A research project from Ambientia based on Storybook testing environment  
ICT Project

|  |  |
| --- | --- |
| Supervisor | Tommi Lahti |
| Stakeholder | Juha Wiberg |
| Developers/Writes | Robert Blankson Ephrem Ewnetu Saida Essadiki Taha-Yassine Ennabili  Kaisa Otterklau |

HAMK  
March 2023

Content

[1 Introduction to Storybook 2](#_Toc129089283)

[1.1 What is Storybook? 2](#_Toc129089284)

[1.2 How did we use Storybook in our project? 3](#_Toc129089285)

[2 Testing possibilities with Storybook 3](#_Toc129089286)

[3 Accessibility testing 4](#_Toc129089287)

[3.1 Set up the a11y addon 5](#_Toc129089288)

[4 Interaction testing 5](#_Toc129089289)

[4.1 Set up interaction testing 6](#_Toc129089290)

[4.1.1 Feedback form - Submit the form successfully 6](#_Toc129089291)

[4.1.2 Feedback form - Submit the form with testing result with failure, checking the form validation 7](#_Toc129089292)

[4.1.3 Feedback form - Check for the ‘required’ form validation 9](#_Toc129089293)

[4.1.4 Footer - Find a footer by given header 10](#_Toc129089294)

[4.1.5 Footer - Does the social media icon include the right link 11](#_Toc129089295)

[4.1.6 Footer - Do the links open the corresponding website in the footer? 12](#_Toc129089296)

[4.1.7 Feedback page - Are all three components visible? 13](#_Toc129089297)

[4.1.8 Date picker - Select specific date from the date picker 14](#_Toc129089298)

[4.1.9 Login form - Login using email and password in the 15](#_Toc129089299)

[4.1.10 About us – component testing 16](#_Toc129089300)

[6 Test Runner 16](#_Toc129089301)

[6.1 Setting up test runner 17](#_Toc129089302)

[7 Visual testing 18](#_Toc129089303)

[7.1 Set up visual tests using Chromatic 18](#_Toc129089304)

[7.2 Visual Test Summary 21](#_Toc129089305)

[8 Coverage tests 22](#_Toc129089306)

[9 Import stories in other tests for other tools 22](#_Toc129089307)

# Introduction to Storybook

Storybook is a development environment for UI components. It allows developers to create components independently and showcase components interactively in an isolated development environment. Storybook also enables developers to view different states of each component and develop them interactively. Developers can also collaborate with UX/UI designers, product managers and other stakeholders during the process of building UI components. The add-ons are easy to add to your project using command line and do not require complicated configuration.

Chart, box and whisker chart

Description automatically generated

For example, if the project need a banner for a page, you will need three files:

* Banner.js
* Banner.css
* Banner.stories.js

In the Banner.js file it is important to write your code as usually and depending on your programming language – JavaScript and typescript can be used. Storybook reads a file Banner.stories.js, where you are calling the Banner.js file and bring it now to Storybook environment. In Banner.stories.js file you will be building a template and naming your props as needed. Here is an example code. From the image below it can be seen that in Storybook environment we can change the prop called alt, whenever you do changes in Storybook environment – the changes will not be edited in code.

*import React from 'react';*

*import Banner from './Banner';*

*export default {*

*title: 'COMPONENTS/Banner',*

*component: Banner,*

*};*

*const Template = (args) => <Banner {...args} />;*

*export const Default = Template.bind({});*

*Default.args = {*

*alt: 'Banner',*

*};*

## How did we use Storybook in our project?

Storybook can be used for development for components, but our focus was to get more information and practical experience about the testing environment in Storybook.

Storybook provides an easy to use and clean-room environment for testing components in isolation, meaning that all components can be run through testing alone. The main focus of this project was to implement interaction testing. More info can be found in the next chapter.

# Testing possibilities with Storybook

Storybooks can be a powerful tool for testing the functionality and behavior of React components.

* **A test runner** can be employed to automate test execution and increase testing efficiency.
* To test component or page appearance, **visual tests** can be utilized.
* **Accessibility tests** can be conducted to evaluate the accessibility of components or pages.
* **Interaction tests** can simulate user behavior to assess component or page functionality
* **Coverage tests** can measure code coverage to ensure that all code has been tested.
* **Snapshot tests** can help detect rendering errors and warnings.
* **Importing stories into other tests** can enable the use of other tools in the testing process.

