,

Automated Functional Tests for a Web Application

Erik Svensson Fahlström
Johan Wulf

Department of Electrical and Information Technology
Lund University

Supervisor: Christin Lindholm

Examiner: Christian Nyberg

June 17, 2022

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

When developing a large application, many companies use techniques such as Test Driven Development in order to get a range of tests on which they later can validate their code with. This has become the standard in many work environments. Problems arise when an application is already developed and tests has not been written in the course of the development. This makes it near impossible to validate the entire code base without spending hundreds of hours writing unit tests which is a problem when engineers costs a lot of money. It is in a companies interest to then find ways to write tests that cover larger parts of an application.

The People Domain team at IKEA has developed an application, where the need for tests was brought to light first when large parts of the application already was developed. This is why this thesis, in coordination with IKEA, investigates and implements an automatic test flow which is run when a developer pushed code to the applications repository. The application is developed by IKEA to make it easier for co-workers to effect their schedule. This is done by creating time slots which are used as requests for when someone want, or does not want, to work.

Since there were no previous tests of the UI made by IKEA, this was a new area to discover. It became apparent that it was harder to automate the entire test flow than first believed. This due to problems with the way that the application authenticates users. The authentication is done through Microsofts portal, which means that it is not the application itself that authenticates. It would require a large re-factorization of the applications structure to make tests work as wanted. To implement the test flow, a number of frameworks and libraries were investigated and evaluated and choices were made for the solutions that seemed to fit IKEAs needs the best. With a combination of meetings, interviews and literature studies information was gathered, both about the application and about the testing of software.

The thesis concludes in a test suite which automatically tests parts of the application. The tests are written with help from the test framework Cypress, and tests the functionality to create a new time slot.

Keywords: Test, automation, single sign on, frameworks, JavaScript

Sammanfattning

Vid utveckling av stora applikationer använder sig många företag av tekniker så som testdriven utveckling för att skapa en stor samling av test som de sedan kan använda sig av för att validera skriven kod. Detta har blivit standarden på många arbetsplatser. Problem uppstår när en applikation redan är utvecklad och test ej har skrivits under utvecklingens gång. Detta gör det näst intill omöjligt att validera koden utan att spendera hundratals timmar på att skriva enhetstest vilket är ett problem när ingenjörer kostar mycket pengar. Det är då i ett företags intresse att hitta sätt att skriva test som täcker större delar av applikationen.

People Domain teamet på IKEA har utvecklat en applikation, där behovet av tester upptäckts först efter att stora delar av applikationen redan är färdiga. Därför undersöker och implementerar detta examensarbete, i samråd med IKEA, tester till ett särskilt flöde i applikationen. Målet med examensarbetet är att implementera ett automatiskt testflöde som körs när en utvecklare lägger till ny kod i applikationen. Applikationen är utvecklad av IKEA för att underlätta hur medarbetare kan påverka sina arbetstider. Detta görs genom att lägga in önskemål för vilka tider man vill och inte vill arbeta.

Då det inte fanns någon tidigare testning av användargränssnittet från IKEA så blev detta ett nytt område att utforska. Det visade sig vara svårare än först beräknat att automatisera hela flödet, vilket berodde på problem med att autentisera användare. Autentiseringen sker via Microsofts portal, vilket gör att det är inte själva applikationen som autentiserar. Problemet visade sig ligga i hur applikationen sedan hanterade autentiseringen från Microsoft. Detta hade krävt en större refaktorisering av applikationens struktur. För att kunna implementera detta testflöde undersöktes ett antal olika ramverk och bibliotek som sedan utvärderas och val gjordes för de som ansågs passa IKEAs behov bäst. Med en kombination av möten, intervjuer samt litteraturstudier inhämtades information, dels om applikationen och dels om testning av mjukvara.

Examensarbetet resulterar i ett testramverk som automatisk testar delar av applikationen. Testerna är skrivna med hjälp av ramverket Cypress, och testar funktionaliteten för att lägga in ett önskemål om arbetstid.

Nyckelord: Test, automation, enkel inloggning, ramverk, JavaScript

Acknowledgments

This Bachelor's thesis was made in collaboration with the People Domain team at IKEA, to which we would like to extend an enthusiastic thank you for giving us the opportunity to write our thesis with your team. It was a great experience, and we both have learned more than we would have ever expected.

This thesis would not be possible without our supervisors, Christin Lindholm from Lunds Tekniska Högskola and Jonas Gustafsson. Therefore, we would like to thank Christin for being a great resource for thesis writing, providing us with knowledgeable answers at all times, and Jonas for making sure that we always progress and making us feel welcome to the team.

We also want to thank Damodar Manthripragada, Ioan Rosgrim and Shameer MP from IKEA for being to great help for our technical questions, and for helping us understand the flows within the applications.

Lastly, we want to thank Christian Nyberg for being our examiner and making sure that our thesis is excellent!

Table of Contents

Chapter 1

Preface

In this thesis, author Johan Wulf has developed tests for the co-worker application and Erik Svensson Fahlström has developed test for the UPPS application. The thesis has been written together.

Chapter 2
Introduction

This chapter contains the background on which the Bachelor thesis is based upon. It also contains a description of the company that the thesis is written in cooperation with, IKEA. Also, the purpose and goals of the thesis is contained within this chapter.

2.1 Background

IKEA is a multinational furniture retailer with 374 stores in 30 countries, employing over 166000 co-workers. The vast majority of all co-workers are retail store workers. In order to improve employee retention, a need for co-workers to influence their schedule was discovered and thus My IKEA Availability (MIA) was developed. Before the planner sets up the schedules for co-workers, they are able to add their availability in the MIA tool. The input information is then taken into consideration when creating schedules for the thousands of co-workers at IKEA.

2.1.1 People Domain

The People Domain department develops tools for IKEAs employees to make certain tasks easier for both ground floor workers and managers. In other words, they are developers for HR related software. Such software could be, scheduling, handling sick leave and other systems that improve the relation between managers and co-workers.

2.1.2 Test Enabler team

The Test Enabler team is a team who specializes in testing. They act as a support for developers when developing tests for applications and so forth. The team itself does not write test for developers but they have great knowledge on different frameworks and ways of testing. They can be a great asset to use when needing help with testing.

2.1.3 My IKEA Availability

My IKEA Availability is a web application which will be used to collect preferences regarding working hours from IKEAs employees. The preferences are put

in to MIA by each individual employee, by signing in to IKEAs internal platform and navigating to the MIA application. This makes it possible for the schedulers to retrieve this data when it is time to make the schedules. MIA enables the scheduler to take as many working hour preferences in to consideration as possible when creating the schedule.

There are four different types of availability that a co-worker can set themselves as.

1. Unavailable
2. Available
3. Available - prefer to work
4. Available - prefer not to work

Until now, the functionality of the application has been tested in an unorganized and manual way. The focus has been to deliver new functions for the app, not to ensure of good coding standards. Each developer has had the responsibility to perform function tests when they have added code to the repository. When launch is coming closer, the focus needs to shift to ensure quality and for that the tests needs to be done quicker. This requires automatic testing.

To achieve automatic testing of MIA, test frameworks need to be examined and evaluated. IKEA has proposed two frameworks which, with others, will be investigated, Mocha and Selenium. The goal is for the tests to be ran automatically when a developer pushes code to the MIA application repository.

The focus of this thesis will be on automatically testing the frontend of the applications.

2.2 Purpose

The purpose with the thesis is to examine how to implement an automatic test flow for the MIA application. Furthermore, the thesis shall also examine and evaluate different test frameworks which can be used to test the application. After evaluation, a test framework is to be chosen and tests will be written. The thesis is expected to create an automatic test solution for the application in the form of a test flow which will be run every time a developer pushes code to the repository. A flow in this case would be how a certain user interacts with the application. This is done to simplify the developers work with the application.

2.3 Goals

The thesis will examine how to automate tests of IKEAs application MIA. Tests for testing the applications UI will be written and implemented. An automated test flow will be developed and a proof of concept will be shown as a final result.

