The process of changing out expandable elements in a large-scale web application

Armin Vehabovic

Master's thesis in Software Engineering

Supervisor: Adj. Prof. Dragos Truscan

Faculty of Science and Engineering

Department of Information Technologies

Åbo Akademi University

Autumn 2020

Acknowledgements

I would like to express my gratitude to Elio Nushi who with his honest and helpful comments have made this thesis amazingly fun to write. A thank you to Dragos Truscan who kept me inspired throughout the writing of this thesis. A big thank you to the team of My Pages at IF who keep doing an amazing job. Thank you, thank you all, I hope you enjoy reading this as much as I did while writing it.

Abstract

The Nordic insurance company IF is taking on a new company-wide image which will define its new visual identity. Given the large amount of work, the *My Pages team* which is responsible for the Nordic self-service portal in IF, is doing the work in smaller iterations. This will facilitate the transition to the new visual identity. While the overall work has already started, a major part of the project remains to be changed so that it adapts to the new visual identity. The work done in this thesis is a description of the process that I as part of the My Pages team followed to successfully achieve that goal.

An expandable element is a visual element that when collapsed has the most important information visible and can be clearly distinguished to be clickable. After clicking the expandable element, the content is pushed down, and a hidden area below the expandable element is toggled which contains more descriptive information. We describe here the work done to implement the new expandable elements from IF's own visual identity library in My Pages. First, the incorporation of the expandable element from the Visual Identity Library is made. Then the implementation of the custom My Pages changes to the expandable elements is done. The custom functionality puts structure to more information in the expandable element and makes it more readable. Furthermore, the tests which fail after the implementation of the expandable elements are fixed and those that randomly fail are improved so that they run successfully throughout the runs.

My Pages web application is built upon multiple programming languages and libraries, such as HTML, SCSS, JavaScript, and jQuery, which are integrated into the ASP.NET web framework. The implementation of the new expandable elements required in-depth modification of the whole project and its components. The team works with an Agile methodology and the documenting of the work progress is done in Azure DevOps and it is followed up in daily team meetings. To successfully fix the feature tests we use the following technologies and libraries: Specflow (which is a Behavior Driven Development test framework that does the coupling of feature tests files with scenarios for each individual step), Selenium WebDriver (which does the automation of tests execution in the browser), Chromedriver (that is a bridge between the actions

of the feature tests and the Chrome browser) and TeamCity which is a continuous integration server that hosts the execution of the feature tests and creates the CI Builds. Finally, the deployment of the solution to production use is done with Octopus setup (which deploys packages with scripts containing ARM templates to different environments).

keywords: Web Application, ASP.NET, HTML, Bootstrap, SCSS, BEM, JavaScript, jQuery, Agile, Software Testing, Specflow, Selenium WebDriver, Chromedriver, TeamCity, Octopus.

Table of contents

Glossary

DOM	Document Object Model, is an object model for HTML
IE11	Internet Explorer 11, a web browser by Microsoft
IIS	Internet Information Services, server software by Microsoft
SLOC	Source lines of code, a software metric to measure the lines of code

1. Introduction

In the background, the software development process of complex systems requires careful elaboration and hard work from the development team. These details are often undermined and difficult to be estimated from the side of the users of the system. From the customer's perspective, the software development process is hidden and only the released product is visible. Creating a responsive web application - that is, an application which adapts to best display the web page on different devices (i.e. mobile, tablet, desktop), is not an easy task. There is no way to guarantee a completely flawless user experience. As such, software development is a process that improves their functionality through reiterating (i.e. questioning current ways and evaluating new ideas based on the experience of the users) and developing a better solution for each iteration. The user experience must be taken into consideration by testing both internally (i.e. development team) and externally (i.e. end-users) the ideas. This is an iterative process where the application is developed, tested, and redeveloped until the best experience for the customer is achieved.

Recent developments in web technologies and libraries allow for multiple methodologies of changing the layout of a web page. In CSS, the flexbox is a very good example which illustrates the power of new components. Flexbox makes it possible to style containers and blocks which can be ordered and aligned according to very strict requirements. These methodologies sometimes cause confusion even among experienced developers and doing major visual changes to the system requires time and practice. The layout needs to be interpreted the same way in all available browsers and their versions. To make the layout look the same across the browsers there is a need for browser-specific CSS overrides. Furthermore, in many solutions, there is a high percentage of legacy code, which works as intended but is prone to errors which could arise due to incompatibilities. To ease this process there are many tools, libraries, frameworks that can be used (ReSharper, ASP.NET).

The work consists of understanding the requirements for the expandable element and the implementation of the expandable component from the Visual Identity Library. The feature tests for the expandable component are improved to consistently succeed throughout runs.

1.1. Expandable element

The expandable is an HTML element which is styled to look like a rectangular box. When this element is collapsed, it shows only the most important information and can be clearly distinguished to be clickable. Clicking the expandable element causes the content after it to be pushed down. Then a hidden area below the expandable element is toggled and within that more descriptive information about the product is shown. The information inside an expandable element contains texts, matrices, actions (i.e. other clickable components). The area inside the expandable element is highlighted to show the whole information about it. [1]

The expandable elements which the work focuses on have already been a part of My Pages. This includes the data shown in the expandable elements, the custom functionality, and the feature tests.

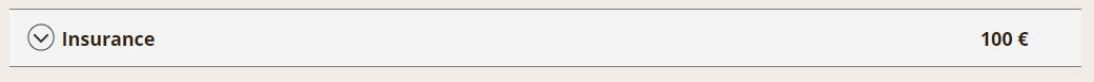

Figure 1. Example of an old closed expandable on a desktop device

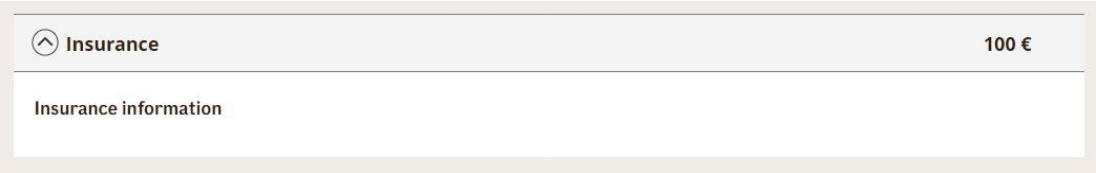

Figure 2. Example of an old open expandable on a desktop device

An example of the old expandable element and what it looked like when viewed on a desktop device can be seen in Figure 1 and Figure 2. The first figure illustrates the old expandable element in a closed state and the second figure illustrates an opened old expandable element. None of the figures illustrate the animation type of styling on the expandable element. The *hover over* mouse effect makes the color of the expandable header darker, and it highlights the expandable element. When viewed on a tablet device the expandable element behaves almost like on a desktop device and, for this reason, we consider it unnecessary to illustrate.

Figure 3. Example of an old closed expandable on a mobile device

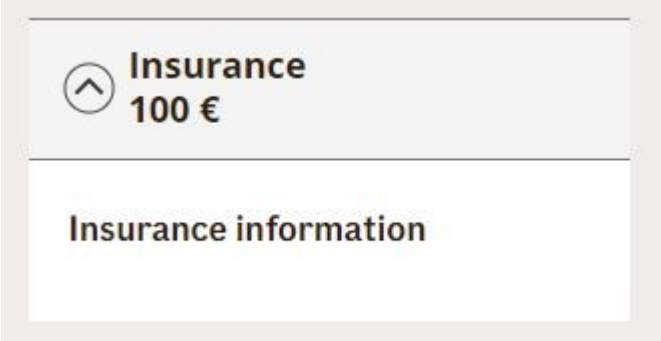

Figure 4. Example of an old open expandable on a mobile device

Figure 3 and Figure 4 illustrate the old expandable element when viewed on a mobile device. Figure 3 is in a closed state and Figure 4 is in an opened state. Even though the space for displaying information in a mobile device is limited considering the narrow screen, the content is still easy to read and appealing from the perspective of the customer.

The difference between desktop, tablet, and mobile devices and how it influences the expandable elements can be read in chapter 2.1.7 where we discuss Bootstrap and responsive design.

1.2. Visual Identity Library

The Visual Identity Library is IF's own style library which is being used by all teams making any web application intended for internal or external use. The Visual Identity team in IF is responsible for making new components and maintaining the style library. It removes the need for individual teams, using the library, to correct any issues with components. Thus, the style library maintains a higher quality throughout its lifecycle.

The Visual Identity Library provides the look of the expandable element, specifications, code samples and JavaScript pseudo-code. The provided assets from the Visual Identity Team are meant to help any team in IF working with any of the components from the Visual Identity Library.

The Visual Identity Library is heavily inspired by Bootstrap and the basic concepts are the same. The responsive design, grid system, and the way components are made are all based on the Bootstrap library. More about Bootstrap can be read in chapter 2.1.7.

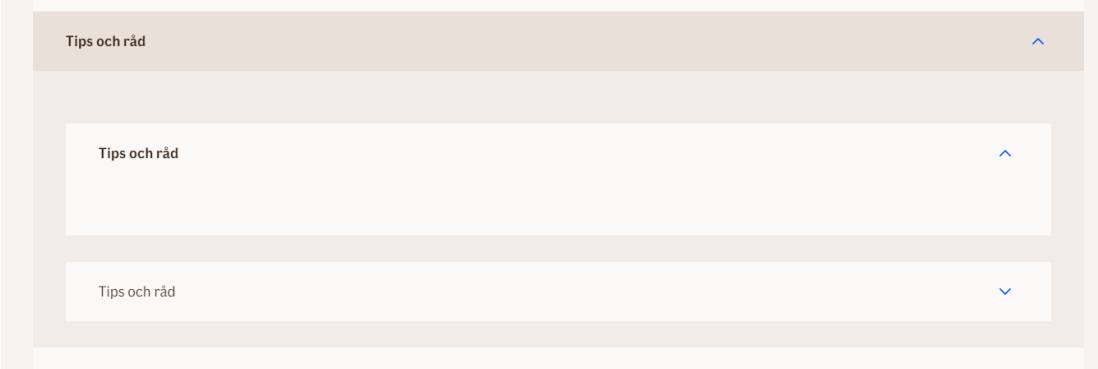

Figure 5. Example of an expandable element provided by Visual Identity Library

Figure 5 illustrates an example from the Visual Identity Library of what the expandable element can look like. The two nested expandable elements are of a different color so they can be distinguished from the other content inside the expandable element.

```
<div class="if expandable is-open">
  <div class="if title" aria-expanded="true" aria-controls="exp1" tabindex="0">Tips och råd</div>
  <div class="if content" role="region" id="exp1" tabindex="0">
    <button class="if button primary" type="button">A button</button>
  </div>
</div>

<div class="if expandable">
  <div class="if title" aria-expanded="false" aria-controls="exp2" tabindex="0">Tips och råd</div>
  <div class="if content" role="region" id="exp2" tabindex="-1">
    <button class="if button primary" type="button">A button</button>
  </div>
</div>
```

Figure 6. Example of the HTML code for the expandable element provided by the Visual Identity Library

Figure 6 shows the code example that the Visual Identity Library provides for the expandable element. The necessary CSS classes which make up the expandable element can be seen, as well as other tags which make the accessibility possible and make the expandable elements possible to tab through (i.e. using the tab key on the keyboard to focus the different elements on the web page). The code acts as a starting block for the teams who will use the expandable element.

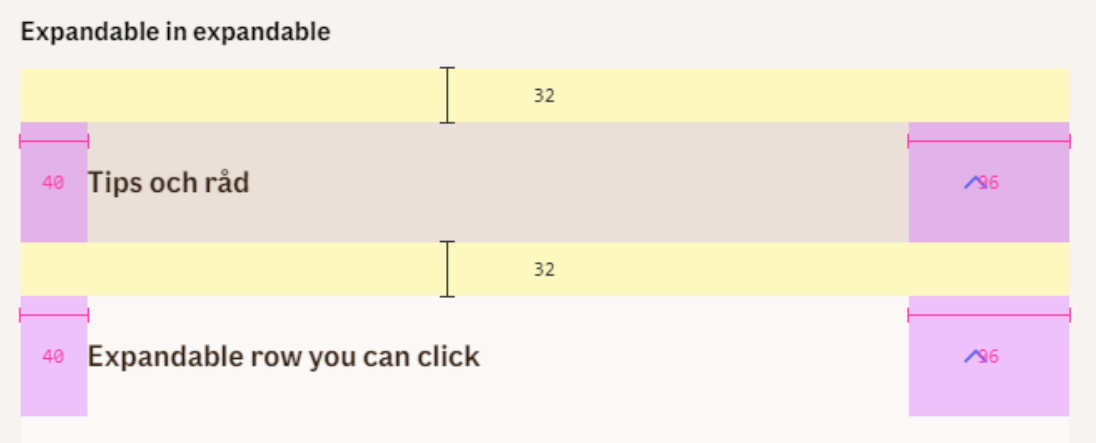

Figure 7. Example of the expandable element specifications provided by the Visual Identity Library

Figure 7 illustrates the specifications for the expandable element provided by the Visual Identity Library. The specifications are defined in the library provided by the Visual Identity Team and as such do not need any work.

