state.category,
        note: this.state.note
    };

    try {
        let url = this.state.location.split("@");
        let at = url[1].split("z");
        let parts = at[0].split(",");
        if (parts[0] > 0 && parts[1] > 0 && parts[0] < 90 && parts[1] < 180) {
            this.props.onSubmitForm(locationData, this.props.token, this.props.userId);
        }
    } catch (error) {
        this.setState({error: error});
    }
```

Figure 13. Splitting the link for valid coordinates

The function doesn't check the validity of the rest of the link, what matters here is that it contains coordinates with valid latitude & longitude values.

Categories are defined in a separate file and a list of them is exported. At launch the application goes through them to add them to the category selection in the PostForm -component. This makes it easy for the author to add categories if more are needed.

5 Results and Conclusion

5.1 Results

The scope of the application grew slightly larger than the rather simple needs of the user. Instead of being a tool with a simple interface, it turned into a functional web application with authentication, better scalability as well as a possibility to add more features on different pages. The author has started using this tool as intended: to stash Google Maps -views in order to draw them.

The categories were diverse and clear enough, that it was easy for the user to find which image they were looking for.

As for comparison to Google Maps' bookmark function, this works a lot faster, and again, saves the whole link, including the composition. The initial goal for the thesis was met.

5.2 Conclusion

Some features would have been great additions, but there wasn't enough time to complete them. Firstly, a marker layer system, in which you would've had a separate menu with checkboxes that define which categories' links are shown.

A larger feature that the author planned was to use the Google Maps API to get random street view links. Similar websites exist already, but there would be a menu from which the location could be saved to the database, without going back to the map view.

In the future the author intends to create a browser extension for saving links straight into the database, without having to use the form within the application.

The component structure ended up being unnecessarily convoluted. Making PostForm into a component that gets all its functions through props from the higher MapView component would've been a better option; the Redux store ended up also being connected to both MapView and PostForm -components, instead of only the higher component and passing relevant information or functions as props.

References

Akamai Online Retail Performance Report: Milliseconds Are Critical. 2017. Study report published by Akamai. Accessed on 13.11.2020. Retrieved from https://www.akamai.com/uk/en/about/news/press/2017-press/akamai-releases-spring-2017-state-of-online-retail-performance-report.jsp

Axios Github. N.d. Github -page containing basic information on Axios. Accessed on 21.10.2020. Retrieved from https://github.com/axios/axios

Babich, N. 2019. What are the Similarities & Differences Between UI Design & UX Design? Article on Adobe's website. Accessed on 17.3.2021. Retrieved from https://xd.adobe.com/ideas/process/ui-design/ui-vs-ux-design-understanding-similarities-and-differences/

Firebase documentation. N.d. Documentation on Firebase's website. Accessed on 30.9.2020. Retrieved from https://firebase.google.com/docs

Hamedani, M. 2020. Redux in 2020. Article on Mosh Hamedani's website. Accessed on 16.10.2020. Retrieved from https://programmingwithmosh.com/redux/redux-in-2020/

Harrington, B. 6.11.2003. Announcing new project. Sodipodi mailing list message. Accessed on 14.5.2021. Retrieved from https://sourceforge.net/p/sodipodi/mailman/message/10064135/

Heuristic Evaluation. Article on Interaction Design Foundation's website. Accessed on 21.3.2021. Retrieved from https://www.interaction-design.org/literature/topics/heuristic-evaluation

Heussner, K.M. 2009. Google Puts the World at the Tip of Painter's Brush. Article on Abc News' website. Accessed on 13.3.2021. Retrieved from https://abcnews.go.com/Technology/story?id=8295043&page=1

Hunt, P. 2013. Why did we build React? ReactJS blog. Accessed on 14.5.2021. Retrieved from https://reactjs.org/blog/2013/06/05/why-react.html

Introduction to React Leaflet. N.d. Documentation on React Leaflet's website. Accessed on 29.11.2020. Retrieved from https://react-leaflet.js.org/docs/start-introduction

Introduction to the Figma developer API. N.d. Documentation on Figma's website. Accessed on 24.3.2021. Retrieved from https://www.figma.com/developers/api

Kucheriavy, A. 2015. Good UX is good business: how to reap its benefits. An article on forbes.com. Accessed on 23.3.2021. Retrieved from https://www.forbes.com/sites/forbestechcouncil/2015/11/19/good-ux-is-good-business-how-to-reap-its-benefits/?sh=1b378aa74e51

Munro, L. 2019. The Role of Color in Product Design: UX of Color Palettes. Article on the Adobe Xd Ideas -website. Accessed on 1.3.2021. Retrieved from https://xd.adobe.com/ideas/principles/web-design/ux-of-color-palettes/

Nielsen, Jakob. 1994. 10 Usability Heuristics for User Interface Design. Article on Nielsen Norman Group's website. Accessed on 12.4.2021. Retrieved from https://www.nngroup.com/articles/ten-usability-heuristics/

React Documentation. N.d. Documentation on ReactJS' website. Accessed on 30.9.2020. Retrieved from https://reactjs.org/docs

React Redux introduction. N.d. Article on React Redux' official website. Accessed on 3.3.2021. Retrieved from https://react-redux.js.org/introduction/why-use-react-redux

Thunks in Redux. 2017. Article on medium.com. Accessed on 5.4.2021. Retrieved from https://medium.com/fullstack-academy/thunks-in-redux-the-basics-85e538a3fe60