2.4 Problem Specification

This thesis aims to answer the following questions:

1. How is the MIA application tested at the moment?
2. In which ways can the tests be automated?
3. Which framework is best suited for running tests on MIA?
4. How do you automate browser activity?

2.5 Motivation

Testing is becoming an increasingly larger part of software engineers´ life. It is now, in most places of work, expected that everyone that writes code also knows how to test it. Many companies have a policy where your code is not seen as finished if the creator has not written tests to confirm the functionality of the code.

As projects grow larger, a need for test automation will develop. It is unsustainable to expect developers to test their code every time they commit code to shared repositories. Furthermore it is also a must to continuously test functions to ensure proper execution of the code.

2.6 Scope Limits

In order for the thesis to be carried out in the set time constraints, a handful of limits has to be set.

- The tests will not cover the entire application, but instead specific flows that's decided on in accordance with IKEA.
- The MIA application is written in JavaScript, with Node.js, and thus this thesis will not investigate other programming languages.
- The test flow will only be written to use the Microsoft Edge browser.
- A desktop viewport will be the only viewport used for testing the application.

Chapter 3
Technical background

This chapter will describe what software, framework and techniques that were used during the thesis work. It will also clarify the technology stack that IKEA has used to develop My IKEA Availability.

3.1 Technical stack

This section is to give a better understanding of which libraries and frameworks IKEA has used to develop My IKEA Availability, and their relevance to the thesis.

3.1.1 Node.js

Node.js is an open source JavaScript runtime environment, used for making back-end applications with JavaScript [1]. Node.js runs on a single thread, making it sub optimal for CPU heavy operations. It is usually used for making web sites and web based APIs. It is "an asynchronous event-driven JavaScript runtime" [1]. As a backend, MIA uses Node.js.

3.1.2 Node Package Manager (npm)

Node Package Manager is a package manager included in Node.js. It is used to keep track of and use both the developers own packages and others for JavaScript [2]. By pulling these packages into existing projects, it is a simple way to update, delete or install packages. The Node Package Manager could be used in any Node.js based JavaScript project. To use Node Package Manager in a project, the user must have a package.json file to keep track of which packages are being used and what version. Since Node Package manager is installed locally on every computer it must always installed on new computers, to be able to work on existing projects.

3.1.3 React

React is "A JavaScript library for building user interfaces" [3] created by Facebook. It is component based, meaning that a developer can build multiple components, for instance navigation bar, header and footer separately to then combine them for a full page.

React works by manipulating the DOM, Document Object Model, to build a web page. The Document Object Model is a "programming interface for web documents" [4] that is used to generate and manipulate web documents which makes up the world wide web.

When building a website, often it is a requirement that the page is interactive. That is, it changes based on what happens with data in the background. One such example is getting a new message on Facebook and instantly getting a notification for it even though you have not updated the page. Updating the entire DOM is very costly, which lead to React being developed.

In React, other than the users actual DOM, the React library also runs a virtual DOM. When changes happen which needs for UI to be updated, the updated UI is first generated by React virtually, and then the DOM is compared to the users actual DOM. React then computes the way which to update the users DOM in a way requiring the least amount of resources as possible, see figure 3.1.

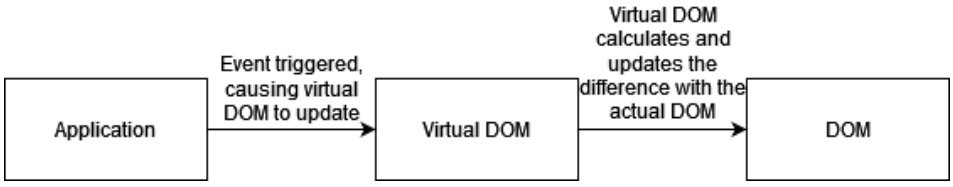

Figure 3.1: Flow of the DOM in React

The application My IKEA Availability is built using React for front-end rendering.

3.1.4 PostgreSQL

PostgreSQL is an open source database [5], which handles both relational (SQL) and non relational (JSON) querying. PostgreSQLs architecture is object-relational and multiprocessing. Since it is object-relational based it could handle more complex data types than a relational database. For each client connection to the database it uses its own memory, by doing this it will demand a high memory capacity for systems which have many client connections. The MIA application uses PostreSQL as the database of choice.

3.2 Test frameworks, libraries and tools

The following frameworks, libraries and tools were seen as possible candidates for use in the development of the test suite for MIA. They were all investigated and evaluated as such.

3.2.1 Mocha

Mocha is a JavaScript test framework made to run on Node.js [6]. Mocha has different interfaces for testing, such as Behavior-driven development, BDD and Test-driven development, TDD. BDD is an agile development process that focuses on collaboration between developers to give a better understanding on how an application should work. TDD is a method were tests are written before the code, these tests are constructed depending on which requirements the software has. These interfaces that come with Mocha differs in the way tests are constructed, but they still work in the same way.

To use assertions in Mocha, a developer could choose an assertion library by themselves or else Mocha will use its default library which is a built in library in Node.js. Mocha allows the user to run their tests both asynchronous and synchronous. Running asynchronous allows tests to wait for functions to run before the test is completing, synchronous allows the test to continue without having to wait.

Example of test in Mocha

```
1 describe('Index of element', function() {
2     context('Is Mar present', function() {
3         it('Should return the index of the first apperance
              of Mar in the array', function() {
4             ['Jan', 'Feb', 'Mar', 'Apr'].indexOf('Mar').
                  should.equal(2);
5         });
6     });
7 });
```

Inspiration for the example has been taken from [7].

3.2.2 Chai

Chai is an assertion library for testing NodeJS applications [8]. Since Chai is only a library it is often paired with a test framework of choosing. Chai uses different kind of interfaces to make sure tests are passing, for example the Assert interface.

Example of test in Chai

```
1 var assert = require('chai').assert
2   , month = 'January';
```

```
3
4 assert.typeOf(month, 'string'); // To just see if the test
      passes
5 assert.typeOf(month, 'string', 'Month is a string'); //
     Adds a message to the test output
```

Inspiration for the example has been taken from [9]

As shown in the example above different kind of calls could be made to make sure functions or variables is giving the correct output. On line four and five in the example one could make sure the type of the variable foo is a string with or without a message in the result. If the variable foo is a string this test passes.

3.2.3 Selenium

Selenium is a framework that automates web browsers [10]. It has support for a range of different programming languages, such as Java, Python, Ruby, JavaScript and Kotlin, and browser drivers, such as Chrome, Firefox, Edge, Internet Explorer and Safari. Selenium comes packaged in the Node Package Manager, npm. The framework makes it possible for a developer to simulate user input on the front-end of a web application. To use in automated web testing, a developer would need to combine Selenium with a test/assertion library, such as Mocha and Chai.

Setup

To install Selenium npm is used since it is a Node package.

```
1 npm install selenium-driver
```

Afterwards, the developer needs to select a web driver themselves since Selenium does not come with browser support out of the box [11]. When a web driver has been chosen, the developer instantiates it. The following example uses Chrome as the web driver.

```
1 let driver = await new Builder().forBrowser('chrome').
     setChromeOptions(options).build();
```

Afterwards, we can use the "driver" object to call for any other functions, such as navigation and interactions.

Navigation

To navigate to a website with Selenium, one would run the following command:

```
1 await driver.get('https://google.com/');
```

This would land us on the Google homepage, just as navigating to it through your browser would.

Locating elements

Selenium allows developers to select specific elements on a page. This can be done with the "Locators" and "Finders" that Selenium offer. Locators are a range of different selectors that can be utilized with Selenium. The locators can find elements based on a lot of different attributes, such as class name and id. Selenium offer a wide range of locators that can be used to find elements on the pages [12].

Utilizing the locators, you can pass them to the "Finder" function of the Selenium webdriver. An example follows where a div with the id of "selectme" is selected.