1.3. Goal of the thesis

The main goal of the thesis is to implement the new expandable component from the Visual Identity Library, thus replacing the old expandable component with the new one in all the places it's being used. The goal is to follow the company-wide goal of IF to implement the new Visual Identity in all applications visible to the customers of the organization. The task is to learn the process of the My Pages team, from the starting of a new work item, to the development, the testing and finally the production deployment. The work consists of improving and correcting any correlated errors that testers find with the expandable elements. It also consists of improving any error found in the Visual Identity Library, the errors related to custom My Pages changes made to the expandable element are to be corrected. Other expandable element errors are to be reported to the Visual Identity team. Finally, the feature tests which are broken by replacing all the expandable elements that need to be fixed and improved, so that the feature tests consistently succeed.

1.4. Thesis structure

I start by introducing the technologies, frameworks, and ways of working to be able to work with the team successfully and to implement the new expandable elements. Then I continue with the requirements of the thesis and how they play a role in the work. Later, I present the results of the thesis, how everything was done, the initial issues, their solutions, and other improvements for the task. I also discuss how the evaluation process of the work was done. Finally, I discuss the lessons learned during the whole development of the tasks and further improvement that could have been made or can be made in the future.

2. Background

In this chapter the different technologies, frameworks, and ways of working are introduced and elaborated.

2.1. Web technologies

2.1.1. Hyper Text Markup Language

HTML is a markup language which describes the composition of the web page. The markup consists of elements which are defined by tags and these tags come in pairs, the opening tag, and the closing tag. The elements in the HTML document describe the content which the browser renders accordingly. The content is labeled by the HTML elements (e.g. “this is a heading” describes the H1 tag). The most basic HTML document has the *html*, *head*, and *body* tags. [2]

```
<html>
    <head>
        <title>Example</title>
    </head>
    <body>
        <h1>H1 header tag</h1>
        <p>Paragraph tag</p>
    </body>
</html>
```

Figure 8. Example of HTML document with text

Figure 8 illustrates a HTML document with page title set to example, a H1 tag and a P tag with its defined texts.

2.1.2. Cascading Style Sheets

CSS describes how the HTML elements should be rendered in the web browser [3]. The web browsers interpret the CSS in their own unique way, requiring browser specific CSS overrides. The CSS selector selects the element by the id, the class name or the HTML element and applies the styling. The declaration block consists of one or more declarations which are separated by semicolons. Every declaration has a CSS property name and a value which are separated by semicolon. The CSS properties can e.g. change the size of the font by using the property: font-size:18px, the color of the background: background-color:blue or aligning the text by using: text-align:center.

```
h1 {
    color: red;
    font-size: 22px;
}

p {
    background-color: red;
}
```

Figure 9. Example of CSS styling of the h1 and p tags

Figure 9 illustrates how the H1 and P tags in Figure 8 can be styled. The H1 tag is colored red and its font is set to 22 pixels. The background color of the P tag is set to red.

2.1.3. Syntactically Awesome Style Sheets

SASS is a preprocessor and can extend the functionality of CSS. It reduces the repetitive definition of common properties such as colors, fonts, and spacing. The library compiles the preprocessed SASS file into regular CSS which is responsible for styling the web page. This makes the SASS compatible with all the CSS versions and browsers. [4] Defining common variables to reuse across the implementation, enables easier manipulation of common elements.

```
$red: #ff0000;
$light-red: #ff3232;

h1 {
    color: $red;
    font-size: 22px;
}

p {
    background-color: $light-red;
}
```

Figure 10. Example of using SASS for styling the H1 and P tags

Figure 10 illustrates the usage of variables with SASS, the colors are defined in the beginning of the file and used to color the H1 tag and the P tag.

2.1.4. Block Element Modifier

When dealing with large-scale solutions, styling the web pages without any CSS class naming structure brings unnecessary complexity. Assigning a unique describing class name escalates in difficulty as the number of components and blocks grow in the solution. And the CSS classes are inherited down the DOM structure and the more specific CSS selector overrides all the others. “Specificity is a weight that is applied to a given CSS declaration, determined by the number of each selector type in the matching selector” [5]. There are methodologies to make CSS writing, organizing and maintaining clearer and less complicated, and one such methodology is BEM which stands for *Block Element Modifier*. Each of the three terms that BEM describes has a clear meaning and definition: *Block* is a standalone element and is usable on its own. *Element* is semantically tied to the Block, an item inside of the block, an *Element* has no meaning on its own. *Modifier,* when applied on the element it changes the appearance or behavior [6]. The web page is made by blocks which all have their own elements and modifiers defined. The blocks are reusable, and the CSS is easier to handle throughout the solution. Thus, BEM puts structure to the CSS and makes it readable throughout the solution.

HTML

```
<button class="button">
        Normal button
</button>
<button class="button button--state-success">
        Success button
</button>
<button class="button button--state-danger">
        Danger button
</button>
```

CSS

```
.button {
        display: inline-block;
        border-radius: 3px;
        padding: 7px 12px;
        border: 1px solid #D5D5D5;
        background-image: linear-gradient(#EEE, #DDD);
        font: 700 13px/18px Helvetica, arial;
}
.button--state-success {
        color: #FFF;
        background: #569E3D linear-gradient(#79D858, #569E3D) repeat-
x;
        border-color: #4A993E;
}
.button--state-danger {
        color: #900;
}
```

Figure 11. Example of the BEM methodology [6]

Figure 11 illustrates the utilization of the BEM methodology to style the buttons by using a block and modifiers. The button is the block, a standalone element which is modular. The Success button and the Danger button use the button block, but modifier is also applied to change the buttons´ appearance.

2.1.5. JavaScript

JavaScript is a lightweight and object-oriented scripting language with exceptional functions. It fully integrates with the HTML and CSS to make interactive web pages. [7] JavaScript runs in the browser and manipulates the DOM to successfully accomplish actions. It is supported by all major web browsers by default. [8] It can listen to events such as clicks, mouse hovering, and execute actions when they are triggered. The JavaScript events and such are case sensitive and use camelCase for naming the variables.

```
<p id="text1" onclick="myFunc()">Click me.</p>

<script>
    function myFunc() {
      document.getElementById("text1").style.color = "red";
    }
</script>
```

Figure 12. Example of the onclick functionality in JavaScript

Figure 12 illustrates the click functionality in the JavaScript library. When the P tag is clicked the JavaScript will find the P tag and change its color to red.

2.1.6. jQuery

jQuery is a rapid, lightweight, and feature-rich library which simplifies usage of the functionality available in JavaScript. [9] The jQuery library streamlines the usage of JavaScript by wrapping the functionality of JavaScript in methods. This enables the functionality in jQuery to be executed by a single line of code. The jQuery library supports cross-browser functionality which significantly facilitates the usage of the functionality. It can do more with less code, the many lines of complicated JavaScript code are simplified with jQuery. According to the jQuery documentation, jQuery library offers functionality such as HTML/DOM manipulation, CSS manipulation, HTML event methods, Effects and animations, AJAX, and utilities. [10]

```
<p id="text1">Click me.</p>

<script>
    $("#text1").click(function() {
      $(this).css('color', 'red');
    });
</script>
```

Figure 13. Example of the click functionality in the jQuery library

Figure 13 illustrates the click functionality in the jQuery library. The jQuery finds the P tag with the id text1 and listens for a click event. On clicking it will trigger the functionality and change the color of the element to red.

2.1.7. Bootstrap

Bootstrap is a CSS library which was created and released in 2011 by two engineers at Twitter. The motivation for it was the need of the company to have a unified design library for their systems. The different teams at Twitter used their own set of styles and designs, which caused issues within the company. As a result, different applications within the company had different design languages. Since its early adoption as open source, the Bootstrap library has gained momentum throughout the years and world-wide approval. Today, Bootstrap is the most popular HTML, CSS and JavaScript library in the world. [11]

It is easy to start working with the Bootstrap library by watching one of the many guides available (e.g. on YouTube, Bootstrap documentation). By using the containers and classes of the Bootstrap library, the user can build a customizable and appealing front-end. Additionally, there are many icons and themes which can be used as groundwork for styling the design of the web page.

2.1.7.1. Responsive design

Responsive design of a web page refers to adapting the layout to fit the user's device (e.g. mobile device, tablet device or desktop device). The web page is shown on the device the way it is defined programmatically. The Bootstrap library defines CSS classes which style the containers, blocks, and elements for the different devices. The work of optimizing the web page for different devices is important to keep the web page consistent throughout the devices. The web page is used by many kinds of devices daily, and the web page needs to be usable on all the devices. The information and different actions available on a desktop device need to be just as usable on a mobile device or tablet device.

```
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
```

Figure 14. Bootstrap example of the responsive design meta tag [12]

To make the web page responsive the code in Figure 14 is required in the HTML document under the Head tag. The meta tag ensures proper rendering and touch zooming for all the devices. [12]

The Bootstrap library defines 5 distinct breakpoints, which are used to distinguish different devices, such as mobile device, tablet device, desktop device, and larger desktop device. The breakpoints differentiate the devices by their relative width. The web page is optimized for the different devices with the CSS classes provided by the Bootstrap library.

	Extra small <576px	**Small** ≥576px	**Medium** ≥768px	**Large** ≥992px	**Extra large** ≥1200px
`.container`	100%	540px	720px	960px	1140px

Figure 15. Bootstrap example of the 5 defined breakpoints in the library [13]

Figure 15 illustrates the specifications for the 5 breakpoints defined in the Bootstrap library. The container for the Extra small device, when the width of the device is smaller than 576 pixels, is full width. The Small breakpoint, which is valid above 576 pixels and below 768 pixels, has a 540-pixel container. The Medium breakpoint, which is valid above 768 pixels and below 992 pixels, has a 720-pixel container. The Extra small, the Small, and the Medium breakpoints are used for the mobile device, tablet device and desktop device. The breakpoints above the Medium breakpoint are used for wider screen desktops.

The container is the most basic layout element in the Bootstrap library for the construction of a web page. It is necessary to use the container as the outer element with the grid system. The grid system inside the container is adjusted to the width of the container. The container on a web page is defined just once per web page but can be nested when needed. [13] The container can use margins, paddings, and the content inside the container can be center aligned to optimize the view to necessary requirements.

The layout is optimized for different devices in the Bootstrap library by using media queries. The media queries use the CSS property Media to specify the width at which the breakpoint is valid. With the usage of the Media queries, it is possible to manipulate the layout to optimize it for different devices.

```
// Small devices (landscape phones, 576px and up)
@media (min-width: 576px) { ... }

// Medium devices (tablets, 768px and up)
@media (min-width: 768px) { ... }

// Large devices (desktops, 992px and up)
@media (min-width: 992px) { ... }
```

Figure 16. Bootstrap example of the media queries. [14]

Figure 16 illustrates the media queries used for tablet, normal and wider desktop devices. The absence of the media query for the mobile device is because the Bootstrap library is developed for a mobile first approach.

Furthermore, to add structure to the data displayed inside a container the Bootstrap uses a grid system. The grid system is building blocks made with flexbox in CSS which is used to make the web page. By using the grid system, it is possible to build a web page which will adapt the content to the used device.

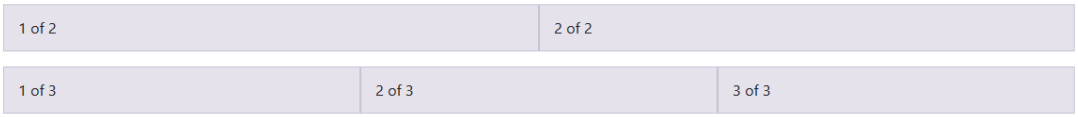

Figure 17. Bootstrap example of the grid system. [13]

Figure 17 illustrates the grid system on the desktop device, the blocks are all equally sized. By using the Bootstrap grid CSS classes, different grid systems can be made to fit the needs of the requirements.

The grid system slices the available width of the web page container in 12 columns, the blocks are set to use a specified number of the columns. It is built by specifying the number of columns set for each of the devices (e.g. in a mobile the number of columns can be 12 but, in a tablet, and above it can be set to 6). The grid system will make the layout adhere to the available space and, thus, the layout is adjusted to different devices. It is set to 12 columns because it is divisible by lower numbers.

