MASTER THESIS

From Design to Code; A Study of Automation From User Interface Design Program to *Production(?)* Code

---

Albin Frick

May 1, 2021

Master Thesis in Interaction Technology and Design, 30 ECTS
Master of Science in Interaction Technology and Design, 300 ECTS
UMEÅ UNIVERSITY

Contents

Abstract

Hello this is the abstract

1 Introduction

When building applications for companies there's a lot of moving parts. Most often the project is worked with an agile work form [1]. This means that the product is evolving, with a tight connection with the customer. Before development starts some research needs to be done to find what the customer wants, how the application should be implemented, and how it should look. When working on a big project the design and the implementation of the product are done by different competence groups, mainly designers, and developers.

The communication (and or relationship?) between these two parts are essential to creating a good user experience (UX). The communication problem between designers and developers can be compared to the common *impedance mismatch problem* in the data storage realm. Designers and developers are trying to achieve the same thing but see information differently. The front-end developer is trying to figure out how the design should be written in code. How the design elements should or shouldn't effect each other, how each of the should be positioned, etc. The designers see the relationship between the elements, how the dynamic of contrast, colors, and white space is set for the whole application.

In later years most development are done around components. Whereas components are design and/or functional elements in the application. A component could be a button, a navigation bar, a card[2], etc. This is done to save code meaning that all components can be reused through out an UI and between UI's. Many companies create big libraries of these components to be used within different projects. This works great for when the design languages. However when a consulting firms clients often does not have the same design language. This results in that the consulting firms often needs to redesign there components between projects.

1.1 The Company & the Problem

Knowit is an IT company with about 2600 employees with facilities in Sweden, Norway, Denmark, Finland, and Germany. Knowit has three main branches, Experience, Insight, and Solutions. These branches tackle different types of problems, enforce user experience with the customers brand (Experience), create system solutions to help customers digitalize (Solutions), and management-consulting (Insight). This project was done under the Experience branch to help them with an important part of initiating projects.

This branch is consulting firm for customers that need a digital tool with high availability, accessibility, and great user experience. Every project under Knowit experience is run by teams of designers and developers. Where the designers make the design for the applications, including general designing guidelines, for each project. Then the developers implement the design to a functioning application.

The problem that has araised is that the design and setup for every project is

very similar project to project. This is setting up a color palette, typography and basic components such as buttons, forms, etc. When the designer has done this the developer needs to convert this to code. This process is often very similar for every project but needs to be redone because most project doesn't use the same frameworks and/or tools.

1.1.1 Knowit Initial Requirements

The initial requirements from Knowit was to make a design system[3] to solve to problem at hand. These following are those requirements:

- The system needs to be applicable to all types of projects.
- The system needs to be modular, you can choose to just use parts of it.
- It has to be easy to change global parameters such as colors, fonts, margins, etc.
- The system will be open source and easy to make changes to.
- The system needs a thorough documentation.

After some research the conclusion was made that there would not be enough time to administrate the design system after it was built. Therefor the angle of attack was changed to focus on the making of components. The before mentioned requirements still stood but instead of building a design system we wanted to build an automated process to generate code from a design.

1.2 Objective

Most applications is built by components such as buttons, forms, cards[2], etc. These components are often redesigned and rebuilt from project to project making it bothersome but necessary work.

The aim of this study is to make this setup time extensively more efficient with a tool that generates components.

To make this possible the following research questions must be answered:

- Is it possible to automate the whole process from UI design program to browser runnable code?
- Can a user-friendly tool be built that automates component generation?
- Will automation between design and development increase communication between designers and developers?
- Does this tool speed up the development process and if so how much?

1.3 Demarcations

This automation could be used to create a whole component library and be a part of a whole design system. This project was done under the limited time of 20 weeks. Therefore only a basic component generator will be produced. With *basic* components the only things that will be taken in to account is shape, color, typography and layout. More advanced styling such as gradients, SVGs, etc. will not be apart of the prototype. The prototype will not be full accessibility becuase the elements used to build the components where only divs, for containers, and/or p-tags, for texts. The focus was on the data flow from design tool to usable code and therefore the component itself were not required to be perfect.