The HTML element:

```
<div id="selectme">..</div>
```

The select statement with Selenium:

```
var divLocator = driver.findElement(By.className('selectme
    '));
```

Interactions

It is also possible to interact with a web page through clicks and key entry. If you want to find an input element with the id "selectme" and type "text" in the element, one would do as follows.

The HTML element:

```
<input id="selectme">..</input>
```

The select statement with Selenium:

```
var inputElement = driver.findElement(By.id('selectme'));
await inputElement.sendKeys("text");
```

This would write "text" in the input element with the id "selectme".

3.2.4 Puppeteer

Puppeteer is a Node.js library allowing developers to automate web browsers [13]. It is built on Chrome using the DevTools protocol [13][14]. Puppeteer comes packaged in the Node Package Manager. Puppeteer makes automating web behaviour possible.

Setup

As Puppeteer is a package in npm, we use the package manager to install the package.

```
npm install puppeteer
```

Puppeteer comes with Chromium in the package, so it is good to use out of the box. In our JavaScript file we would write as follows, to instantiate Puppeteer. Note tho, that the Puppeteer package is only for automating browser activity. Thus, in order to perform automated web tests, one would have to use a test/assertion library, such as Mocha and Chai.

```
1  const browser = await puppeteer.launch();
2  const page = await browser.newPage();
```

Now, we can go to use other functions, using the page, such as navigation and interactions.

Navigation

To navigate to a website with Puppeteer, one would run the following command:

```
1  await page.goto('https://google.com/');
```

This would land us on the Google homepage, just as navigating to it through your browser would.

Locating elements

Puppeteer uses CSS selectors [15][16] to identify and manipulate HTML elements. These are built in to the CSS styling language. By using the CSS selectors, one can find different elements on the webpage. An example follows here a div with the id of "selectme" is selected.

The HTML element:

```
1  <div id="selectme">..</div>
```

The select statement with Puppeteer:

```
1  var divLocator = await page.waitForSelector('#selectme');
```

Interactions

It is also possible to interact with a web page through clicks and key entry. If one want to find an input element with the id "selectme" and type "text" in the element, one would do as follows.

The HTML element:

```
1  <input id="selectme">..</input>
```

The select statement with Puppeteer:

```
1  await page.type('#selectme', 'text');
```

This would write "text" in the input element with the id "selectme".

3.2.5 Cypress

Cypress is "a complete end-to-end testing" framework, used for automating tests for browser applications [17]. It has support for the web browsers Chrome, Firefox and Edge [18]. As Cypress is a complete test framework, it comes with both assertions and browser automation in the package. This comes from Mocha and Chai that Cypress uses as bundled tools [19]. Thus, a developer would not need any other libraries than Cypress to get started.

Setup

As Cypress is a npm package, we use the Node Package Manager to install it.

```
npm install cypress
```

Afterwards, the tests are ready to be written.

Navigation

As Cypress comes with assertions out the box, the developer is expected to write the tests at the get go with Cypress - which differs from Selenium and Puppeteer. A test that would navigate to the page google.com and assert that one ended up on the correct page would look as follows.

```
describe('Visit Google', () => {
  it('goes to google.com', () => {
    cy.visit('https://google.com/')
  })
})
```

The test by the name of "Visit Google" which has the expected behaviour of "goes to google.com" has now been described.

3.3 Single Sign On

Single Sign On (SSO) is a way for organizations to use the same login for different sites, applications and devices [20]. For instance, if one enters a work mail and password when logging in to a laptop, the computer now knows your identity and can use it for other applications that also uses SSO. When logged in to the laptop, you automatically get authorized to view applications.

3.4 Visual Studio Code

Visual Studio Code is an IDE developed by Microsoft with support of Windows, Linux and MacOS [21]. It supports a large number of programming languages, and also allows developers to make their own extensions for the program which can be shared on the marketplace. The People Domain department uses Visual Studio Code for the development of My IKEA Availability.

3.5 INGKA Technology Radar

INGKA Technology radar is a diagram. This diagram gives a overview about different tools and frameworks and so forth [22]. It can be used by employees to keep up to date which technologies are being used, adapted and phased out within IKEA.

Chapter 4
Methodology

This chapter contains the methodology for which the thesis was carried out, and which phases the thesis went through. The chapter also contains information about how communication between the thesis workers and IKEA was carried out, as well as source criticism for all sources cited in the thesis.

4.1 Phases

The thesis was split up into four phases, see figure 4.1. Initially, the plan was to carry out the two first phases in an iterative manner [23] and the following two sequentially. This was due to the thesis workers believing once development began, it would not require any new data gathering and analysis.

As was found out though, implementation issues with the tests required the thesis workers to gather more data about the application and on how to perform tests on it. Thus, the first three phases were done iteratively, and the last phase came sequentially after development was done.

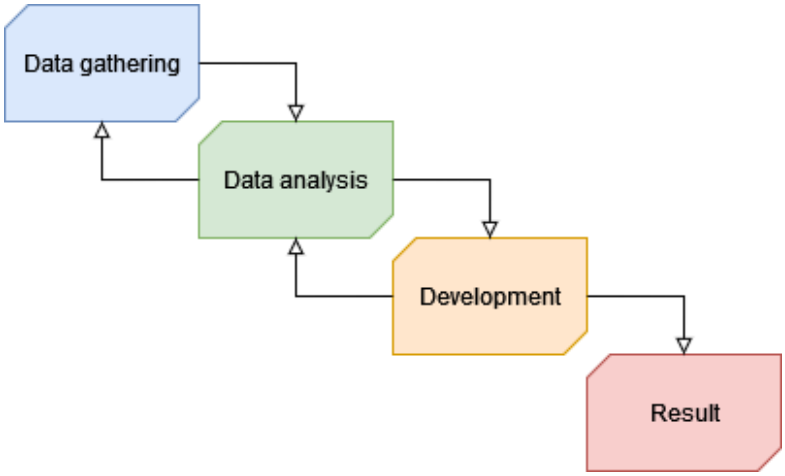

Figure 4.1: Flow of the order of which the phases were carried out

The phases are explained in detail in section 4.2 through 4.5

4.1.1 GANTT

In the beginning of the thesis, a time plan GANTT-schedule was made, which was to be used as a guide for when certain things needed to be worked with, see figure 4.2. It quickly became apparent that the time schedule would not be held. This was due to many factors, such as IKEA having deadlines to hold and the thesis woprkers having exams. An updated GANTT-schedule with the actual time frame will be displayed in the Analysis chapter.

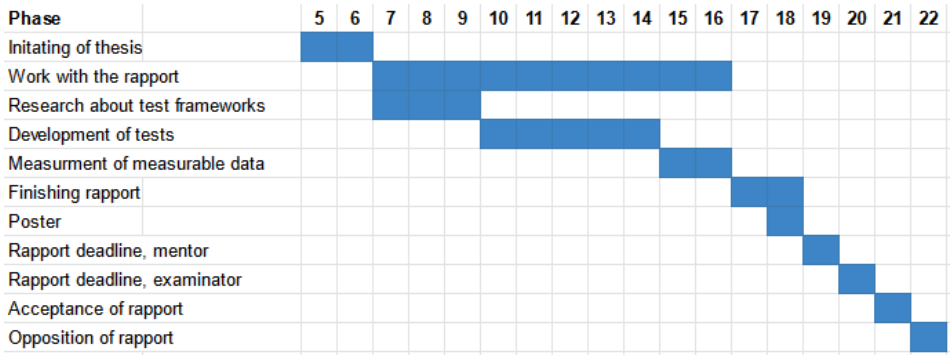

Figure 4.2: The GANTT schedule made in the early phases of the thesis

4.2 Data gathering

The first phase consisted primarily of gathering information from literature, websites, meetings and interviews. This was done in order for the thesis workers to establish a clear picture about what needed to be done in order for them to succeed with the goals of the thesis.

All meetings were documented, either by making notes or recording the meeting, for future reference. The meetings were all carried out on Microsoft Teams.