2.1.7.2. Expandable element

By using the expandable elements, much of the information displayed on the web page can be made more compact. The expandable elements make the web page cleaner, more readable and easier to use. Clicking on an expandable element, reveals more details about the insurance or invoice.

The Bootstrap library calls the expandable element *an accordion*. The name expandable element comes from the Visual Identity Library. The two terms have the same meaning, but the term expandable element is used throughout the thesis. The expandable element was used for the work of implementing the user story and, thus, it is used for the thesis.

Figure 18. Bootstrap example of a closed expandable element [1]

In Figure 18, we can see an example of a closed expandable element from the Bootstrap library. The expandable element has only one informative data point and it is highlighted and clickable by the user.

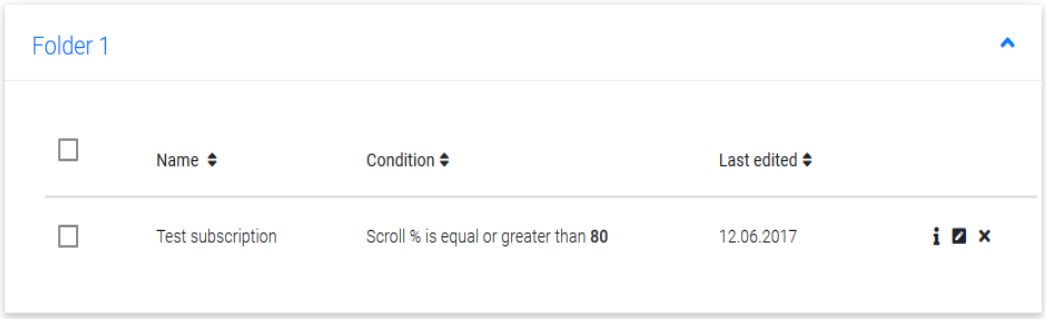

Figure 19. Bootstrap example of an opened expandable element [1]

Figure 19 shows an expandable element from the Bootstrap library which is expanded. The data inside can be anything; in the example it is a table with one row of information. The area around the expandable element is highlighted to make clear the expanded area related to the expandable element.

2.2. CI/CD pipeline at IF

2.2.1. Azure DevOps

Azure DevOps is a cloud application owned by Microsoft that offers a variety of services to host, develop and ease the process of building and maintaining software projects. It provides several tools, such as Azure Boards, Pipelines, Repos, Test Plans, and Artifacts, which help the development and the organization of the work within the team. [15] Currently, My Pages uses only Azure Boards and Azure Repos but, in the future, it will also use the other available services.

Azure Boards allows software teams to manage the software projects. All the work is tracked and viewed in a comprehensible way via a virtual board, where tasks can be created and assigned to the people who will be responsible for their completion. Azure Boards makes it easy for teams to collaborate and organize the work. The board can be customized to fit the needs of the team. It can be used in teams who work with different kinds of software development processes such as Scrum, Kanban etc. Azure DevOps is tightly coupled with Agile work methodology and it complements the Scrum framework. [16]

The My Pages team uses the Scrum framework to deliver high quality items.

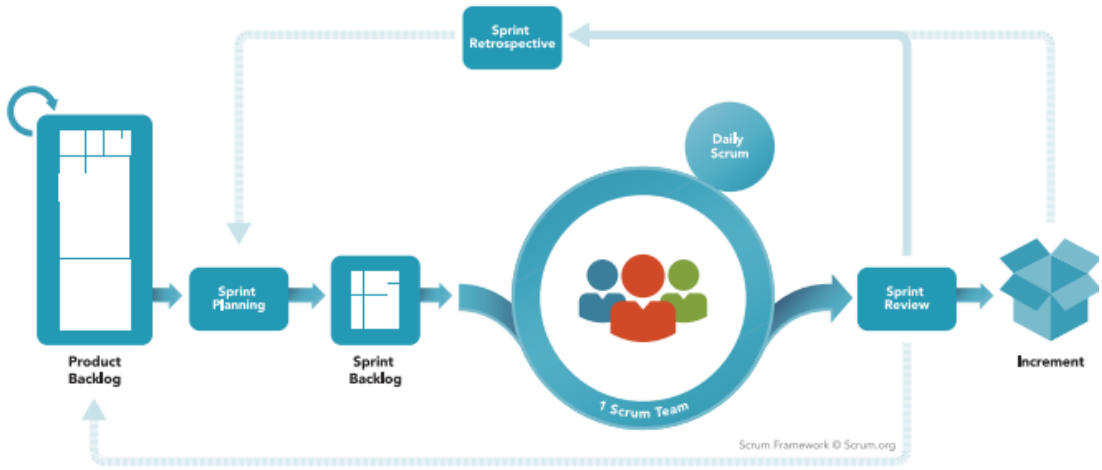

Figure 20. Example of the Scrum Framework [17]

Figure 20 illustrates the workflow of the Scrum framework. The scrum team works in periods which are usually 2-3 weeks long and they are called sprints. *Daily scrum* meetings take place to go through the sprint board and to help mitigate difficulties faced during the work. New work items are placed in the *sprint backlog* and prioritized by the product owner. The product owner represents the stakeholders such as customers and business management. The scrum master is the leader of the team and helps everybody understand the scrum theory, practices, and rules. Before the start of every sprint, a *sprint planning* meeting is organized to plan the work which will be done during the sprint. At the end of each sprint, the *sprint review* meeting takes place to introduce the new functionalities to interested parties. The developed functionalities are deployed to the production environment one week after the ending of the sprint. The *sprint retrospective* gives an opportunity to reflect on the sprint and to improve the process around the Scrum framework. [17]

The work in the Azure Boards is structured in the following way: for each task that is to be done a new work item is created in the virtual board. There are different work items corresponding to the type of task that must be done. The different types of work items that can be created for My Pages are epics, user stories, technical stories, bugs, and acceptance bugs.

An *epic* is the large picture of the work and the epic consists of user stories which split the work into smaller manageable pieces, but towards the same goal. A *user story* defines the requirements for the feature that is to be implemented. The progress of the user stories is tracked by tasks which provide detailed information about the progress of the user story. A *technical user story* is a work item which defines non-functional requirements to make it more consistent, manageable and easier to maintain. *Bugs* are errors in the production environment where the functionality is not behaving according to the requirements. *Acceptance bugs* are problems with the functionality which are identified in the test environment during the acceptance testing. Issues like the previous stated can be separately deployed to the production environment and this procedure is referred to as a hotfix. This is done by creating a deliverable package of the current production code and the fixed code, the package is then deployed to the production environment.

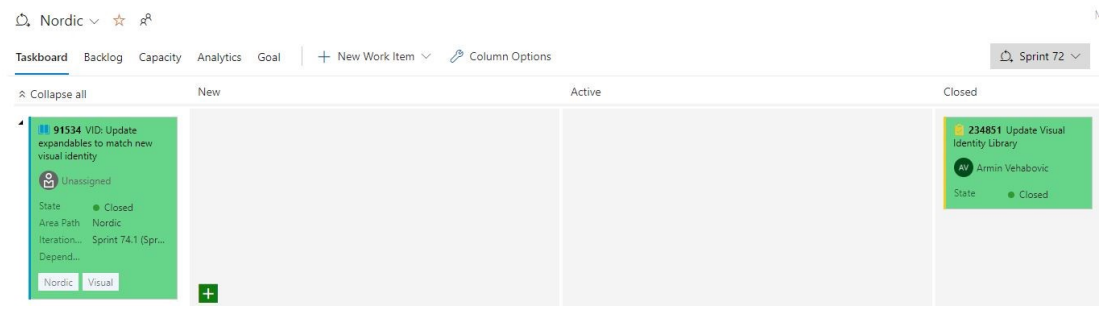

Figure 21. Overview of the Sprint Board with the expandable element user story visible

Figure 21 illustrates the sprint board for sprint 72, and the user story for the expandable elements can be seen in the board. The task to update the Visual Identity Library is marked as done by being placed in the Closed column. Tasks are put into new, active, or closed columns to indicate their progress.

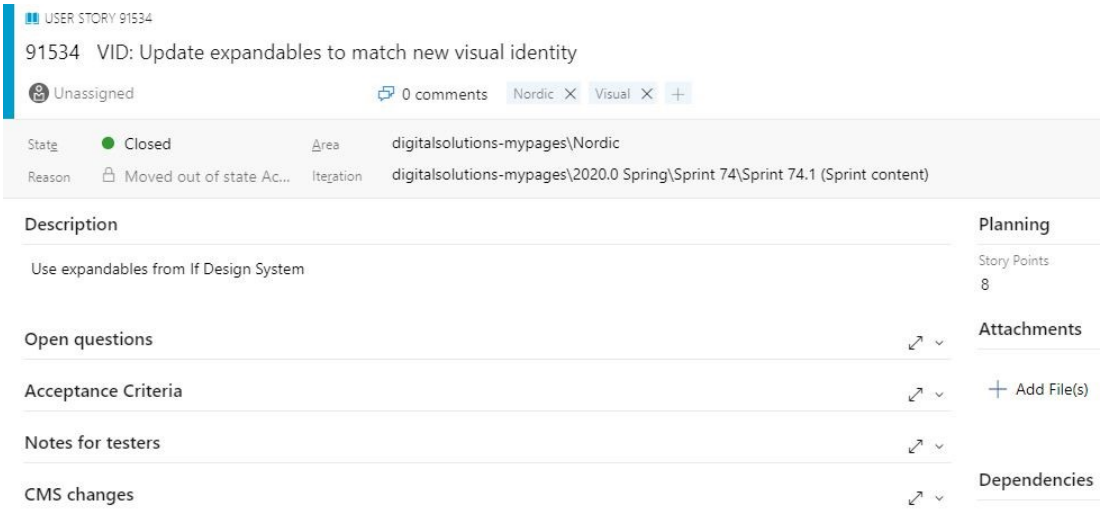

Figure 22. The Overview of the User Story for implementing the new expandable elements

Figure 22 illustrates the user story for the expandable elements. The requirements of the story are defined, and any other information related to the story is also shown.

2.2.2. Software Testing

Testing the software is very important to keep its quality high by finding the errors before they reach any environment. The software testing can be done by automated tests, by software testers or by both. The software testers complement the automated tests, and they do not do each other's job in most cases. For every new functionality added to the solution, it is very crucial to add tests which assert the logic of the implementation. The new code can have an impact on older functionality in unforeseen ways, and unintentional bugs can be introduced. The tests are meant to act as a sanity check to safeguard the requirements. Any broken logic will make the test fail which alerts that something is amiss. The purpose of automated tests is to catch issues that arise during development of new functionality. The software testers cannot test all the functionality that is already implemented. They are only responsible to test the new functionality, which is the reason why the automated tests complement the software testers, by testing all the previous functionalities every time a new functionality is introduced.

There are different kinds of tests which test many kinds of functionalities. The feature tests are responsible for initializing proper test data, imitating the requests of different services, and primarily testing the layout. The unit tests are responsible for imitating the services and testing that any logic in the solution to those services works as intended. The integration tests are responsible for testing the connection between services and the solution.

The automated acceptance tests are made to continuously test the same requirements on the solution. When the requirements change, the tests need to be altered to reflect the changes. The automated tests are made to make sure that the features of the already developed story are working even after the new code is committed. The manual testing is done by the software tester which also tests the requirements of the user story. Small changes, such as the new source code breaking the already existing functionality or older functionality not working as intended, can also be detected through manual testing. The software tester goes through all the requirements and makes sure that they are all satisfied. They also try to break the functionality in unpredictable ways (e.g. by causing issues as they go back and forth, clicking the actions in a random order, trying to input potentially harmful data etc.). The final goal is to make sure that the

functionality should not be compromised by any unconventional input given by the user.