2 Background

present contextual or prerequisite information that is important or essential to understand the main body of the thesis

Arbetssätt & flöde

Competitors

The process of creating a web-application is typically done in three major steps:

- Design
- Develop
- Publish

The tool that this project is creating is will almost follow these three steps except the last one. Instead of publishing however the components that will be generated need to be distributed between designer and developer. Therefor ***publish***, for this project is replaced by ***distribute***.

All these steps are often, if not always, done iteratively within them selfs and as a whole cycle. However there must be some sort of design to start a meaningful and useful development effort. If there is nothing that has been developed there is nothing to publish. So therefore these three must be done in order.

The tool that this project produced has a part in all these three steps.
Design(Figma) –>Develop(Figma API, FigmaConverter) –>Distribute(package manager (NPM)).
In this section we will dive deeper in how to traverse these steps, and how it is possible to automate it.

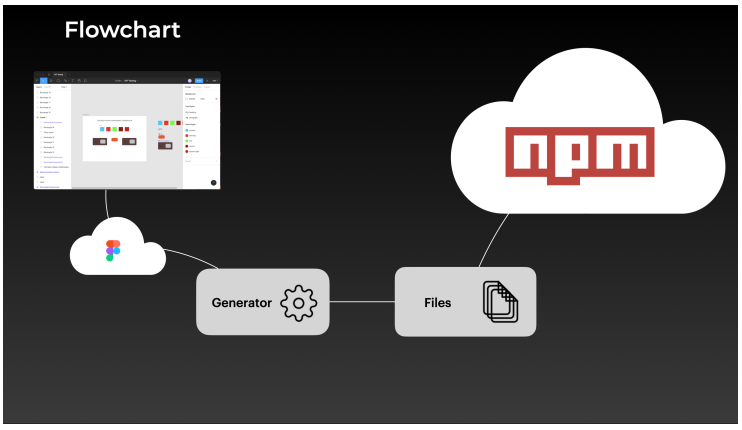

Figure 1: Chart over how data flows in the program

2.1 Competitors

The idea of making a design program generate functional code is not new. In this section some of the competitors of this genre of programs will be brought up and compared.

2.1.1 Webflow

Webflow was founded in 2013 and is a product from the famous program Y Combinator. Webflow allows the user to design, create and publish a website all from their program. Webflow is, as Figma, a network based application that works form a web browser.

Webflows website: [4]

2.1.2 Visly

Visly website: [5] Visly is was founded in 2018 and is very similar to Figma in how the user designs the product. Visly uses the design to create React components [6]. React is a component based JavaScript framework made by Facebook. Visly essentially makes it possible to create these components visually.

2.1.3 Bravo

Build Native IOS och Android apps with Figma. Think this can be the closest to the what I'm trying to do.

2.1.4 Comparison

3 Theory

The method of this study is being made possible from the theory that is displayed in this section.

3.1 Design

The design work of a project is largely held at the start of the project. Where the project team is in contact with the customer and hash out what the application and system will look like. With software there is often hard for a customer to know what they want and what is possible to do. This is a part of the design segment of the project and often needs some iterations to get things done.

A fairly regular way of making the design of websites is to first create prototypes with very low details and then build in details from there. This is Low Fidelity (LoFi) to High Fidelity (HiFi) prototyping. Where the customer can be a part of the steps to create something that they want and can be done. The medium with which these prototypes are created varies but most often than not a User Interface (UI) design application is used to make the final HiFi-prototype. This HiFi-prototype is visually accurate with the final product meaning that all colors, typographies and layout are complete. This then needs to be translated into code by the developer.

3.1.1 UI Design Applications

User interface design is often the first step in build an application. This is done to ensure the customer that the product is going to look as they want it to and that the team building the application is on the same page. **(Komma överens)**. Sketch [7] was released in 2010 and was one of the first UI design applications and lead the marked for may years. In latter years applications like Figma [8] (released 2016) and Adobe XD [9](released 2015) has come to take over Sketches overwhelming dominance.

In this thesis Figma is the tool that is being looked at because this tool is already being used by Knowit. Figma is also one of few design applications that have an open API which makes this project possible.

3.1.2 Figma

Figma is a UI Design application which is web based. Meaning that the whole program is run over network.