4.2.1 Understanding the application

The first few meetings being held on-line, taking place in the first and second month of the thesis, were semi-structured meetings primarily lead by the team at IKEA. All meetings contained the two thesis workers and their mentor at IKEA. Other employees with specific expertise were brought in as needed, depending on the topic of the meeting. The meetings had as a goal to introduce the thesis workers to the team and also the application, how it was structured and built.

The application is built using ReactJS, NodeJS and PostgreSQL. It consists of four different repositories hosted on GitHub. The four different repositories, and their responsibility, are as follows.

- nodejs-backend - the backend of the application handling all logic.
- react-front-end - the frontend of the UPPS part of the application.
- availability-co-worker-tool - the frontend of the co-worker part of the application.
- availability-management-database - database handler.

The application consists of thousands of lines of codes.

4.2.2 Defining the thesis scope

IKEA determined the thesis scope and workflow for the entire project, see figure 4.3. Due to their knowledge of the application and their ability to identify the critical flows, IKEA with the thesis workers decided on which test flow to be tested. The test flow was a critical part of the application and when testing it the thesis workers followed the workflow in figure 4.3. Also the time constraint of the thesis work was taken into consideration when deciding upon said workflow and test flow.

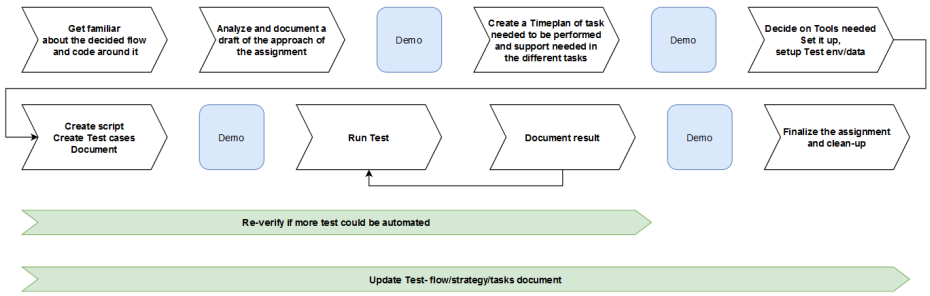

Figure 4.3: The workflow which was agreed upon between IKEA and the thesis workers

The team at IKEA and the thesis workers also came to the conclusion that thesis workers should work in parallel with the team at IKEA, as to not have any dependencies between each other. This was to reduce eventual unnecessary deadlines and delays that might come to be.

4.2.3 Test needs

When an understanding of the application was established, another round of meetings were conducted to gather information about how the application was tested at the moment. The data gathered from the meetings made it apparent that while the MIA team might have a test strategy, they did not at the moment have an

automated test flow. The only tests that were done were unit tests, and these did not cover the entirety of the code base. Therefore, an automated flow which tested multiple parts of the application, a so called integration test, needed to be constructed.

For the thesis workers, this meant researching best practices in automating test flows for web applications. Libraries and frameworks needed to be researched and evaluated, to get the best possible result.

4.2.4 Meeting with Senior Developers

Once the thesis workers felt like they had an understanding of how the application was constructed and how they should perform their tests, they held a meeting with two of the Senior Developers from the People Domain team. The plan was to have one single meeting but since both of the developers could not attend it was split up in two different meetings where one of the developers attended each meeting. The focus of these meetings was to verify that the thesis workers had done the correct analysis of the application and also to get information about if their choices were viable or not.

The thesis workers had questions they wanted answered in this meeting. These were as follows.

1. How important is the support of other programming languages for the test framework?
2. How important is the support of other web browsers than Chrome for the tests?
3. How is the automated tests to be deployed?
4. How should a test be verified?

The developers shall be called developer A and B respectively.

Answer to questions

1. How important is the support of other programming languages for the test framework?
   Developer A: Not important at all, as all other code for MIA is written in JavaScript or related languages.
   Developer B: Not important, same answer as developer A.
2. How important is the support of other web browsers than Chrome for the tests?
   Developer A: Developer A stated that support for Chrome was the most important, as the majority will use Android phones to connect to the application.
   Developer B: Safari support is to be preferred, as most users will use an iPhone to connect to the application.

3. How is the automated tests to be deployed?
   Developer A: Did not know the answer to the question.
   Developer B: Due to how the application is built, with a rather complex infrastructure, developer B did not find it believable that an entirely automated test flow could be constructed in time.

4. How should a test be verified?
   Developer A: No clear answer, suggestion to query the database.
   Developer B: The tests should be verified by checking the front end. For instance, if a time slot has been accepted by the manager, it should appear in the UI.

4.2.5 Meeting with Test Enabler team

A meeting with the Test Enabler team was conducted, and the topic was that of how to handle the log in of the application. The preferred solution, according to the team, was to have a way to turn off authentication for the application entirely. This was due to the fact that the log in function was preferably not to be tested, since it is Microsoft's implementation.

4.3 Data analysis

With the second phase, data gathered from the first phase needed to be analyzed. For the thesis workers to analyze the data, setting up MIA on their local machines was needed. This made it possible for them to conduct local tests and have an environment where they could see how the application worked. Afterwards, the thesis workers made a draft of the test flow which was approved by the MIA developers.

4.3.1 Flow to be tested

In order for the thesis workers to verify the understanding of the test flow, a flowchart was created. The flowchart was presented to developers and explained by the thesis workers to make sure that they had understood the applications flows and was then revised to correct any errors. See figure 4.4 which shows the flow through the application from a code standpoint. From a users perspective this flow could be explained as a co-worker who inserts a timeslot when said co-worker is available for work. This is the first part of the flow, after the timeslot has been created the flow goes into the Unit People Planning Specialist(UPPS) part of the application. Here a notification is shown to the UPPS and then accepts the timeslot. The last part of the flow is again a part of the co-workers perspective. Here a notification is shown of the accepted timeslot and then shows up in the co-workers calendar.