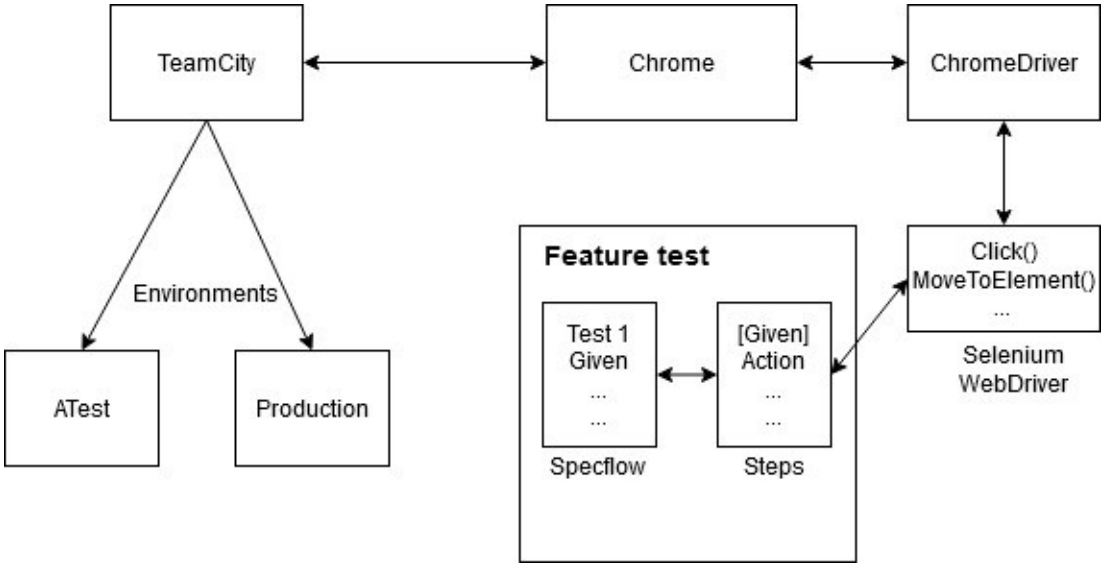

Figure 23. Overview of the My Pages testing flow

Figure 23 illustrates the overview of the My Pages testing flow. The feature test is written in the Specflow file and interacts with the Selenium WebDriver. Chromedriver acts as the bridge between the action and the Chrome browser for successful execution. TeamCity continuously runs the tests and deploys builds which have succeeded.

2.2.3. Test environment

A test environment is used for the purpose of testing. In the best case, it is a reflection of the production environment, but with synthetic data. The test data can also be just synthetic test data created specifically for the test case under testing. The test environment is used to successfully test the new functionality without any interruptions to production. It is separately hosted to avoid any possible mistakes to the production environment. There can be multiple environments which are used for different stages of testing the user stories. One test environment for sprint testing is the initial testing stage for a user story. After the sprint testing, the user story undergoes acceptance testing, which is done in another test environment.

It is crucial that the test environments have consistent test data and services just like the production data would have. During all stages of testing a newly developed user story, it is vital that the data is what it is expected to be in the production environment. If that is not the case, then it is possible for errors in production to occur due to the code not reflecting the production data.

The developers of the solution work on a version of the web application which runs on the local machine. The local environment is not connected with the test environments. It is meant as a way for the developer to continuously test the development of the new functionalities. A separate environment is necessary for the programmers to be able to work without interrupting any ongoing testing.

2.2.4. Specflow

Specflow is an open-source testing framework for .NET Framework Web applications. It is built upon the behavior driven development software process. [18] According to the documentation of Specflow, the *Behavior Driven Development (BDD) is an agile software development practice that encourages collaboration between everyone involved in developing software: developers, testers, and business representatives such as product owners or business analysts* [19]. In Specflow, there are two types of files: feature test files and step files. The feature test file is coupled with step files to provide the necessary functionality for the feature test. Feature test files use bindings to bind the steps with the functionality in the step files

The feature tests in Specflow are written using Gherkin which defines the feature test scenarios in a natural language. Gherkin gives structure and meaning to the feature test scenarios. Gherkin uses a special set of keywords to define the steps, which are the Given step, the When step and the Then step. [20] The Given step initializes the test data needed for the solution to display the web page. The When step defines the actions done in the browser by the scenario. The Then step defines the expected outcome of the scenario. Gherkin's style for writing scenarios is effective, easy to read and easy to use for the developer [20].

```
Feature: SpecFlowFeature1
    In order to avoid silly mistakes
    As a math idiot
    I want to be told the sum of two numbers

Scenario: Add two numbers
    Given the first number is 50
    And the second number is 70
    When the two numbers are added
    Then the result should be 120
```

Figure 24. Example of a feature test scenario made in Specflow

Figure 24 illustrates a Feature test definition where the name of the feature and its description are defined first. Then a scenario is created to test the functionality by adding two numbers and checking whether the result is as specified in the Then step.

```
[Binding]
[Scope(Feature = "SpecFlowFeature1")]
public class SpecFlowFeature1Steps
{
    [Given(@"the first number is (.*)")]
    public void GivenTheFirstNumberIs(int p0)
    {
        // Action
    }

    [Given(@"the second number is (.*)")]
    public void GivenTheSecondNumberIs(int p0)
    {}

    [When(@"the two numbers are added")]
    public void WhenTheTwoNumbersAreAdded()
    {}

    [Then(@"the result should be (.*)")]
    public void ThenTheResultShouldBe(int p0)
    {}
}
```

Figure 25. Example of the code-behind file for the steps of a scenario

In Figure 25, the binding defined first is responsible for linking the Specflow file to it. It is linked by binding the same name as was defined in the feature test file. The steps

are responsible for initializing the test data, doing the necessary actions in the web browser, and asserting the layout.

2.2.5. Chromedriver

Chromedriver acts as the intermediate between the feature test actions and the Chrome web browser. It makes it possible for the feature tests to interact with Chrome. The Chromedriver enables actions such as user input, web page navigation and JavaScript execution. The Chromedriver is published for each release of Chrome by the Chromium team. [21] The versions of the Chrome browser installed on the machine and the Chromedriver should be compatible with each other to allow for the successful execution of the feature tests.

The feature tests can run in any other browser, but the Chrome web browser may be the biggest one in terms of usages [22]. To be able to use another browser, the driver for the browser needs to be downloaded and initialized in a similar manner as to Chromedriver. There are drivers for all the major browsers available online such as Firefox, Edge, Safari, and IE.

2.2.6. Selenium WebDriver

The Selenium WebDriver is the bridge between Chromedriver and the feature test framework Specflow. The WebDriver is responsible for executing the actions of the feature test scenarios in the browser. It is a driver which can do the necessary actions to fully utilize the feature test's automation. Selenium executes the actions in the browser as a user would do, it drives the browser natively. [23] The WebDriver simulates the user action by clicking the visible element on the web page. Another alternative is to use the JavaScript clicking functionality which finds the element in the DOM and clicks it. In the latter case, the web page may be rendered differently than the structure of the DOM. This is because of the styling of the elements and the custom JavaScript functionality which may alter the view of the web page.

2.2.7. TeamCity

TeamCity is a continuous integration server which continuously checks Git for changes in the branch. Each submission is followed by an automated build to guarantee that the changes made to the files are consistent with the existing code and the detection of possible problems is done early. [24] TeamCity makes a package of the committed code and pushes it to the TeamCity agents which continuously run the tests. The overview page of TeamCity shows a real-time status indicator of the running test. It also offers detailed information about the tests that its agent is running. The package made by TeamCity for running the tests is used to push the committed code to any environment only after the tests pass successfully.

The TeamCity agents must be configured to successfully run the tests. The installation file is available from the TeamCity Web UI and it configures the agents to communicate with the TeamCity Server [25]. The agent also needs to be properly configured and have the correct libraries installed for the solution under testing. Setting up a new TeamCity agent is as easy as creating another virtual machine. The image of the virtual machine in the Azure Cloud is then saved and used to initialize more agents. The number of agents is limited by the number of licenses purchased [25].

The tests on the TeamCity agents are parallelized and many groups of tests can run on multiple agents at the same time. The parallelization of the tests decreases the time needed to run all the tests. It is also important for successfully running the continuous integration builds. The results of the tests will be the same without regard to whether they run on the developer local machine or TeamCity agent. In the case of the local machine, the tests run in a sequential order which causes a considerable delay compared to running them on TeamCity agents. This is because TeamCity can employ multiple agents which work in parallel and significantly reduce the amount of time needed to run the tests, thus making it feasible for continuous delivery.

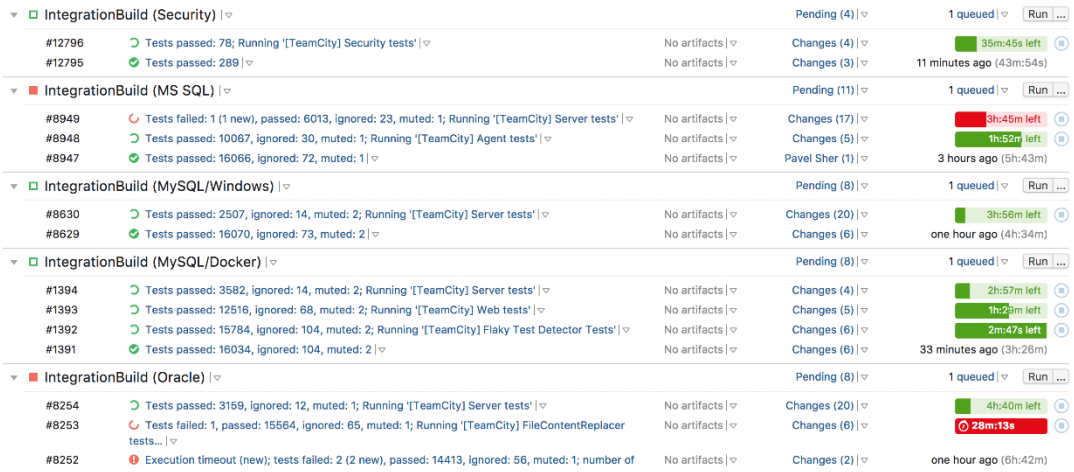

Figure 26. Example of the TeamCity overview page [26]

Figure 26 illustrates the overview page of TeamCity where the current running tests are shown. The tests are split into groups which run separately on different TeamCity agents. The continuous builds are run constantly on the agents, if there are any agents available. The tests which are successful are shown in green and are indicated by a successful message. The group of tests run sequentially; if any test fails in the group, then the whole group is marked as failed and a failed message is shown. Every time the tests are run, they are marked with a running build number which makes it possible to track the commit of the build.

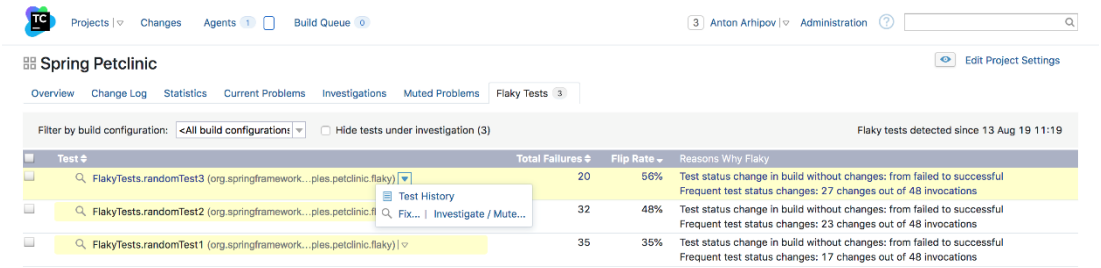

Figure 27. Example of the flaky test feature in TeamCity [27]

Figure 27 illustrates the Flaky Test functionality in TeamCity. A flaky test is a test that fails on a TeamCity agent without any obvious reason related to the code. We discuss more about flaky tests and how they were improved in chapter 4.1.

2.2.8. Git

Git is a version control system which is mostly used for software code. It assists the developers when collaborating on a project. The code is stored in a remote repository, that is a workspace which is stored online and is accessible to the team. The repository is cloned to the local machine and every time the code is edited, that change happens in the local repository and then it can be propagated to the remote repository via the push command. The developer can fetch the latest version from the remote repository by pulling the latest changes. Changes that are done to the code are committed to the repository and are assigned a unique id and stored in a tree data structure. The developer can write his own version of the code by creating a new branch which keeps the master version of the code separated from the newly introduced changes, thus, making it possible to revert the changes if it is necessary. [28] The master branch is the official version of the code which is later deployed and ultimately all the branches should be merged to the master branch. A branch can have many commits, and sometimes we might not need to merge all the changes of the branch to the master branch but only one of its commits. The merging of a single commit of a branch with the master branch is referred to as cherry-picking in the Git terminology.

When merging or cherry-picking from branches, merge conflicts can arise due to code incoherence. The merge conflict can be resolved with one of the many merge tools available online, such as Git Extensions, SourceTree and many more. To avoid this kind of problem it is often recommended to cherry-pick only one commit at a time to a branch. Cherry-picking many commits at once can have unpredictable consequences. The work in Git should be tracked by single commits because when multiple commits are squashed into one, Git cannot guarantee the merging to be correct. This is because the history of the work done is lost and if a mistake was made in one of the commits, it is not possible to trace it back. Consequently, the whole group of commits must be reverted.

The commits in Git can be reverted but the process becomes increasingly complicated as new changes are pushed and different commits pile up. The work done by other developers can be on the same functionality as the earlier commit. Trying to revert the older commit will produce multiple merge conflicts with the later developed

functionalities. These conflicts need to be resolved by rewriting the latter changes so that they work with the older functionality.