# Accessibility testing

Accessibility tests are used to make web applications suitable for everyone, meaning that it sopports keyboard navigation, screen reader support, touch-friendly, useable color contrast, reduces motion and zoom support. Storybook provides an official a11y addon. Powered by Deque's axe-core, which automatically catches up to 57% of [WCAG](https://www.w3.org/WAI/standards-guidelines/wcag/) issues.   
Read more here:<https://storybook.js.org/docs/react/writing-tests/accessibility-testing>.

By highlighting the different sections, we can assess the appearance and usability of our components for users with varying levels of vision. Unfortunately, certain components contain inaccessible portions that require redesign. An example accessibility report displaying the highlighted areas is presented below. The Accessibility add-on allows us to adjust the appearance and functionality of the user interface based on the percentage of users with different types of visual impairments. Graphical user interface, application

Description automatically generated

## Set up the a11y addon

Run this in your CLI and add the "@storybook/addon-a11y", in main.js file. With the setup done, you have now access to Storybook accessibility testing.

*yarn add --dev @storybook/addon-a11y*

Text

Description automatically generated



# Interaction testing

Interaction tests checks the user behavior of the component such clicks and form entries, Furthermore, it checks if the state of the component updates correctly. In the stories file it checks the behavior using the play function. The testing has been written using Storybook-instrumented versions of Jest and Testing Library. They provide playback interface and browser-based debugging. More info can be found from <https://storybook.js.org/docs/react/writing-tests/interaction-testing>.

## Set up interaction testing

Run this in your CLI and add the "@storybook/addon-interaction ", in main.js file. With the setup done, you have now access to Storybook accessibility testing.

*yarn add --dev @storybook/testing-library @storybook/jest @storybook/addon-interactions*

Text

Description automatically generated



### Feedback form - Submit the form successfully

Description: A user should be able to write the information to the form and submit it

Assumption: Storybook testing is being used

Test Steps:

1. Find ‘Name’ field and write the name
2. Find ‘Email’ field and write the email
3. Find ‘Feedback’ field and write the feedback
4. Find a button and click it

Expected Result: Form filled and test is successful

Result:

Graphical user interface, application

Description automatically generated

### Feedback form - Submit the form with testing result with failure, checking the form validation

Description: A user should be able to write his/hers name to the form but will write email with an error we should see error message and test should fail

Assumption: Storybook testing is being used

Test Steps:

1. Find ‘Name:’ field and write the email “Kaisa.”
2. Find ‘Email:’ field and write the email “kaisa.fi”
3. Message should pop up and test should fail
4. Find ‘Feedback:’ field and write the email “Kaisa.”
5. Find a button and click it

Expected Result: The test will fail in step two

Result: The test passed..?

Conclusion: Forms are not validated, need to be worked on

### Feedback form - Check for the ‘required’ form validation

Description: A user should not be able to submit the form with a missing value of name either the email

Assumption: Storybook testing is being used

Test Steps:

1. Find ‘Name:’ field and write the email “kaisa”
2. Find ‘Email:’ field and write the email “kaisa.fi”
3. Find ‘Feedback:’ field and write the email “” (Leave it empty)
4. Find a button and click it

Expected Result: The test will fail in step three and form is not completed

Result: As expected, the test failed in the step three

### Footer - Find a footer by given header

Description: Find an footer with header text

Assumption: Storybook testing is being used

Test Steps:

1. Defining a footer constraint by finding a header which includes text *“Lets keep in touch”*
2. Hover the mouse over the text

Expected Result: Storybook can find the text and hovers the mouse over the text.

Result: Text passes successfully

Graphical user interface, text, application

Description automatically generated

### Footer - Does the social media icon include the right link

Description: Does the social media icon contain the right link corresponding to list Youtube, Facebook, Twitter and Instagram

Assumption: Storybook testing is being used

Test Steps:

1. Find all icons, which share the same role as link *(.getAllByRole('link'))*
2. Expecting the first link to contain a YouTube https link
3. Expecting the second link to contain Facebook https link
4. Expecting the third link to contain Twitter https link
5. Expecting the fourth link to contain an Instagram https link

Expected Result: All social media are available and tests will pass successfully

Result: All tests passed successfully

### Footer - Do the links open the corresponding website in the footer?

Description: All social media icons are connected to a link. Trying to make sure the links open successfully.