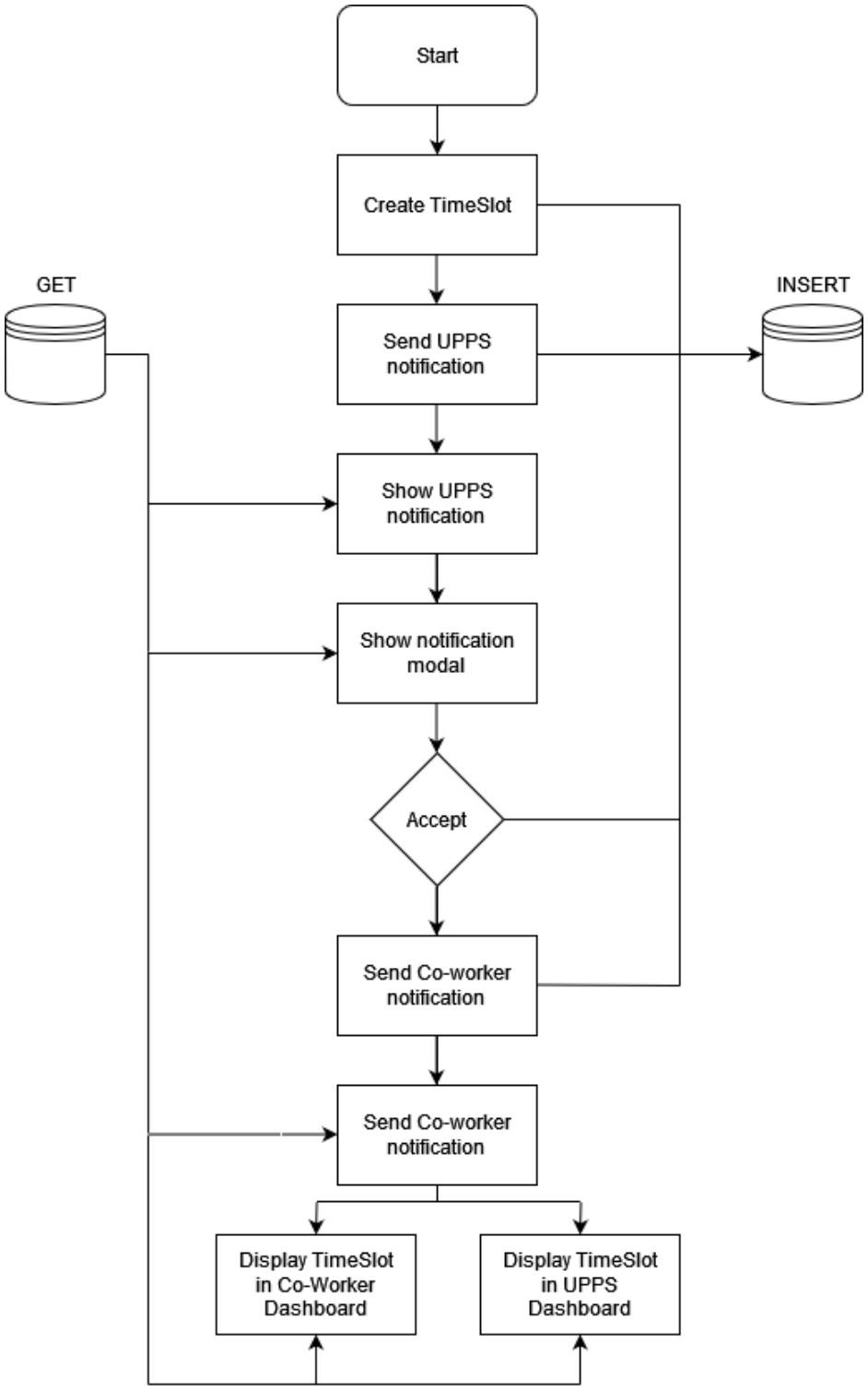

Figure 4.4: Flowchart over the application functionallity which is to be tested

4.3.2 Senior Developer interview analysis

The data gathered from the two meetings was not conclusive, and the thesis workers had to evaluate the information.

Meeting with developer A

The thesis workers got access to the database to be able to connect and query it to make sure data is stored correctly. As for the importance of support for different web browsers, it should be able to support native browsers of operating systems. Microsoft Edge, Safari and Chrome, this is because the co-workers will access the application, through their mobile phone. Therefore the thesis workers could not be sure of which web browser is being used. Even though the thesis workers will test on Chrome it would be wise to take into consideration for future work that multiple browsers should be supported. Since the co-workers will most likely access the application through their mobile phone, the viewport should be set to reflect these characteristics. For the programming language, it was decided that it was not that important, because the testing could be done in any language. As long as it fills the testing capabilities that was needed.

Meeting with developer B

The topics of this meeting focused more on the testing than on the technical questions. Since the stack is built in JavaScript it was confirmed that the choice of programming language of the test framework should be compatible with JavaScript but does not need to support other languages.

As most of the co-workers probably will access the application by Safari, it was decided to focus on testing for Safari and then other browsers depending on popularity. Also new information for the thesis workers was presented, that they did not need to query the database to ensure that data is stored correctly. They only need to make sure that the correct buttons and text etc. is shown to the co-worker and manager.

The testing should be built locally before implementing it in any pipelines or automation. By doing this developers could still use the automated test manually when writing code to ensure that the function and quality still remains.

4.3.3 Choosing frameworks and libraries

When choosing frameworks and libraries for the tests a wide variety of option was evaluated. These were as follows.

- **Browser automation tools:** Puppeteer and Selenium
- **Test frameworks:** Mocha and Chai
- **All in one framework:** Cypress

The thesis workers concluded that the best choice was Cypress, and the decision is discussed in the Analysis chapter 5.2.

4.4 Development

Once sufficient data was gathered regarding how to conduct test and how the application worked, phase three and thus development of the tests started.

4.4.1 Setting up Cypress

Firstly, as Cypress is a Node.js framework, a Node.js repository needs to be setup. This is done with the npm tool which is included in Node.js. A folder with the name of mia-testflow was created, and then chosen as the working directory.

```
1 mkdir mia-testflow
2 cd mia-testflow
3 npm init
```

The terminal will prompt for several questions, such as package name, version and author. After the package is created, the Cypress framework needs to be installed. This is done using npm and the following command.

```
1 npm install cypress
```

This will install Cypress in the working directory. Thereafter, a command can be ran to open Cypress.

```
1 npx cypress open
```

When this is done, a folder called "cypress" is created, with the following folder structure.

```
cypress
├── fixtures
├── integration
├── plugins
└── support
```

The fixtures folder contains test data that can be used throughout the tests. Integration holds the test files, plugins is to extend or change the commands of Cypress and support is used in order to write reusable code. At first, Cypress fills these folders automatically with files to showcase the frameworks abilities. These folders were cleared, and only the integration and support folders were used for the thesis.

4.4.2 Handling the log in

One of the major problems that was faced had to do with the application log in. As mentioned before, the application uses Microsofts Single Sign On, which means that all log in attempts calls a Microsoft made API in order to authenticate. This was an issue due to the fact that Cypress do not support multiple domains, meaning if you have started the tests on localhost:3000 for instance, you can not access Microsofts domain for the log in.

There were attempts to circumvent this by using a POST request to the Microsoft API directly from Cypress, and thus allowing an access token to be created and saved in the browser storage. This did not work due to the MIA application requesting the token in a different format than what is returned from Microsoft.

In conjunction with a senior developer within the MIA team, a by-pass was created in the MIA application with help functions in Cypress. This work around only worked with Microsoft Edge, but it was concluded that it was more important for the thesis workers to get the tests written than to support the originally planned web browsers. Thus, all further tests were written to support the Microsoft Edge browser.

4.4.3 Test development

A total of three different flows were developed. They were split up due to the fact that they were dependent to each other, meaning that the first test needs to be ran in order for the second test to work, and so on. The tests are listed in the following table, and were conducted in the order that they are listed in. Since the application itself is split in two parts, one for co-workers (regular workers at IKEA, working the floor) and UPPS (managers that create the schedule for the co-workers), the different test scripts needed to be split with this in mind. To create the tests Cypress functions of getting certain elements and in some cases click or just view them were used. These functions is used to simulate how a user could use the application, in this case the flow described before.

- Co-worker: add timeslot
- UPPS: accept timeslot
- Co-worker: verify timeslot

Co-worker: add timeslot

All workers at IKEAs warehouses are referenced as co-workers in this thesis. As the functionality to be tested was the adding and accepting of timeslots, a suitable start was deemed to be for a co-worker adding a timeslot through the application. This flow was developed in a simple way, and in order for the UPPS part to be able to identify and accept the correct application once ran, it was decided that the co-worker application would always be made the 15th of the next month, between 06:00-07.00.

By having time and date known for the application, it makes it possible for the continuation of the test to identify the correct timeslot. It is necessary that the correct timeslot is identified and accepted since that is part of the validation criteria of the test.

UPPS: accept timeslot

UPPS is the person who is using the data that is collect through the application, they use it to make the schedule for warehouses. The tests for UPPS was to make sure an availability was made by a co-worker and then approved. After it was approved the availability was supposed to be viewed as a timeslot on the correct date of the availability. Since no calls to the database or APIs was done, the tests needed to make sure it was passed correctly by checking the UI for different elements depending on which action that was done.

First of all the thesis workers made sure the account logged in had the correct role of UPPS. This was done by opening the profile page and checking if there was a text containing UPPS beneath the username. By finding this the thesis workers could make sure the correct role was used. A notification tab was used to show new availabilities, to access this tab the class name of the element for the tab was used to click the notification bell to show the modal with notifications. Then to accept the right notification a few selectors was being used to always choose the one made by the co-worker profile used when testing, this to not accept ones that are not currently active for the test being ran.

When an availability is accepted it should be shown as a time slot on the correct date. To navigate to correct date a calendar was being used, accessing this with a certain selector and then choosing the correct date. To make it simple and not so brittle, the first of each month was always chosen. When clicking on the first of the month that the co-worker had put its availability it should be shown as a time slot, with the correct co-worker profile attached. If it was shown, the test had passed and log out was done.