Git works with multiple test environments. To successfully work it needs a branch which is associated with the test environment. As the need arises to get a commit into a specific test environment, it just needs to be cherry-picked to the specific branch.

Below we shortly describe the most used terms in Git:

Commit - change done to the repository

Repository – a workspace for the team in Git

Branch – own version of the code which is separated from the master version of the code

Master - the default branch in Git

Pull – to fetch the contents of a branch

Push – to commit the local changes to the repository

Merge – to combine the changes in different branches or commits

Git does not need continuous internet connection to work. Internet connection is required only when the developer wants to pull the latest changes or when he needs to push the current work to the remote repository.

To avoid the standard usage of Git in a command prompt and make it easy to use for everyone there are tools which provide an easy graphic interface for all its commands. Git Extension is one of these tools which hides the intimidating commands of Git and replaces them with intuitive buttons which do the same thing.

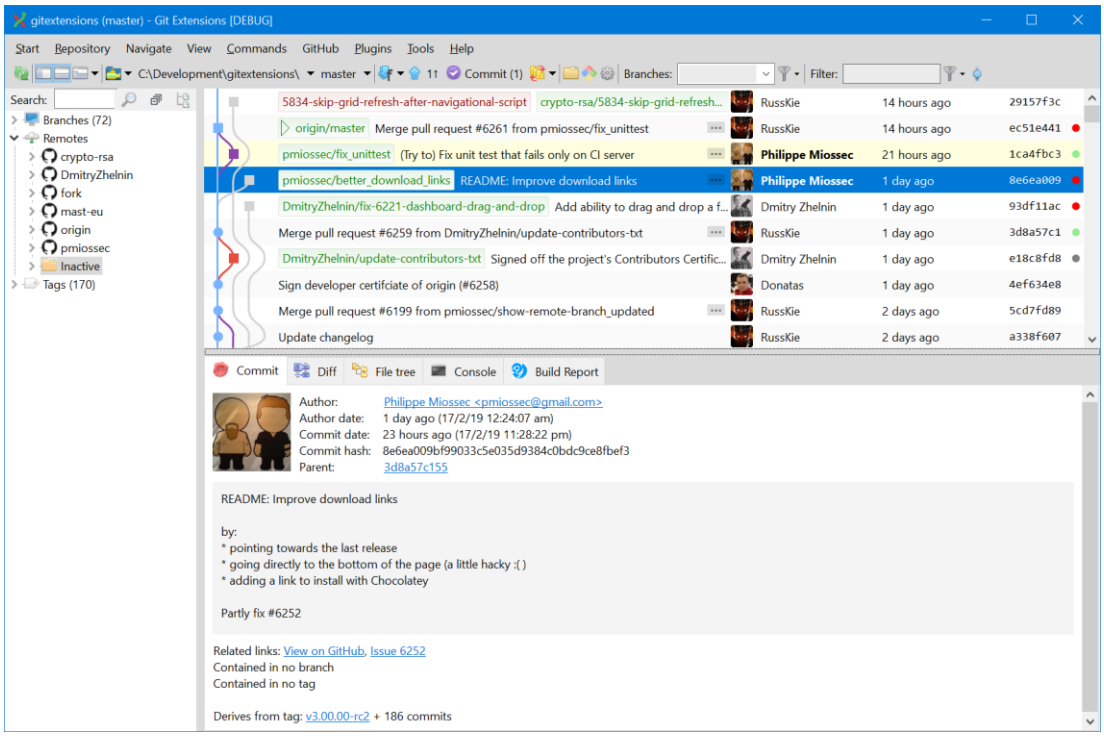

Figure 28. Example of the Git Extension user interface [29]

For somebody who has never used Git Extensions before, it can be quite confusing in the beginning. In Figure 28, the master branch is opened and there are many commits made by different developers in the repository. A merge conflict is waiting to be resolved before the work can be committed to the repository. Also 11 commits can be seen which have not yet been pushed to the repository. Furthermore, on the left of Git Extensions the different branches the user has on the local Git repository can be seen. To be able to see the branches in the repository the user must choose the option to check out the remote branch; it is not visible how to do it in the picture.

3. Requirements

The requirements of the work described in this thesis are based on IF's protocol where the definition of a complete work is stated. The goal is to implement the expandable component from the Visual Identity Library and deploy the change to My Pages production environment. The whole task is done by me and without any contribution from other team members. Below I am providing a list of the formerly mentioned requirements:

- Use Azure DevOps Boards to inform the progress of the work
- Attend the daily meetings with the team
- Implement the new expandable elements from the Visual Identity Library
- Implement the older functionality tied to the older expandable elements
- Make the tests consistently pass
- Pass UI review
- Pass code review
- Pass Sprint testing and Acceptance testing
- Deploy to Production

It is necessary to attend the daily team meetings and to update the Azure DevOps Boards to reflect the work done to the expandable elements. Implementing the new Visual Identity Library expandable element while keeping the custom functionality of the older expandable elements is important. And it is crucial to keep the custom JavaScript functionality and the custom way of displaying the information from the older expandable element. The feature tests all need to pass to ensure that the functionality is still working as intended. Also, the feature tests need to be made such that they consistently pass multiple runs on the same machine. The layout needs to pass a UI review and the code needs to pass a code review for it to be able to be tested. The sprint testing and acceptance testing need to pass to make it acceptable for the production deployment.

4. Solution

At the beginning of a new user story, it is crucial to read through the user story and to understand the requirements. Then the identification of the components or services required for the work of the user story is done. The requirements demanded the usage of the expandable element component from the Visual Identity Library. To succeed in the implementation, it is necessary to understand the expandable element examples from the Visual Identity Library. They provide the needed resources to correctly implement the expandable element. Additionally, it is necessary to understand the difference between the new and the old expandable element. This supports the implementation of the new expandable element and facilitates the deletion of the old element.

The user story is split into smaller tasks to have an oversight of the work done. It is necessary to split the story into smaller tasks which are manageable (e.g. not too large) and solve them once at a time. Dividing the big task into smaller ones avoids the situation when only one task occupies the DevOps board and keeps the team updated about the progress. Creating a branch in the Git repository is done before the start of the development to support the continuous integration. The developer commits the changes regularly to the branch to keep the work available to the other members of the team. The creation of the tasks and the branch allows for the work to be done by multiple developers. It is necessary to merge the master branch to the branch used for the user story frequently. When the changes of other developers are on the same file, merge conflicts happen. These conflicts should be fixed as soon as they arise, because the help of the developer who made the changes might be needed for their resolution.

The Visual Identity Team follows the requirements which apply to the whole company when making the components. The expandable element supports only one value in the header by default. The requirements for the My Pages team involved improving the visual aspect of the expandable element. Improvement had to be made on the way that the information was structured in the header of the expandable element to make it clearer. It was necessary to have the changes incorporated into the expandable element component to make it consistent. The information in the old implementation was

already structured properly, thus making it easier for the developers to work on the task.

The grid system was essential to use in this case, since it supports different devices and is easily adapted. The idea of using tables was quickly rejected in favor of a grid system which achieves more structure, supports different devices, and is more manageable to work with. Using tables in this case would be more difficult because the column structure would need to be remade.

```
<div class="if grid">
    <div class="if row">
        <div class="if col-9--sm col-12--xs">
            <span>Insurance name</span>
        </div>
        <div class="if col-3--sm col-12--xs u-text-right-up--sm">
            <span>100 €</span>
        </div>
    </div>
</div>
```

Figure 29. Code example of the grid system used for the expandable element

Figure 29 shows the code responsible for producing the grid system in the expandable element. It can be tailored to the needs; it supports multiple values in the header. The CSS classes col-12, col-9 and col-3 have their respective styling in the Visual Identity Library. The modifiers *xs* and *sm* apply the styling for the columns in the mobile device and the tablet device and above respectively. The code in Figure 29 is used in every expandable element to make it consistent.

Figure 30. Example of the structure of the expandable element on a desktop device

Figure 30 shows what the grid system looks like on a desktop device. The grid system is clearly marked in red to illustrate the change. In this case, the insurance value uses 9 out of the 12 available columns. The price uses three columns and is right aligned.

Figure 31. Example of the structure of the expandable element on a mobile device

Figure 31 illustrates the compacted grid system looks like on a mobile device. The columns are full width to make best use of the available space. The amount of information presented on the mobile device was in some cases too much. The expandable elements on mobile devices were optimized by shortening or rewriting the text. Additional adjustments were done to the default padding of the expandable element. It created more available space for the information, thus, solving the problem of the available space inside of the expanded area which was too small, such as in the cases where there are nested expandable elements with a product matrix inside. In the former case, the matrix needs more space than what is available in the expandable element. These changes improved the way that the information is displayed.

```
@media #{$mobile-device} {
    .if.content > * .if.panel.is-expandable {
        & > .if.title {
            padding-left: 1rem;
            padding-right: 3rem;

            &::after {
                right: 1rem;
            }
        }

        & > .if.title + .if.content {
            padding-left: 1rem;
            padding-right: 1rem;
        }
    }
}
```

Figure 32. Example of the overrides of the mobile expandable element

Figure 32 illustrates the CSS which overrides the default spacing on mobile devices. The media query ensures that the CSS only applies to the mobile devices. The variable *mobile-device* is defined in a global CSS file which is responsible for selecting only the devices which are smaller than 610 pixels. The CSS selects all the nested expandable elements and applies the override to the title CSS class.

The finished expandable elements are used throughout the My Pages solution in all the environments. The next illustrations of the finished expandable elements examples were taken from the insurance page on the self-service portal of My Pages on a test environment.

```
<div class="if panel is-expandable light">
    <div class="if title">
        <div class="if grid">
            <div class="if row">
                <div class="if col-9--sm col-12--xs">
                    <span>Mopedförsäkring</span>
                </div>
                <div class="if col-3--sm col-12--xs u-text-right-up--sm">
                    <span>4 559 kr</span>
                </div>
            </div>
        </div>
    </div>
    <div class="if content">
    </div>
</div>
```

Figure 33. Example of the code for the expandable element

Figure 33 illustrates the code of the expandable element, the structure and the CSS classes needed to produce it.

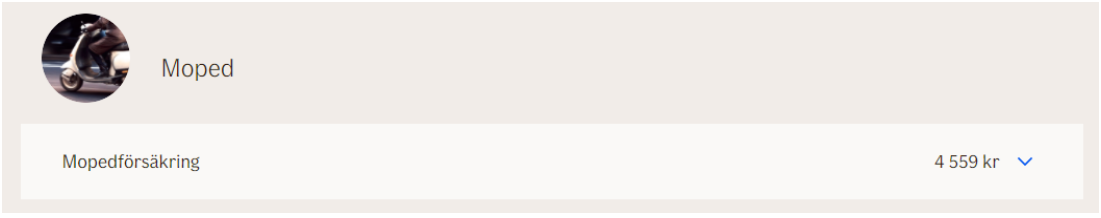

Figure 34. Example of the finished expandable element on a desktop device when closed

Figure 34 is an example of the expandable element when viewed on a desktop device. In the picture above the expandable element represents the insurance.