Figma has an open REST-API, (Representational State Transfer), that supply's the information of the Figma document over the Internet [10], [11]. Figma is also web-based meaning that the program runs over a network connection. Because of this the API is constantly updated after each change in the design, which great for collaboration and getting the API constantly updated.

3.1.3 Figma Components

Figma also allows its users to create components. This is a set of elements that is combined to a component. This is for example a button that could be used in many different places around the UI. A component is a great use case for this because when a component is made all its copies or children *"looks"* at the original component. This means that if a change is needed to be done to that component the original is the only one that is needed to be changed.

3.1.4 Figma Styles

Figma has a feature that let's the user store color, text and effect styles. These are a way for the user to store default styles for their design. This style can later be used in many different elements. For example if the default color for a design is green the user can store this green color as a color style. If the user changes its mind after a while and they want the default color for a design to be blue. The only thing that the users then needs to do is to change the color style and all the elements that uses this default color will change its color. The same principle extends to typographies and effects such as shadows and blurs too.

3.2 Develop

Most modern websites is run with DHTML (Dynamic HTML), which is built by HTML, CSS and JavaScript.

In modern web design there are a lot of different frameworks and languages that makes it easier for developers to make create products but all these tools have the one thing in common. They all convert to one of the native browser languages.

One of the goals of the thesis is to create a tool that could be used in as many projects as possible. Knowit is, as explained in the intro, a fairly large company and has a lot of different projects with lots of different frameworks.

3.2.1 REST API

REST stands for **Re**presentational **S**tate **T**ransfer and is a architectural style of distributed hypermedia systems. That was created by Roy Fielding in 2000 with the release of his dissertation [12]. For an API to be called *RESTful* it needs to fill six requirements [13].

1. Client–server - the UI and data storage are separated
2. Stateless – The server does not store any information of the client. The client must provide all information for every request.
3. Cacheable - *Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If*

a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests. - SIMPLIFY

4. Uniform interface – *By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components. REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.* - SIMPLIFY
5. Layered system - This architecture consists of hierarchical layers that constrains what each components can do, such as a component can only interact with the layer that it is on.
6. Code on demand (optional) - This allows the client to download and execute applets or scripts and there for extending client functionality

All of these requirements makes it light weight and very easy to understand and thereby introducing fewer problems into the system.

3.2.2 JavaScript and TypeScript

JavaScript is a programming language that reports its error much later than many other languages. Variables can take any shape in JavaScript. That is for instance a number or a string. This can at first seem as something good that the language is highly dynamic but this is also very error prone. Where most of the errors are discovered after the program is run.

Incremental testing is required. Due to its highly permissive, error-tolerant nature, JavaScript programming requires an incremental, evolutionary approach to testing as well. Since errors are reported much later than usual, by the time an error is reported it is often surprisingly difficult to pinpoint the original location of the error. Error detection is made harder by the dynamic nature of JavaScript, for instance, by the option to change some of the system features on the fly. Furthermore, in the absence of strong, static typing, it is possible to execute a program and only at runtime realize that some parts of the program are missing. For all these reasons, the best way to write JavaScript programs is to proceed step by step, by writing (and immediately testing) each new piece of code. If such an incremental, evolutionary approach is not used, debugging and testing can become quite tedious even for relatively small JavaScript applications. [14]

3.2.3 Node.js

Node.js is an open source project that lets its users run code on the server asynchronously. Which is a vey efficient way of running JavaScript code without a browser.

3.2.4 Syntactically Awesome Style Sheets (SASS)

SASS is an extension language to CSS that makes CSS supercharged. With SASS it is possible to have the stylesheet split up into multiple files, create functions, etc. SASS is then compiled to regular CSS so the browser is able to understand it. SASS is also called a preprocessor for CSS because of this. SASS has two different syntaxes, the indented syntax commonly referred to just SASS and Sassy CSS (SCSS). The indented syntax was the original syntax for SASS and is only dependent on indentation. SCSS syntax is very similar to regular CSS but with the qualities of SASS. Because of the resembles of normal CSS SCSS is the most easy to learn and most popular of the two syntaxes. For this project SCSS will be used because of the resembles to CSS.