Co-worker: verify timeslot

To validate the test on the co-worker part as well, it was decided to create a third flow to once again enter the co-worker application and check that the timeslot was entered in the calendar. Lastly, once all tests were validated, the timeslot was removed to make the tests re-runnable.

4.5 Evaluation of results

The fourth and last phase begun after development was finished, and was conducted to gather and evaluate the result of the thesis work.

4.5.1 Showcase for People Domain Developers

In order for the thesis workers to get some feedback, and for the People Domain team to get the results gathered from the thesis, a showcase was conducted. In this presentation, the thesis workers talked about the thesis as a whole and discussed problems met during the development of the tests. There was also a demo held for the tests in which the entire flow was shown.

To end the meeting, a discussion within the People Domain team was held and the theme of the discussion was on how to improve testing in the future and what part of the testing that could be automated.

4.6 Communication

This section describes how the communication worked during the thesis, both between the thesis workers and also between the thesis workers and IKEA.

4.6.1 Between the thesis workers

The main form of communication between the thesis workers were Microsoft Teams, that is when the thesis workers did not meet face to face. As much work as possible were conducted on site either at IKEA Sockerbruket or at Campus Helsingborg. The thesis workers felt it was a lot easier meeting in person to achieve their goals for the thesis.

4.6.2 Between the thesis workers and IKEA

Since the outbreak of Covid-19 a lot of companies including IKEA shifted to work remotely. At the start of the thesis almost all communication was by email. When the thesis workers got their credentials for IKEA the communication switched to Microsoft Teams. Most of the weekly and daily meetings between the thesis workers and IKEA were conducted remotely. Every Wednesday, the People Planning work group had possibilities to work on site at IKEA Sockerbruket to help each other in person.

IKEA suggested having "semi-daily" stand-ups with the thesis workers on every Monday and Thursday in order for the thesis workers to inform their mentor of what they were working with, if they had any questions or anything that blocks them from continuing their work or similar problems. These stand-ups were conducted over Microsoft Teams.

The People Domain team also has a meeting room booked at Sockerbruket every Wednesday in order for the team to meet and collaborate. The thesis workers attended these "workshops" a couple of times. It was noted though that there were a lot of meetings taken place during this time. The thesis workers made a decision that they would attend these workshops when they had time, otherwise they would work in a separate room.

4.7 Source criticism

The following section describes the sources used in the thesis, and an evaluation on their trustworthiness.

4.7.1 Frameworks and libraries

The vast majority of the sources cited in the thesis is in the category of frameworks and libraries. Reason being that a lot of unknown technology was used, and thus the thesis workers needed to read up on how to use them.

As there are many frameworks and libraries competing for users, one need to take precaution when reading their websites. Of course they are there to assist developers using the frameworks, but they are also there to try to create interest for new users to download their framework or library and to use it in their project.

Since this thesis has evaluated multiple different frameworks and libraries independently, of which everything is discussed in the thesis, the main use for the citations has been their documentation. That is, most of the citations used in this category has been used to gain a technical understanding of the aforementioned frameworks and libraries. The sources can thus be seen as trustworthy in the way that they are referred to in this thesis.

The following citations fall under the category of frameworks and libraries:
[1] [3] [5] [6] [7] [13] [17] [18] [19] [10] [12] [8] [9] [15] [11]

4.7.2 Tools

To develop the test suite, a number of tools have been used. The sources cited from these tools are not used in a way that can inflict factual errors in the thesis, rather they are cited to leave a reference on where to download the said tool.

The following citations fall under the category of tools:
[2] [21]

4.7.3 Technical Documentation

Another set of information that was gathered was technical documentation from software providers. They can be seen as trustworthy as the documentation is only explaining how to use their software, and are not to inflict any factual errors.

The following citations fall under the category of technical documentation:
[16] [14]

4.7.4 Literature

Some literature was studied to gain an understanding of certain concepts discussed in the thesis. The literature was mostly used to refer to an other unexplained sub-

ject, that the literature explains:

The following citations fall under the category of literature. [23]

Chapter 5
Analysis

This chapter analyses the choices made in the thesis, and explains why one thing was chosen over another.

5.1 Choice of programming language and server environment

As the MIA application was developed mainly in JavaScript with a NodeJS backend, it felt like a natural choice for the thesis work to continue with the JavaScript language. This was done because the result of the thesis was hopefully to be used by IKEA in their future work with the application. As the People Domain uses this specific technical stack it would be a strange choice not to keep the same stack in the thesis.

5.2 Choice of frameworks and libraries

When choosing between frameworks for automating web browser behaviour, the thesis workers had a number of prerequisites that needed to be full filled. Firstly, the framework needed to be written in JavaScript due to the request of IKEA and the fact that most of the MIA application is written in JavaScript. Having different languages responsible for running and testing the application often might not be the best solution. Second, it had to be based on the NodeJS framework, for the same reason as why it needed to be a JavaScript framework. Third, due to the time constraint with the thesis, it needed to be relatively easy to learn. Thesis workers had some previous experience with Puppeteer, which made it seem like a good alternative for the thesis to use. All the frameworks that were investigated, that is Puppeteer, Selenium and Cypress fulfilled these requirements.

When evaluating the alternatives, at first it seemed like an obvious choice for the thesis to use Selenium as the web automation tool and Mocha and Chai as the test and assertion libraries. This is due to that the thesis workers first evaluated the frameworks based on metrics that later became irrelevant after speaking with the senior developers of MIA. These metrics were things such as support for other programming languages and support for other web browsers. After investigation though, a new framework was found that had both web automation and test/assertion libraries built into it, Cypress. When trying it out, the thesis work-

ers found that it was much easier and faster to write the tests in Cypress as it was built specifically for testing web applications. Remember, that while Puppeteer and Selenium is a tool that allows to build test for web applications, they are not made with that specifically in mind. They are web automation libraries.

5.3 Single Sign On

As the MIA application uses Single Sign On from Microsoft Azure, this login procedure was to be automated. This proved to be a more difficult task than first imagined. As Cypress does not support cross domain in their testing, Cypress did not allow the user to be redirected to Microsoft's login portal, which showed to be a problem. An idea of how to circumvent the login is by doing a post request with necessary information to the Microsoft login portal. A response was received with an access token that the thesis workers thought would be enough. As the thesis workers tried this it was discovered that it was not enough, so the thesis workers sat down with a Senior Developer from the team. As the thesis workers discussed the solution both parties saw that the structure of the access token was different from the response that were received and what is being used in the application. Because the thesis workers knowledge of how the application is written was limited, the thesis workers did not have a clear understanding of what needed to be done. The developer did some further research and discovered that it was not possible to automate the login flow correctly.

To move on with the work, the thesis workers somehow needed to get past the login, to do this one of the developers helped with the work. With the help the thesis workers could get past the login with one downside, it is not using the testing accounts for the application. It uses the INGKA account which is logged in locally on the computer that is being used for the tests.

5.4 Automation of the test flow

When the work with the thesis was first started, both the thesis workers and IKEA had in mind an automatic test flow that would run for example every night or every time someone pushed something to the MIA repository. As time went on, the thesis workers, in combination with developers on the MIA team, came to the conclusion that this would not be possible to achieve in the time frame of the thesis work. This was mainly due to the application being rather complex, with four different repositories responsible for different parts of the application.

5.5 GANTT

The following GANTT schedule is the actual schedule that came to be, previously discussed in section 4.1.1.

The fact that the GANTT schedule was made before the work actually started, meant that it was a draft and estimation on how much time was needed on each part of the thesis. Because of the issues with SSO, the development took longer time than expected since this problem became more central than first believed.

The initial part of the thesis where frameworks were investigated, took longer than expected. This was because there were a lot more frameworks than the thesis workers initially expected to work with. This part took two weeks longer, since more comparisons and discussion was needed to pick what the thesis workers thought was the best choice for this particular work.

Lastly, the finalizing of the thesis and making of the poster were postponed due to the fact that the development itself took two weeks longer.