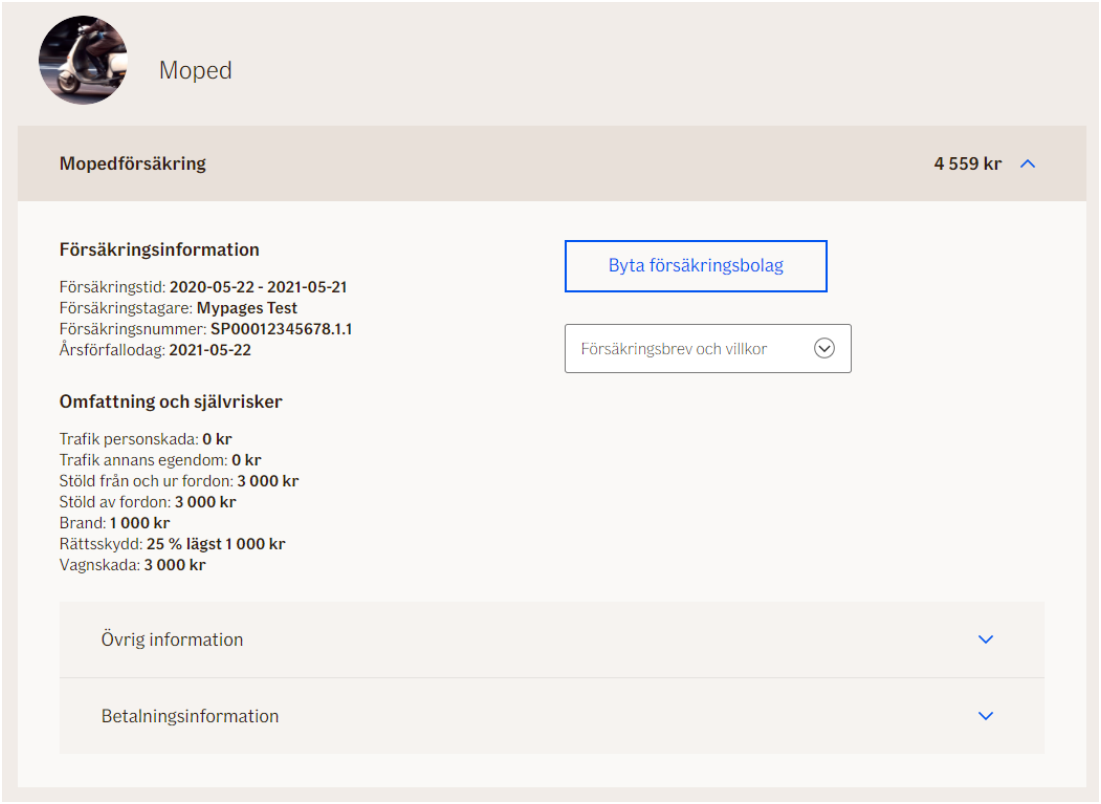

Figure 35. Example of the finished expandable element on a desktop device when opened

Figure 35 shows the expandable element when opened, revealing the contents inside of the expandable element. The contents are the following: descriptive insurance information, an insurance action, documents associated with the insurance and two additional expandable elements which contain more information regarding the insurance.

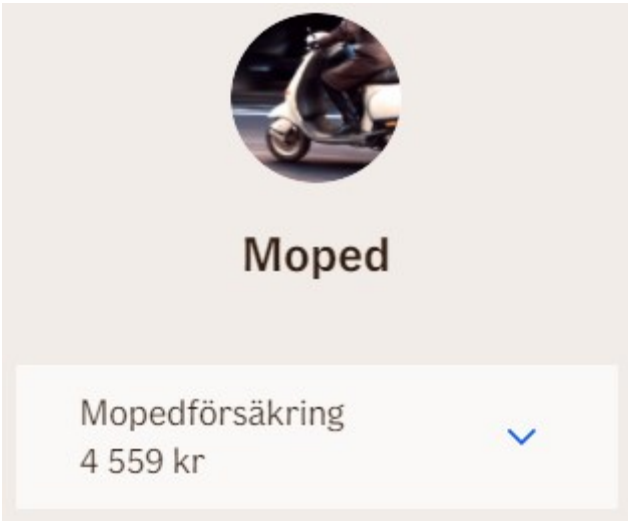

Figure 36. Example of the finished expandable element on a mobile device when closed

Figure 36 shows the expandable element when closed on a mobile device.

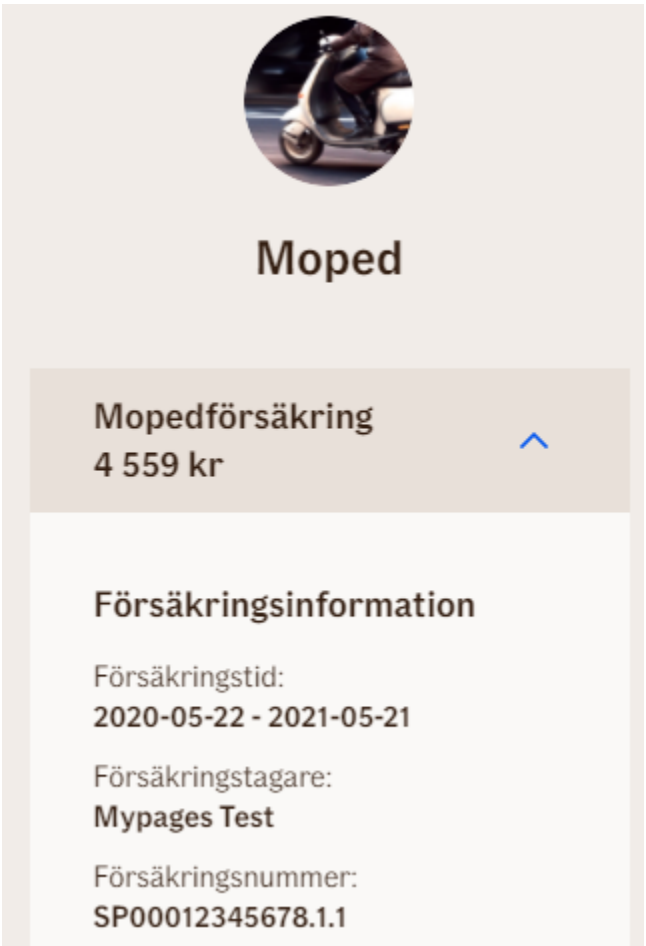

Figure 37. Example of the finished expandable element on a mobile device when opened

Figure 37 illustrates the expandable element in an opened state. The information is compressed to fit the smaller devices while still maintaining usability.

The opening and the closing of the expandable element is done by jQuery functionality. It appends the *is-open* CSS class to the expandable element which renders the expandable element in an open state. If we remove the CSS class, only the header of the expandable element is rendered. The jQuery functionality is triggered when the expandable element is clicked.

```
$(".if.panel.is-expandable .if.title").click(function () {
    OpenExpandable($(this).closest('.if.panel.is-expandable'));
});
$(".if.panel.is-expandable .if.title").on("keypress", function(e) {
    if(e.which != 13) {
        return;
    }
    OpenExpandable($(this).closest('.if.panel.is-expandable'));
});
function OpenExpandable(expandable) {
    var expandableTitles = $(expandable).find(".if.title");
    var expandableContent = expandable.find(".if.content");

    expandableTitles.attr("tabindex", "0");
    if (!expandable.hasClass("is-open")) {
        expandableContent.removeAttr("tabindex");
    } else {
        expandableContent.attr("tabindex", "-1");
    }

    expandable.toggleClass("is-open");
}
```

Figure 38. Example of the jQuery code for opening and closing the expandable element

Figure 38 illustrates the jQuery code responsible for opening and closing the expandable element. The jQuery function listens to a click event on the expandable element and executes the functionality. It finds the *if title* CSS class and appends the *is-open* class, it also adds the tabindex to the element. The tabindex makes it possible to use the tab key on the keyboard to focus the expandable element [30]. The second event listens to the enter key and triggers the opening of the expandable element.

There were specific undesired cases when the custom functionality of the expandable elements rendered them in the opened state when the web page was loaded. One of

these situations was in the case of clicking the insurance on the Overview page where the old functionality would redirect the user to the insurance page, it would scroll to the insurance and open it by default.

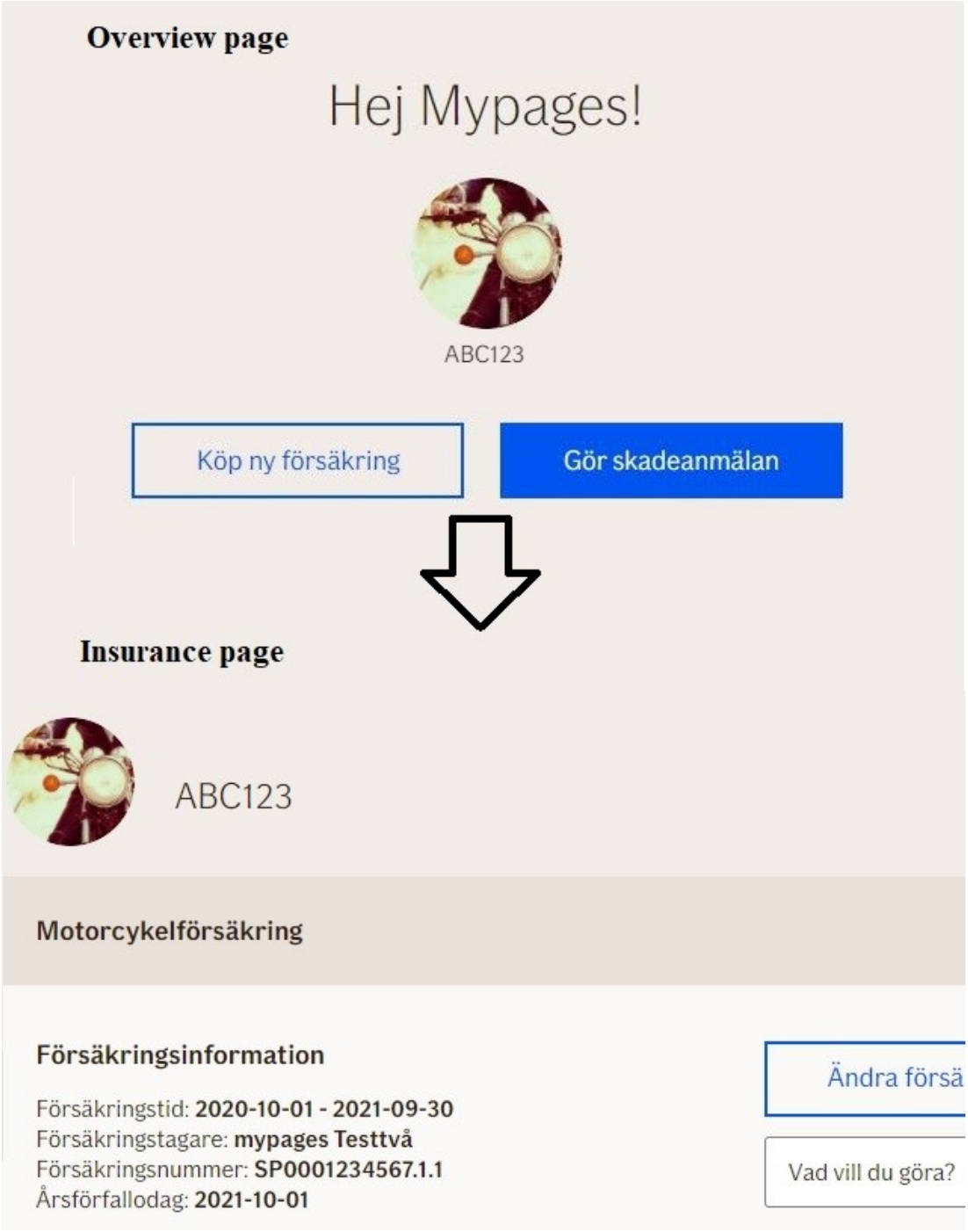

Figure 39. Example of how the JavaScript functionality works

Figure 39 illustrates how the custom JavaScript functionality works. The insurance is visible on the overview page, the user can click it to view the full details. The action redirects the user to the insurance page where it scrolls to the expandable element and opens it by default. The requirement was that when the web page is loaded with a unique id in the URL parameter, the expandable element is rendered as opened. In the markup of the expandable element there is a unique id. This id can be anything unique, e.g. insurance id, invoice id or the document id. When the id in the URL query parameter and the id of the expandable element matches the functionality in the markup, it appends the *data-selection=true*. Whenever a page is loaded the jQuery functionality searches for any expandable element with the *data-selection=true* tag. The functionality is executed, and the action is seen as the expandable element being scrolled and opened during page load.

```
function findExpandable() {
    return $(".if.panel.is-expandable[data-selection=true]");
}

function expandSection() {
    var toExpand = findExpandable();

    if (this.length === 0) {
        return this;
    }

    toExpand.scrollTo(10);

    var headers = toExpand.find('.if.title').first();

    if (headers.length === 1) {
        headers.trigger('click');
    }
};
```

Figure 40. Example of the JavaScript functionality to open the expandable element

Figure 40 illustrates the functionality for opening and scrolling to the expandable element. The custom functionality for the old expandable elements was debugged thoroughly to understand all the scenarios. This was necessary because the functionality was split in many smaller functions which were used in other places, such as: just scrolling to the expandable element, or opening the inner expandable element.

4.1. Flaky tests

By flaky tests it is meant tests which randomly fail sometimes on the same machine without any interference. It poses a challenge because tests should be coherent to make them of a use for the testing purpose. Otherwise, the test is considered of no use because it does not test the functionality properly and gives false negatives. The work was to debug by any means necessary and figure out why exactly the test failed and how to properly fix it. It was made harder by the fact that running the test locally on the machine and running the same test on a TeamCity agent did not have the same outcome. This was also something which had to be figured out why it was the case.

The feature tests in TeamCity are set up by My Pages so whenever a feature test fails, a screenshot is taken as a last step before it exits the feature test. The screenshot is saved both on the machine and in TeamCity. It is done to facilitate the work by figuring out at what step it failed in the feature test scenario. By knowing what the last thing on the screen was, it helps figuring out what caused the test to fail. But this information alone does in no way guarantee success, it still must be reproduced by the programmer and debugged.