3.2.4.1 Variables

Variables in a stylesheet is useful especially when dealing with colors. Often a website has a set color scheme from the design. This can easily be set as a variable if the colors needs to be changed for some reason. Then there is only one entry that needs to be changed. Regular CSS do support variables but they are a little "clunky". To set a variable you set two dashes infront of the variable name. The clunky part of this impolementation is that you cannot use this variable as it is later on in the code. You have to surround the variable with *"var()"*. An example of this can be seen below.

```
1  --MyColorVariable: #ff9a67;
2
3  div{
4    background-color: var(--MyColorVariable);
5  }
```

What SCSS

3.2.4.2 Mixins

3.2.5 Web Components

Web components are custom made components that are reusable.

Web components are built in to HTML5 which makes them work natively in all major browsers and in congestion with all frameworks. Which makes web components the perfect candidate for this project.

This [15] could be very useful.

3.2.5.1 Custom Elements

A set of JavaScript APIs that allow you to define custom elements and their behavior, which can then be used as desired in your user interface.

3.2.5.2 Shadow DOM

A set of JavaScript APIs for attaching an encapsulated "shadow" DOM tree to an element - which is rendered separately from the main document DOM - and controlling associated functionality. In this way, you can keep an element's features private, so they can be scripted and styled without the fear of collision with other parts of the document.

3.2.5.3 HTML Templates

The <template >and <slot>elements enable you to write markup templates that are not displayed in the rendered page. These can then be reused multiple times as the basis of a custom element's structure.

3.2.6 LitElement

LitElement is a great light weight class to simplify making web components[16]. LitElement is built by the Polymer Project [17], which is a group of engineers from the Google Chrome team. LitElement combines functionality from custom elements, shadow DOM and HTML templates to a class that make it easy to create these web components with concise and malleable code.

3.3 Distribute

To be able to use the components created from the designer the developers need to be able to get a hold of them. A lot of reusable code that is crated for the web can be accessed from the internet with the help of package managers.

3.3.1 Package Manager

[18]

A package manager is a way to install and update programs with ease. The package manager that this project will use is Node Package Manager (NPM) which is the worlds largest software registry[19].

3.4 Usability testing

Usability testing or "user research" is a very broad term. As Lewis [20] described it: Usability testing, in general, involves representative users attempting representative tasks in representative environments, on early prototypes or working versions of computer interfaces.

Usability testing is essentially done to find flaws in an interface by putting the user in the environment of using the interface. Usability testing is done in all stages of development. From paper prototypes to screen mock-ups with no functionality to implemented existing systems.

Usability testing can be considered a cousin to traditional research methods. When similarities can be found in experimental design with measurement of task performance and time performance, surveys and, observation techniques from ethnography. The participants in usability testing, as in traditional research, must remain anonymous, be informed of their rights and have the ability to leave the research at any time. What separates usability testing from traditional research is often the end goals. Where usability testing has an industrial approach where there isn't

Why do we need user testing? When to use it??

So instead of saying, "how many users must you have?," maybe the correct question is "how many users can we afford?," "how many users can we get?" or "how many users do we have time for?" [21]

[22]

[23]

3.5 A/B Testing

3.6 Statistical analysis

4 Method

This system was thought of as a tool to be used in many different projects. Because of this the mark on accessibility was big. The developer should not be constrained on what framework or lack there of he or she is using. This should work on all platforms.

4.1 Workflow and Actions taken

1. Literature study / External analysis
    - Research which tools to use
2. Preform interviews/collect data for the workflow that is used today in the company.
3. Create a tool that uses Figmas API to create components (testing accessibility/"code quality" trough out)
4. Preform user testing to ensure usability.
5. Finalize tool with certain demarcations.
6. Preform AB testing to study efficiency with and without the new tool for starting up a project.

4.2 Literature study

How to communicate? How to build relationship? How to easier build products?

The subject at hand was thoroughly

4.3 Creating the tool

Meetings with Knowit and where discussions of what could be possible to do under the 20 weeks of work that should be carried out. As seen from the literature study the big problem was how to condense the elements in from the code to easly be used.

Identifying the tools that should be used.