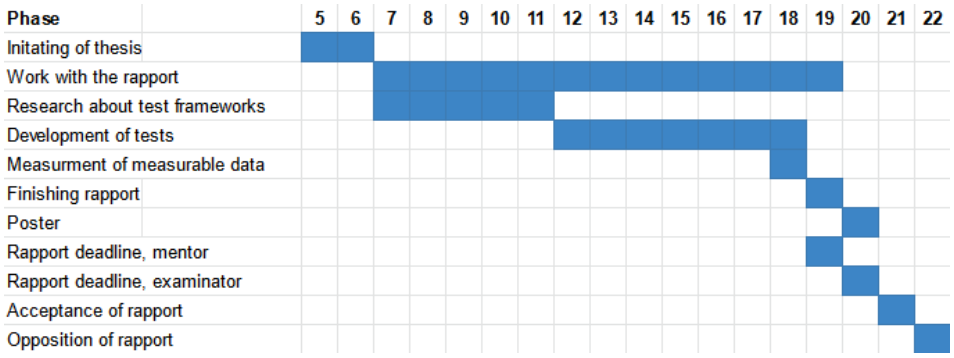

Figure 5.1: The actual time frame that the thesis held

Chapter 6
Results

The results given to IKEA are the test suite, consisting of three test scripts, previously described in the thesis, and proposals on how to make the application more testable. The tests were delivered to the team at IKEA through GitHub. The proposals were discussed during the presentation held at IKEA, and mainly consisted of fixing a test profile, a test profile in this case would be to make it possible to turn off the authorisation for the application. Or fixing how the application authenticates users logging in.

6.1 Showcase to IKEA team

As to get feedback for the developed tests, the tests were presented to the team responsible for the My IKEA Availability application. This was done through a presentation held by the thesis workers, with the team consisting of Software Engineers, Engineering Managers and Scrum Masters present. A demo was held to show what had been developed also the code itself was shown and shared, this made it possible for the team at IKEA to get a better understanding of the test suite and which problems existed.

Short snippet of code from the actual result

```
1  describe("coworker application", () => {
2      it('successfully loads', () => {
3          cy.visit('http://localhost:3004/');
4      })
5
6      it('loads timetable', () => {
7          cy.get('.heading-text');
8          cy.url().should('include', '/availability');
9      })
10
11     it('opens the timeslot modal', () => {
12         cy.get('span').contains('Create a timeslot').click
               ();
13         cy.get('.timeSlot-modal-close').should('be.visible
               ');
```

```
14      })
```

This is the first thirteen lines of code for the first of three scripts. This is specifically made for the co-worker application.

6.1.1 Presentation

To be able to get feedback and start discussions about the work that had been done in the thesis. A presentation was held, to give as much details as possible for the team at IKEA. The presentation informed about problems that came up while developing and also how they were solved if possible or suggestions on how to solve it. These problems revolved mainly about the Single Sign On.

6.1.2 Feedback from developers

After the presentation the developers of MIA gave feedback on the work. Overall they were pleased with the result, although the result was not as pictured from the beginning. Because of the problems with Single Sign On, the thesis work could not be finished as planned. Instead a test suite which is semi-automatic was developed to come as close as possible to the main goal. The developers understood what the problems had done in the way of halting the development and were very pleased with the result.

6.1.3 Discussion

As a result of the presentation and thesis work, a discussion followed. This discussion was about the work needed to make it possible to fully automate tests and also future work and suggestions. One of the most important topics was the Single Sign On, since this is the biggest problem of the thesis work. If this problem is solved one of the main problems that arose from it, could be solved and that is to make it possible to use a single computer for the tests.

6.2 Test suite

The final result of the thesis work became a semi-automatic test suite, meaning that three scripts were made that ran on both the co-worker and UPPS application. Because of the problems with SSO it was not possible to make it fully automatic as first intended. The scripts ran on two different computers and had to be started manually for it to run. Since the solution for handling SSO made it so that it was not possible to switch users during a test run it had to be done this way with different accounts on different computers. This made it so it were still not possible to run all scripts on one single computer at certain times or when developers made changes to the code.

As the result did not become what was initially stated, the thesis workers had to settle with a semi-automatic test suite. Because of the problems during the

thesis work while developing, this decision was made to not stand still while developing and being able to produce a result. This test suite was structured with three different parts, two for the co-worker and one for the UPPS. It had to be done in this way since tests could not be done on one single computer due to the Single Sign On problem. Therefore the test suite itself is not fully automated, but the tests running are automated.

The test suite is developed using Cypress. When the program is ran, one can choose to either run a specific test or to run all tests, see figure 6.1.

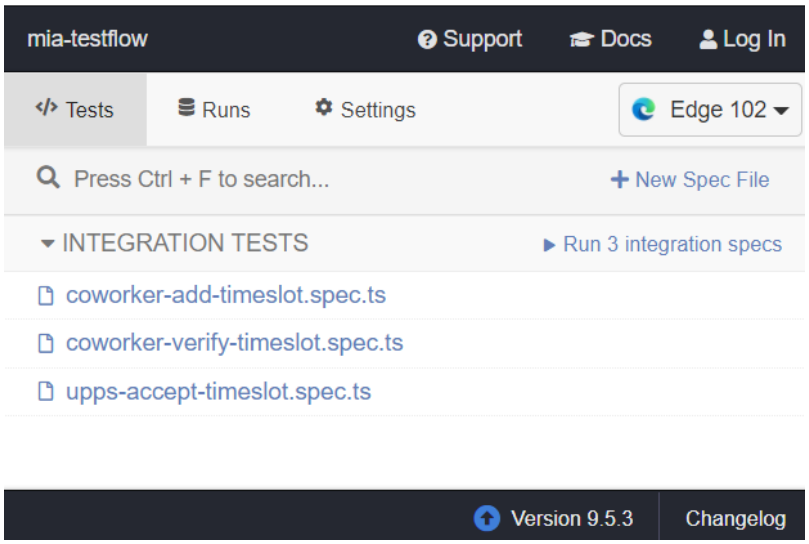

Figure 6.1: Screenshot of the window that pops up once Cypress is ran

By clicking on "Run 3 integration specs", all three tests in the suite are ran. One can also run a specific test by clicking on the name of it.

When a test is run, a new window pops up, see figure 6.2

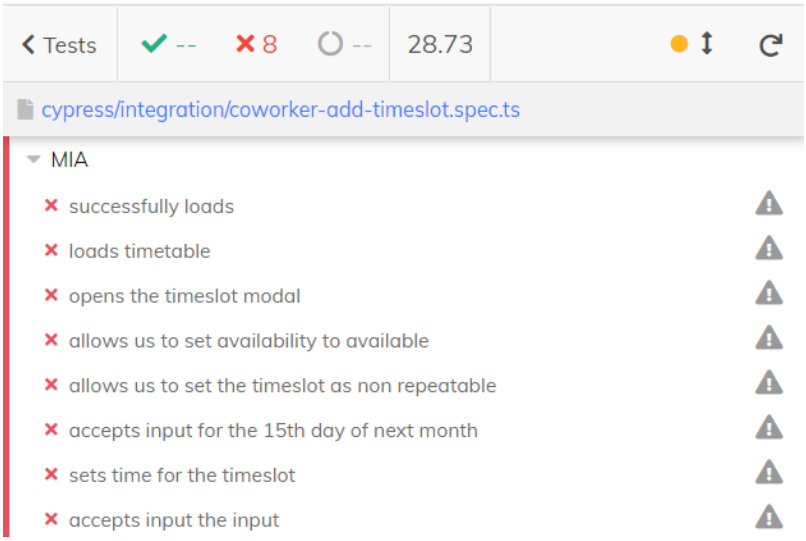

Figure 6.2: Screenshot of the window that pops up once a test is running

This window tells a number of things. First of all, it describes all the tests. Each of the test are a line in the table shown. Second, it shows whether a test has passed a or not. A passing test is denoted by a green check mark, and a failing test is denoted by a red cross. In the case of figure 6.2, all tests have failed.