The main reason for the feature test failing turned out to be related to a timing issue. This is the time difference between the moment when the HTML DOM has finished loading and the time when the feature test asserts the DOM. The timing issue was caused by the animation that the new expandable elements have when they are opened. The feature tests did not wait until the opened expandable element was visible before asserting the information. Any changes to the DOM after the initial page load must be properly waited by the feature test. This is done by waiting for the element in the DOM structure of the information that the test is asserting to be fully visible. We implemented the former procedure by using the wait until functionality from the Specflow library. It ensures that the feature test asserts the layout only when it has been properly loaded. This way the success of the feature test is guaranteed even when the DOM structure is not consistent in loading time. The timing issues were not just caused by the expandable elements. Inconsistencies with JavaScript functions or jQuery calls also made the feature test behave in a flaky manner. The same technique

that we applied to correct the expandable elements was used to fix these issues as well. These fixes made the feature tests more consistent and less prone to fail.

Another reason which causes flaky tests is connected to the way that the tests are run from the agents in TeamCity. The TeamCity agent runs the feature tests in a simulated screen which resolution is lower than expected. Consequently, the window does not fit the same amount of information as it would if it would run the test on a local machine and as a result the test fails. The TeamCity agent is set up to periodically call the TeamCity server to request new work [31]. The work is downloaded to the agent and the feature tests are run on the machine in a headless mode. The limitations of Windows accessing a remote computer via remote desktop causes issues when running the tests [32]. The issues come because of the feature tests to run in much lower resolution than the expected 1920x1080 (Full HD) resolution. The behavior was confirmed by running a feature test and taking a screenshot of the window in a certain test case. By examining the screenshot, it was confirmed that the resolution of the TeamCity agent was 1024x768 pixels. One way of solving this problem is by writing a script which runs every time the agent is turned on it logs in as a user. No work was put into setting up a script or any other solution which would fix the problem and run the feature tests in Full HD resolution. This was due to TeamCity being replaced soon with Azure Pipelines in the My Pages team. More focus was put on fixing the underlying problem that the test running in a lower resolution entailed. The problem of lower resolution on the TeamCity agents was fixed by implementing proper scrolling and focusing on the elements. It was done by using the Selenium WebDriver functionality *MoveToElement*. This ensures that the elements are focused and visible in the window when the scenario tries to interact with the content.

4.2. Custom fonts in the Azure Cloud

The testing of the expandable elements uncovered an issue with the fonts which was crucial to fix. The new custom fonts from the Visual Identity Library did not work as required in the expandable elements. The custom font did not display correctly, but the next defined font available was shown instead.

The CSS property font-face defines in order the fonts which are used on a CSS class, if it cannot find one font it takes the next one in order. The least important font needs to be the one which is available on any computer to ensure that the text is displayed. On a Windows PC the fonts such as Arial and Microsoft Sans Serif are preinstalled [33]. The web fonts WOFF and WOFF2 provide lightweight and easy-to-implement compression of the font data which is suitable to use for web use with CSS [34]. The custom fonts are delivered by the Visual Identity Team and used by all the components.

The issue was investigated by debugging, checking the Chrome console, and the network tab in Chrome which made it obvious that the custom fonts were not loading. After investigating the problem, we realized that the custom fonts are not defined by default in the Azure IIS. A configuration change was needed in the project to make the Azure IIS to allow the custom font definition [35]. In the next deployment the configuration change properly configured the IIS settings and the web application correctly loaded the custom fonts.

```
<system.webServer>
  <staticContent>
    <remove fileExtension=".woff" />
    <mimeMap fileExtension="woff" mimeType="font-woff" />
    <remove fileExtension=".woff2" />
    <mimeMap fileExtension="woff2" mimeType="font-woff2" />
  </staticContent>
</system.webServer>
```

Figure 41. Example of the configuration change needed to make the custom fonts load

In Figure 41 we can see an example of the configuration change needed to allow the custom fonts WOFF and WOFF2 to load in the Azure IIS configuration. Any previous declaration of the custom fonts is removed with the command remove *fileExtension* and with the extension defined. There can be older custom font configurations defined in the environment and as a result the new configuration will not work since it is already defined. Thus, as a precaution it is necessary first to remove then add the new configuration in the IIS.

4.3. Hotfix of the expandable element

Shortly after the production deployment the testers found an issue affecting the expandable elements. The error affected the users who visited the self-service portal by using the IE11 web browser. IE11 is rarely used to browse the internet nowadays but it is still deemed business critical to IF. Due to business criticality a hotfix was needed to ratify the problem with the expandable elements as soon as possible.

The Visual Identity Library only provides a style library and no functionality tied to the components. The functionality for each component must be implemented by the team. Thus, it was our responsibility to make it work in all the browsers.

The problem was that a *forEach* JavaScript function which is not supported by the browser IE11 [36] was used for the logic of opening and closing the expandable element. The other major browsers support the JavaScript function properly, so the problem was not immediately obvious. The unsupported JavaScript function made all the custom My Pages JavaScript and jQuery functionality does not work in the IE11 browser. The reason was because of the way JavaScript files are bundled in My Pages solution to save data when transmitting the files to the user. This way of bundling the JavaScript files is also present in the testing environments.

The fix was to rewrite the logic for the expandable element in jQuery instead of JavaScript. The jQuery library supports all browsers, and the functions are cross-browser compatible [37]. The rewrite of the expandable element logic was delivered to the test environments after the fix. Then a hotfix was planned after the functionality was accepted by the testers. The expandable elements and all the custom JavaScript functionality worked after substituting the previous implementation with jQuery.

5. Evaluation

The evaluation of the expandable elements takes a lot of time and involves many individuals through the process. To successfully finalize the evaluation of the expandable elements the process must be thorough. The focus is placed on visual inspection of the layout and verification of the content of the expandable elements. Also testing the custom functionality of the expandable elements. Below we go through the different evaluation steps, from the perspectives of the individuals involved.

During development it is necessary to visually check the actual changes because only a visual inspection of the layout can guarantee accuracy. Relying on the visual change exclusively by looking at the code in the solution is not an acceptable approach. It may occur that the CSS classes or other functionality will distort the layout. Commonly used CSS classes in a solution can e.g. add margin to a component. The BEM structure prevents such classes, but older code has not been adapted to the new structure. Errors related to the expandable elements that are noticed during the development of the layout are necessary to fix immediately. The errors which have a bigger impact on the expandable elements are discussed with Lead tester, UX Manager or the Visual Identity Team. The solution agreed between the people responsible is deployed by the My Pages team or by the Style library. This depends on the nature of the issue, whether it is related to the common expandable element functionality, or the custom My Pages functionality.

The automated feature tests are responsible for testing the expandable elements. They are testing the custom functionality and the way the information is displayed in the expandable elements. The feature tests have been present before the start of the process to replace the expandable elements. To be able to complete part of the evaluation process it is necessary to have the feature tests run successfully. The feature tests are either run on the local machine or in TeamCity as part of the continuous integration.

A review of the developed UI is made to ensure that the UI follows the requirements. The review takes place between the developer and the UX Manager. The process consists of going through the developed layouts and verifying that it is as required.

The problems found during the UI review are recorded and fixed by the developer. Another review is planned to verify the proposed changes and to finalize the layout.

The code is checked by another developer in the team before it is published to a test environment. Publishing is done by creating a pull request which facilitates the code review. Azure DevOps offers the feature to make pull requests for the Git Repository. The pull request checks the changes done in a branch and compares them to the master branch. The difference in the files is shown in the pull request. The code reviewer can see the changes in a comprehensible way and is able to comment on the code. It is up to the developer to reflect and implement the changes in the pull request. The code review is based on a common rule setup by My Pages team and the experience of the code reviewer. The code review must be completed before the code is pushed to a test environment. It ensures that another developer has checked that the code does not have issues such as: naming inconsistencies, logic errors or improper indentation.

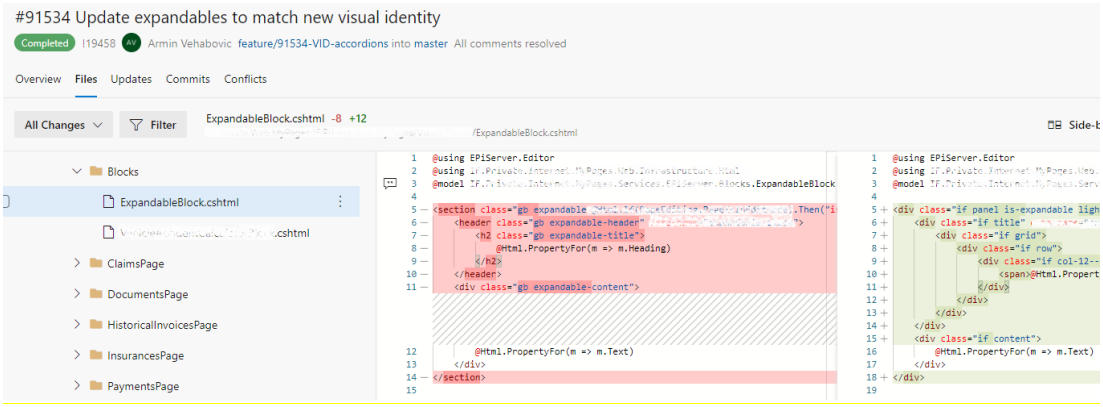

Figure 42. Example of a Pull Request made in Azure DevOps

Figure 42 shows the pull request for the work done to the expandable elements. The code changes done in the file *ExpandableBlock.cshtml* are clearly displayed, the red changes are code that was removed, and the green changes are code that was added.

The software tester is responsible for creating the test data to properly test all the requirements of the user story. The test data is used to go through all the pages and visually verify the implementation of the expandable elements. At the same time the tester needs to validate the custom JavaScript functionality. JavaScript code used in

the expandable elements will be difficult to grasp for a tester without JavaScript programming skills. Hence it is important for the tester and the developer to keep close collaboration during this step of the process. The requirements which are not implemented as specified are recorded and the code is retested when the fix is made available.

The Acceptance testing is the last part of the validation before publishing the work to the production environment. The final visual verification of the expandable elements is done by the Lead Tester and the UX Manager. To successfully verify the expandable elements, they need to get familiarized with the specifications before they start the verification. The specifications are defined by the Visual Identity Team. The look and feel of the expandable element in My Pages need to match the examples listed in the Visual Identity Library. Moreover, the custom changes to the expandable elements made before the new implementation was started still need to work correctly. Errors with the expandable elements are reported and retested when a fix is made available.

The testing and verification of the expandable elements ensures the expandable elements are made according to the requirements. After the verification is done, the expandable elements are published to production.

6. Conclusions

The expandable elements proved to be a big challenge to overcome. The lessons learned have been many and the work done has been successful. The new expandable elements have changed the web pages for the users of the self-service portal My Pages. The change is a small step in the big Visual Identity overhaul. The work continues with reshaping My pages and implementing more new Visual Identity components.

The work started with understanding the requirements and familiarizing with the Visual Identity Library component. The work consisted of reimplementing the previously used expandable elements with the new expandable element component. Resolving the custom JavaScript functionality by debugging the code and understanding the functionality. Improvement of the functionality by making it work for more cases was made. One such example is in the case where the user clicks on the insurance on the overview page and is redirected to the insurance page, where the insurance is focused, and the expandable element is opened. With the improvement of the functionality the previously described feature now works for the insurance of all types.

Furthermore, the feature tests were improved in such a way to consistently succeed throughout runs. It was crucial to understand why the feature tests randomly failed on the same machine without any interference. By fixing the core issues made the feature tests more consistent and less prone to fail. The work was completed by finishing all the requirements of the user story and getting a positive remark from the lead tester and UX manager for production deployment.

The work of implementing and verifying the new expandable elements is already done. The work done is in the production environment as of writing the documentation. The customers of IF who visit the self-service portal, My Pages, are going to see the new expandable elements. The Visual Identity Library is a company-wide style library thus all the pages are expected to soon have the new expandable elements. The Visual Identity overhaul of My Pages and IF’s pages continues after the thesis work with more components being implemented. The lessons learned from the work on the expandable elements components is going to be useful and will ease further development.

7. Discussions

There were about 100 expandable elements changed throughout the solution as part of the thesis work. The changes were done in approximately 133 files, 450 SLOC were added and 750 SLOC were removed from the solution. The reason why the number of files that were changed is larger than the actual number of expandable elements is due to adding custom JavaScript functionality (files and feature tests), as well as modifying existing files. Most of the expandable elements did not have the custom JavaScript functionality and in those cases the change that had to be made was straightforward. In the cases where the expandable elements had JavaScript functionality, debugging was necessary to ensure proper understanding of the old functionality. Only by understanding the outcome of the functionality, it was possible to adapt the JavaScript and fit it to the new expandable elements.