- TypeScript - for building the tool.
- Exploring Figmas API.
- LitElement - For building the components.
- Building the html elements and styling with recursive functions

To build the tool an experimental approach with **TypeScript** was used. This means that code where tested and possibilities where explored during the building of the program. Why **TypeScript**? because of the unknown possibilities at

the start of the development there where some thoughts on making the generator part of the component itself. Thereby the browser had to be able to run the code which made JavaScript a good match. Because of the error prone nature of JavaScript (discussed in section 3.2.2) TypeScript, the supercharged version of JavaScript, was chosen. TypeScript need to be compiled to JavaScript before it can be run in a browser but that hassle is worth the benefits of getting error warnings before runtime.

To understand what could be done with Figmas API it first had to be examined. The website `https://www.figma.com/developers/api` was read through and also some initial HTTP-requests where sent to the API, using platform Postman [24]. The initial response from the API where quite large. This meant that setting the TypeScript types correct for all values in the response would take alot of time. To mitigate this the Visual Studio Code [25] extension quicktype[26] was used to generate types from the JSON response.

From the response we could see that most, or at least enough, of the data were the same as styling in CSS. This meant that styling elements with CSS was possible from the API. Styling is one thing but how do we build the skeleton of the page? the HTML. HTML was born 28 years ago. Back then modularity and component compatibility where not important. The first suggestion to solve this issue was to build a separate HTML-file and then use that as a sub-file to the main HTML-file. This could've worked for a static page with some browsers. But the requirements for the project are that the components can be used for all frameworks too. To solve this the tool LitElement was used. LitElement (3.2.6) is a class that builds web-components (3.2.5.1), HTML templates (3.2.5.3) and shadow-DOM (3.2.5.2). Which combined makes for a really good way of building components. LitElement is a class in TypeScript and JavaScript where we choose to use the TypeScript version to keep the program code consistent.

Should mention that the coding method used is object-oriented, but don't know where

The information from Figmas API is stored as classes of colors, typographies, and components. Where the components class can contain the other two. The strategy was to generate a string that contained the LitElement. Essentaially the program generated code as a string. This string is later written into a new file.

4.3.1 Building the skeleton of the component

To build the HTML inside the LitElement the data-object from Figmas API where run through a recursive function that is run on the component and all it's children (elements).

4.3.2 Styling the Component

For the first "attempt" styling where done in a similar way but because of the nature of the shadow-DOM every CSS-attribute where assigns a property which was reachable from outside the component. This was later redesigned because of the fact that the user could not add a new CSS-attribute to the component if they wished to. To fix this the CSS-attributes where stored in maps [27] and then pushed in to a style element. Instead of creating a property for each style attribute, only one property for each element is created. If the user wishes to add and/or change the styling of an component they target the element as an attribute to the component and inputs regular CSS. The component then creates a duplicate of the styling map for the targeted element and inserts the new styling attributes to the component.

4.3.3 Variables

Figma has a feature called styles. This a way for the user too store and reuse, colors, texts and effects. This is somehting that is also very normal to do in a developer environment. Therefore a decision was made to create a SCSS *variable* file where these would be stored. Because of the time constraint of the project only colors and texts where implmented. This was done similar to the components where the "code" for the SCSS variables where written to a string that later were written into a SCSS file.

4.3.4 Userguide

The program built does not have a graphical user inteface, again because of the time constraint. The user is instead using a command-line interface (CLI) instead. This makes it a bit harder to learn because there are no visual queues of what to input to the program. To solve this a userguide was created in the form of a README on GitHub. This userguide was written as the program were developed. When the program was functional usertesting where preformed to test the validity of the userguide and if there were any prototype breaking bugs. ***there was and they were fixed... atleast soon.***

4.4 semi-structured interviews

this tool is involved with people in many different areas of expertise, form designers to front-end developers to back-end developers. to get a clear view of of how these people work to create the best tool for them the semi-structured interview was used.

for collecting data from the employees of knowit semi-structured interviews were used [28].

4.5 user testing

the user testing

4.6 finalized prototype

4.7 ab testing

5 Result

No results to show yet

6 Discussion

Discussion about the results and why it turned out the way it did. Hopefully this created a great tool that will take over the world

6.1 Future Work

Futher developement should be done with:

- Gradients
- package.json for output

References

[1] D. Cohen, M. Lindvall, and P. Costa, "An introduction to agile methods.," *Advances in Computers, Vol 83*, vol. 62, no. 03, pp. 1–66, 2004.