Cypress also gives the user a number of tools which can be used to diagnose an application, such as snapshots of the application when the tests are ran.

Chapter 7
Conclusion

This chapter is a summary of the most important results, and also the answers to the specific problems that were introduced in the problem specification, stated in section 2.4.

The purpose of the thesis, as discussed in chapter 2.2, was to examine the possibility to implement an automatic test flow for the MIA application, examine and evaluate different test frameworks and to write tests to be shown in a proof of concept. Throughout the previous chapters, all these thoughts have been answered by literature study and a working test suite.

The goal of the thesis was to implement an automated test flow for the MIA application. This goal was partially achieved, as the test flow could not be automated entirely due to how the application was built.

7.1 Answers to the problem specification

The following section answers the problem specification, introduced in the chapter Introduction.

7.1.1 Problem 1 - How is the MIA application tested at the moment?

There were no tests, with the exception of some unit tests, conducted when the thesis began. This made it a necessity to investigate alternatives thoroughly and to be able to motivate the choices made as they could become the foundation of which the People Domain team at IKEA continues with the thesis.

7.1.2 Problem 2 - In which ways can the tests be automated?

The fully automated test flow, which was the goal from the beginning were not able to be achieved due to problems that were discovered while developing. Therefore an answer to this problem was unfortunately not found during the thesis work. The automated part is three different test flows, that is ran manually.

7.1.3 Problem 3 - Which framework is best suited for running tests on MIA?

This was discovered to not have a single best answer, due to different functionalities and appearances of frameworks. There are many different frameworks, and the one used for the thesis was chosen because it had characteristics that the thesis workers found best suited for the task at hand. Even though it was thoroughly investigated, this does not mean that the chosen framework is the best for MIA.

7.1.4 Problem 4 - How do you automate browser activity?

There are different frameworks to use to automate browser activity. As the thesis work is done during a shorter amount of time, all available options could not be considered. Therefore a few frameworks were looked in to, these would show to have the same functionality but written in different ways. Also some of them did not have built in testing while automating browser activity. Automated browser activity can be automated by using frameworks that supports this functionality, in this thesis Selenium, Puppeteer and Cypress were investigated. All of the above can automate browser activity.

7.2 Ethical evaluation

Since the MIA application uses actual user data, one needs to carefully consider the impact of the actions the application might have, especially in the EU with GDPR relevant. The way that the tests are written at the moment, it uses the person running the tests credentials and thus name and so on.

A data cleanup would be performed, as mentioned in future work, nothing would be stored and thus no person data would be available, the impact would be minimal. But since the data is not cleaned, an actual entry is stored in the database.

The database used is the one made specifically for the development environment of MIA. The only people that have access to the database are those working with the MIA application. There is no other user data available in this database, in contrast to the database used in the live application, and thus the impact is minimal in regards of secrecy of personal data.

7.3 Future work

This section contains areas in which the test code could be improved on, and also a few features that we originally planned to implement but did not have the time to.

7.3.1 Support for other browsers

As the application can be used for multiple browsers, support for testing other browsers than the ones chosen in the thesis could be an improvement to be made. Specifically, the thesis workers were not able to test Safari even though it is, according to the team at IKEA, the most used browser for the co-worker application. This was due to the fact that it would need to be tested on a computer with MacOS to support Safari.

7.3.2 Database cleanup

When a test is performed, entries into the database are made. These entries will stay persistent, even though the test is done. When further developing the tests, a cleanup script for the database that is ran once the tests are done could be created. This would mean that the test script did not have any data stored from it, which is preferable.

7.3.3 Viewports

The application supports different viewports, and the view is different on mobile and desktop. The written tests are currently only tested using a desktop viewport. This should not have any impact on the testing of the application, but could be a future development area in order to further validate the application.

7.3.4 SSO solution

Currently the MIA application is hard locked behind Microsoft's SSO, meaning that the thesis workers can not in any way turn it off. In the future, an option when starting the MIA environment to shut off the SSO and instead allow users to access the application without any authorization would be preferable. By doing this, not only would it improve the reliability of the tests, but also it would mean that the tests actually better represent what they are written for.

7.3.5 HTML attributes

When selecting different elements on a page, one can use a wide variety of methods. A lot of times, the MIA applications elements can be selected using specific id or name attributes on the elements. In cases where these are not unique, one needs to select in a more complex way. In the future, the MIA application could be written in such a way that gives every element an identifiable ID making the tests easier to write and less brittle for changes.

7.3.6 Automation

As was discovered a bit into the thesis, the work that would be required for the flow to be entirely automated was to big. The cause of this was that the MIA application is a rather complicated application, needing a series of authorization

to setup and access. In the future, the option for making the application containerizable and thus the tests automatable would incur a great impact on the testing.

Chapter 8
Terminology

Mocha: A JavaScript test framework for Node.js programs.
Selenium: A library with tools to automate web browser UI tests.
Workload: How many employees needs to be working certain hours in different areas.
Unit People Planning Specialist (UPPS): A manager who can approve or decline a co-workers availability application. A manager can also manage the workload.
Co-worker: A user of the MIA application, on a floor worker level at warehouses.
My IKEA Availability (MIA): The application which IKEA has developed, that the thesis examines.

Bibliography

[1] Node.js. *About*. Feb. 2022. URL: https://nodejs.org/en/about/.

[2] npm. *npm Documentation*. Mar. 2022. URL: https://docs.npmjs.com/about-npm/.

[3] *React – a JavaScript library for building user interfaces*. Feb. 2022. URL: https://reactjs.org/.

[4] Mozilla. *Introduction to the DOM*. Mar. 2022. URL: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction.

[5] PostgreSQL Global Development Group. Feb. 2022. URL: https://www.postgresql.org/.

[6] Mocha. *Mocha homepage*. Mar. 2022. URL: https://mochajs.org/.

[7] Mocha. *Mocha Interface*. Mar. 2022. URL: https://www.w3resource.com/mocha/interfaces.php.

[8] Mocha. *Chai*. Mar. 2022. URL: https://www.chaijs.com/.

[9] Mocha. *Chai Interface*. Mar. 2022. URL: https://www.chaijs.com/guide/styles/.

[10] Selenium. *Selenium documentation*. Mar. 2022. URL: https://www.selenium.dev/documentation/.

[11] Selenium. *Selenium WebDrivers*. Mar. 2022. URL: https://www.selenium.dev/documentation/webdriver/getting_started/install_drivers/.

[12] Selenium. *Selenium Locators*. Mar. 2022. URL: https://www.selenium.dev/documentation/webdriver/elements/locators/.

[13] Puppeteer. *Puppeteer*. Mar. 2022. URL: https://pptr.dev/.

[14] Google. *Chrome DevTools*. Mar. 2022. URL: https://chromedevtools.github.io/devtools-protocol/.

[15] Puppeteer. *Puppeteer Selectors*. Mar. 2022. URL: https://pptr.dev/#?product=Puppeteer&version=v13.5.1&show=api-interface-selector.

[16] Mozilla. *CSS Selectors*. Mar. 2022. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors.

[17] Cypress. *Cypress*. Apr. 2022. URL: https://www.cypress.io/.

[18] Cypress. *Cypress - Features*. Apr. 2022. URL: https://www.cypress.io/features.

[19] Cypress. *Cypress - Documentation*. Apr. 2022. URL: https://docs.cypress.io/guides/getting-started/writing-your-first-test#Write-your-first-test.

[20] Svensk e-identitet. *Vad är SSO?* May 2022. URL: https://e-identitet.se/news/vad-ar-sso/.

[21] Microsoft. *Visual Studio Code*. Mar. 2022. URL: https://code.visualstudio.com/.

[22] INGKA Technology radar. *INGKA Technology radar*. Mar. 2022. URL: https://techradar.ingka.com/.

[23] Craig Larman. "Iterative Development". In: *Agile and Iterative Development: A manager's guide*. Addison-Wesley, 2004, p. 9.