Most of the time for implementing the expandable elements was spent fixing minor issues found by the software testers. The implementation which allows the displaying of the information in an expandable element is not trivial, it was made by defining the number of columns for each item in the header of the expandable element.

The changes made to the expandable elements had to be rolled out on a scheduled production release and preferably with a smooth experience. It was important to ensure a minimum discomfort to the customers logged in to My Pages. The changes were delivered to production in the early mornings when the traffic was lower, to avoid large numbers of customers being affected.

There were several issues reported by software testers and there were not My Pages specific. Only a few problems with the expandable elements were My Pages specific (i.e. they were caused by the custom functionality implemented). These issues were fixed on My Pages side. Issues related to the component itself were reported to the team responsible for the Visual Identity Library. The Visual Identity Library team continues the work to improve and fix the new components, and blocks. The fixes made to the expandable element in Visual Identity Library are not just meant for My Pages, but for every team using them. Based on the expandable element component, the benefits of using one style library company-wide are obvious (e.g. single source of

truth, shared language between the developers and the designers, increased brand trust and reduced redundancy). Using a common style library promotes more consistent UX experience across the applications and contributes to the overall homogeneity in the look and feel of the digital services provided by If.

References

[1] Bootstrap Accordion – examples & tutorial. Available from:
https://mdbootstrap.com/docs/jquery/javascript/accordion/#e-table-accordion
(Last read: 20.08.2020).

[2] HTML Introduction. Available from:
https://www.w3schools.com/html/html_intro.asp
(Last read: 30.09.2020).

[3] What is CSS. Available from:
https://www.w3schools.com/whatis/whatis_css.asp
(Last read: 12.10.2020).

[4] Sass Introduction. Available from:
https://www.w3schools.com/sass/sass_intro.php
(Last read: 30.09.2020).

[5] Specificity – CSS: Cascading Style Sheets. Available from:
https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity
(Last read: 27.09.2020).

[6] BEM – Block Element Modifier. Available from:
http://getbem.com/introduction/
(Last read: 22.09.2020).

[7] About JavaScript – JavaScript | MDN. Available from:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript
(Last read: 30.09.2020).

[8] Introduction – JavaScript | MDN. Available from:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction
(Last read: 30.09.2020).

[9] jQuery. Available from:
https://jquery.com/
(Last read: 30.09.2020).

[10] jQuery Introduction. Available from:
https://www.w3schools.com/jquery/jquery_intro.asp
(Last read: 30.09.2020).

[11] About - Bootstrap v4.5. Available from:
https://getbootstrap.com/docs/4.5/about/overview/
(Last read: 20.08.2020).

[12] Introduction - Bootstrap v4.5. Available from:
https://getbootstrap.com/docs/4.5/getting-started/introduction/
(Last read: 20.08.2020).

[13] Grid System - Bootstrap v4.5. Available from:
https://getbootstrap.com/docs/4.5/layout/grid/
(Last read: 20.08.2020).

[14] Overview – Bootstrap v4.5. Available from:
https://getbootstrap.com/docs/4.5/layout/overview/
(Last read: 20.08.2020).

[15] Azure DevOps Services | Microsoft Azure. Available from:
https://azure.microsoft.com/en-us/services/devops/
(Last read: 03.10.2020).

[16] Understand what you get with Azure Boards – Azure Boards. Available from:
https://docs.microsoft.com/en-us/azure/devops/boards/get-started/what-is-azure-boards?view=azure-devops&tabs=agile-process
(Last read: 30.08.2020).

[17] What is Scrum? Available from:
https://www.scrum.org/resources/what-is-scrum
(Last read: 08.10.2020).

[18] Welcome to SpecFlow's documentation! Available from:
https://docs.specflow.org/projects/specflow/en/latest/
(Last read: 13.09.2020).

[19] Getting Started with Behavior Driven Development. Available from: https://specflow.org/bdd/
(Last read: 04.10.2020).

[20] Gherkin Reference – Cucumber Documentation. Available from: https://cucumber.io/docs/gherkin/reference/
(Last read: 13.09.2020).

[21] Chromedriver - WebDriver for Chrome. Available from: https://chromedriver.chromium.org/
(Last read: 10.09.2020).

[22] W3Counter: Global Web Stats. Available from: https://www.w3counter.com/globalstats.php
(Last read: 27.09.2020).

[23] Selenium WebDriver. Available from: https://www.selenium.dev/documentation/en/webdriver/
(Last read: 13.09.2020).

[24] Continuous Integration with TeamCity. Available from: https://www.jetbrains.com/help/teamcity/2020.1/continuous-integration-with-teamcity.html
(Last read: 10.09.2020).

[25] Setting up and Running Additional Build Agents – Help | Teamcity. Available from: https://www.jetbrains.com/help/teamcity/setting-up-and-running-additional-build-agents.html#Installing+via+Windows+installer
(Last read: 08.10.2020).

[26] TeamCity: The Hassle-Free CI And CD Server by JetBrains. Available from: https://www.jetbrains.com/teamcity/
(Last read: 21.09.2020).

[27] Test Automation with TeamCity. Available from: https://blog.jetbrains.com/teamcity/2019/08/test-automation-with-teamcity/
(Last read: 21.09.2020).

[28] Git Handbook – GitHub Guides. Available from:
https://guides.github.com/introduction/git-handbook/
(Last read: 31.08.2020).

[29] Git Extensions. Available from:
http://gitextensions.github.io/
(Last read: 31.08.2020).

[30] tabIndex – HTML: HyperText Markup Language | MDN. Available from:
https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/tabindex
(Last read: 18.10.2020).

[31] Setting up and Running additional Build Agents – Help | Teamcity. Available from: https://www.jetbrains.com/help/teamcity/setting-up-and-running-additional-build-agents.html#Agent-Server+Data+Transfers
(Last read: 03.09.2020).

[32] Known issues – Help | TeamCity. Available from:
https://www.jetbrains.com/help/teamcity/known-issues.html#Issues+with+automated+GUI+and+browser+testing
(Last read: 03.09.2020).

[33] Font List Windows 10 – Typography | Microsoft Docs. Available from:
https://docs.microsoft.com/en-us/typography/fonts/windows_10_font_list
(Last read 09.10.2020).

[34] WOFF File Format 1.0. Available from:
https://www.w3.org/TR/WOFF/
(Last read: 09.10.2020).

[35] How to include a non-standard font-face in azure hosted website without using visual studio? – Stack Overflow. Available from:
https://stackoverflow.com/questions/23831568/how-to-include-a-non-standard-font-face-in-azure-hosted-website-without-using-vi/23835696
(Last read: 27.08.2020).

[36] Can I use… Support tables for HTML5, CSS3, etc. Available from:
https://caniuse.com/?search=forEach
(Last read: 30.08.2020).

[37] Browser Support | jQuery. Available from:
https://jquery.com/browser-support/
(Last read: 28.08.2020).

Svensk sammanfattning

Processen för att byta ut expanderbara element i en storskalig webbapplikation

Introduktion

Det nordiska försäkringsbolaget If utvecklar för tillfället en ny företagsbild som kommer definiera dess nya visuella identitet. My Pages-teamet, som ansvarar för den nordiska självbetjäningsportalen, gör det stora arbetet i mindre iterationer. Syftet med avhandlingen är att beskriva processen som My Pages-teamet följde för att successivt uppnå målet.

Ett expanderbart element har den viktigaste informationen synlig när det är hopslaget och dess klickbara egenskap är uppenbar. Ett klick på expanderbara elementet skjuter ner innehållet på webbsidan. En gömd yta under det expanderbara elementet framträder, och visar mera information. Här beskrivs arbetet som utfördes för att implementera det nya expanderbara elementet från Ifs egna Visual Identity Library. Beskrivningen börjar med implementeringen av de expanderbara elementen, och följs upp av de anpassade ändringar vilka medför mera funktionalitet till de expanderbara elementen. Slutligen, svårigheterna och åtgärderna för att laga alla tester av de expanderbara elementen. Dessa tester säkerställer att de expanderbara elementen har rätt innehåll även då nya ändringar tillkommer.

Visual Identity Library är Ifs egna designbibliotek som används av alla team för att skapa webbapplikationer för internt och externt bruk. Teamet som har hand om den ansvarar för att skapa nya komponenter och underhåller designbiblioteket. Biblioteket tillhandahåller mallar för expanderbara element, specifikationerna, kodexempel och JavaScript pseudokod. De tillhandahållna tillgångarna är menade som hjälp för team i If som använder sig av komponenterna i Visual Identity Library.

Implementation

Det är viktigt för programmeraren att börja med att noggrant läsa igenom och förstå kraven för användarberättelsen. Användarberättelsen definierar kraven och specifikationerna för de expanderbara elementen. För att säkerställa att borttagning av gamla expanderbara element samt implementation av nya expanderbara element fungerar måste programmeraren göra följande: förstå de expanderbara element från Visual Identity Library, förstå anpassningarna av de expanderbara element som behövs för att passa in dem på My Pages, och förstå kodmässigt ändringarna mellan de gamla expanderbara element och de nya elementen. De kodexempel som tillhandahålls av Visual Identity Library utgör grundstommen för implementation av komponenten.

Arbetet med de expanderbara elementen delades upp i hanterbara uppgifter i Azure DevOps Boards för att få en översyn. Framstegen med uppgifterna uppdaterades kontinuerligt för transparens av uppgifterna. En ny gren i Git var nödvändigt för att möjliggöra versionshantering av koden. Det gjorde också möjligt att koden kan levereras till en annan gren än huvudkoden som används för projektet. Dessa åtaganden hade betydelse för att hålla teamet uppdaterat om framstegen och för att möjliggöra att arbetet kan utföras av flera programmerare.

Implementeringen av komponenten handlar om att ersätta den gamla koden för de expanderbara elementen med rätt struktur och CSS klasser. Det säkerställer att komponenten från Visual Identity Library visas på rätt sätt. De anpassade ändringarna för de expanderbara elementen som behövdes för funktionalitet fungerade på de äldre elementen. För att kunna bearbeta och förstå logiken var det nödvändigt att felsöka med de gamla expanderbara elementen. Att ändra logiken så att den fungerar på de nya elementen var trivialt. Ändringarna av de expanderbara elementen gjorde att en del tester inte fungerade. Testerna gjordes för att kunna säkerställa att webbsidan visar rätt information och att den anpassade funktionaliteten fungerar. Animationen på de nya expanderbara elementen gjorde att tester försökte kontrollera innehållet på webbsidan innan laddningen av sidan var färdig. Det åtgärdades med bättre logik, som väntade på att sidan hade laddats upp fullständigt före den kontrollerade informationen. Dessutom åtgärdades ett fel då testerna till synes slumpmässigt inte fungerade på samma maskin mellan testkörningar. Detta berodde på att exekveringen av de automatiska testerna inte gick som väntat i Teamcity Agents.

Resultat

De expanderbara elementen visade sig vara en stor utmaning att överkomma. Lärdomarna av projektet var stora och arbetet blev utfört. Biblioteken, verktygen och de olika hjälpmedlen gjorde det möjligt att fullborda arbetet. De nya expanderbara elementen har ändrat bilden av webbsidorna för användarna av självbetjäningsportalen My Pages. Ändringen är ett litet steg i den stora översynen över den visuella identiteten. Arbetet med att integrera andra komponenter fortsätter efter de expanderbara elementen.

Det här arbetet med expanderbara element förde med sig andra förbättringar av relaterade fel som hittades. Fel som inte hade upptäckts tidigare, men som hade en negativ inverkan på funktionaliteten åtgärdades. Kod för den anpassade funktionaliteten av de expanderbara elementen utelämnades i vissa fall. Detta ledde till att fall där expanderbara element fokuserades och öppnades automatiskt när sidan hade laddats färdigt inte fungerade. Webbsidan påverkades inte i högre grad av felet och sidan kunde användas utan problem.

Testerna som åtgärdades som en del av arbetet hade en positiv inverkan inte bara på de expanderbara elementen utan också på alla tester. Lärdomarna av åtgärderna har gjort att felsökning av tester har blivit snabbare och pålitligare. Ett test som inte fungerar indikerar att något är fel och att felet behöver åtgärdas. Om tester slumpmässigt inte fungerar utan någon ändring mellan körningarna tar det dyrbar tid och ger inget resultat. En stabil körning av testerna har gjort att distributionen av koden till testmiljöerna har blivit pålitligare.