[2] N. Babich. (Jun. 22, 2020). "Simple Design Tips for Crafting Better UI Cards," Medium, [Online]. Available: `https://uxplanet.org/simple-design-tips-for-crafting-better-ui-cards-19c1ac31a44e` (visited on 03/28/2021).

[3] W. Fanguy. (). "A comprehensive guide to design systems — Inside Design Blog," [Online]. Available: `https://www.invisionapp.com/inside-design/guide-to-design-systems/` (visited on 02/28/2021).

[4] (). "Responsive web design tool, CMS, and hosting platform — Webflow," [Online]. Available: `https://webflow.com/` (visited on 02/26/2021).

[5] Visly. (). "Visly," [Online]. Available: `https://www.visly.app` (visited on 03/03/2021).

[6] F. Inc. (). "React – A JavaScript library for building user interfaces," [Online]. Available: `https://reactjs.org/` (visited on 03/18/2021).

[7] Sketch. (). "The digital design toolkit," Sketch, [Online]. Available: `https://www.sketch.com/` (visited on 03/11/2021).

[8] Figma. (). "Figma: The collaborative interface design tool.," [Online]. Available: `https://www.figma.com/` (visited on 02/09/2021).

[9] Adobe. (). "Adobe XD — Fast & Powerful UI/UX Design & Collaboration Tool," Adobe, [Online]. Available: `https://www.adobe.com/se/products/xd.html` (visited on 03/11/2021).

[10] Figma. (). "Figma," Figma, [Online]. Available: `https://www.figma.com/developers/api` (visited on 02/09/2021).

[11] *Representational state transfer*, in *Wikipedia*, Feb. 5, 2021. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Representational_state_transfer&oldid=1004990383` (visited on 02/09/2021).

[12] R. Fielding. (). "Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST)," [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (visited on 03/11/2021).

[13] restfulapi.net. (). "What is REST," REST API Tutorial, [Online]. Available: https://restfulapi.net/ (visited on 03/11/2021).

[14] A. Taivalsaari, T. Mikkonen, D. Ingalls, and K. Palacz, "Web browser as an application platform," in *2008 34th Euromicro Conference Software Engineering and Advanced Applications*, IEEE, 2008, pp. 293–302.

[15] (). "Web Components — MDN," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Web_Components (visited on 02/09/2021).

[16] Polymer. (). "LitElement," [Online]. Available: https://lit-element.polymer-project.org/ (visited on 02/26/2021).

[17] polymer. (). "Polymer Project," [Online]. Available: https://www.polymer-project.org/ (visited on 03/29/2021).

[18] *Package manager*, in *Wikipedia*, Dec. 26, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Package_manager&oldid=996441348 (visited on 02/10/2021).

[19] (). "About npm — npm Docs," [Online]. Available: https://docs.npmjs.com/about-npm (visited on 03/29/2021).

[20] J. R. Lewis, "Usability testing," *Handbook of human factors and ergonomics*, vol. 12, e30, 2006.

[21] J. Lazar, J. H. Feng, and H. Hochheiser, *Research Methods in Human-Computer Interaction*. Morgan Kaufmann, 2017.

[22] J. Nielsen, "Estimating the number of subjects needed for a thinking aloud test," *International journal of human-computer studies*, vol. 41, no. 3, pp. 385–397, 1994.

[23] ——, "Heuristic evaluation, w: Nielsen j., mack RL (eds.), usability inspection methods," 1994.

[24] (). "Postman — The Collaboration Platform for API Development," Postman, [Online]. Available: https://www.postman.com/ (visited on 04/23/2021).

[25] (). "Visual Studio Code - Code Editing. Redefined," [Online]. Available: https://code.visualstudio.com/ (visited on 04/23/2021).

[26] (). "Convert JSON to Swift, C#, TypeScript, Objective-C, Go, Java, C++ and more • quicktype," [Online]. Available: https://quicktype.io/ (visited on 04/23/2021).

[27] (). "Array.prototype.map() - JavaScript — MDN," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map (visited on 04/25/2021).

[28] A. Galletta, *Mastering the Semi-Structured Interview and beyond: From Research Design to Analysis and Publication*. NYU press, 2013, vol. 18.