Assumption: Storybook testing is being used

Test Steps:

1. Find all icons, which share the same role as link *(.getAllByRole('link'))*
2. Expecting the first link open a website
3. Expecting the second link to contain Facebook https link
4. Expecting the third link to contain Twitter https link
5. Expecting the fourth link to contain an Instagram https link

Expected Result: All social media websites open successfully

Result: All tests passed successfully

### Feedback page - Are all three components visible?

Description: The feedback page contains three components – banner, feedback form and a footer.

Assumption: Storybook testing is being used

Test Steps:

1. Find a banner with a text ‘Flexbox Banner’ and hover the mouse over it
2. Find a form which contains text ‘Feedback’ and hover over it
3. Find a footer with a header ‘Lets Keep in Touch’ and hover over it

Result: All tests passed successfully

### Date picker - Select specific date from the date picker

Description: User selects March 8 from the date picker component

Assumption: Storybook testing is being used

Test Steps:

1. Find a label with text *“Go to next month”.*
2. Click on the next month icon.
3. Find element by text ”8”.
4. Click day element

Expected Result: Storybook can find the text “8” and display as “You picked March 8, 2023”.

Result: Text passes successfully

A picture containing calendar

Description automatically generated

### Login form - Login using email and password in the

Description: A user should be able to input the email and password field and be able to login.

Assumption: Storybook testing is being used

Test Steps:

1. Use the getByTestId query “email” to input “[email@email.com](mailto:email@email.com)”.
2. Use the getByTestId query “password” to input "a-random-password”.
3. Using getByRole query “button” to click the button “login”.

Expected Result: Form filled and the text “Your account is ready and we should probably get you started!” displayed

Result: Test Passes successfully

Graphical user interface, text, application, email

Description automatically generated

### About us – component testing

Description: Find an about us with header text ‘About us’

Assumption: Storybook testing is being used

Test Steps:

1. Defining a about us constraint by finding a header which includes text *“About us”*
2. Hover the mouse over the text

Expected Result: Storybook can find the text and hovers the mouse over the text.

Result: Text passes successfully

Graphical user interface, text, application, email

Description automatically generated

*The sidebar interaction testing*

1. xxxx

# Test Runner

Test runner turns our stories automatically to executable tests and informs if it has rendered our stories successfully and catches the broken stories. Test runner is accessible from the command line. It is powered by Jest and Playwright. More in detail information can be found in <https://storybook.js.org/docs/react/writing-tests/test-runner>.

* ⚡️ Zero config setup
* 💨 Smoke test all stories
* ▶️ Test stories with play functions
* 🏃 Test your stories in parallel in a headless browser
* 👷 Get feedback from error with a link directly to the story
* 🐛 Debug them visually and interactively in a live browser with [addon-interactions](https://storybook.js.org/docs/react/essentials/interactions)
* 🎭 Powered by [Jest](https://jestjs.io/) and [Playwright](https://playwright.dev/)
* 👀 Watch mode, filters, and the conveniences you'd expect
* 📔 Code coverage reports

## Setting up test runner

Run this in your CLI and configure main.js as an image below.

yarn add @storybook/test-runner -D

Text

Description automatically generated

Now to run the test enter into CLI.

yarn test-storybook

As it can be seen in the image below we have tested every component of the project with a test runner to check whether they render or not. The result is that some passes others failed due to accessibility issues.

A screenshot of a computer

Description automatically generated with medium confidence

# Visual testing

Visual tests are a useful tool for catching bugs in the UI appearance. These tests take screenshots of each story and compare them to the changes made in-between, enabling verification of the web app's layout, color, size, and contrast. Storybook recommends using Chromatic to run visual tests in a cloud browser environment, which can be easily downloaded and run with zero configuration. Further information on visual testing can be found at <https://storybook.js.org/docs/react/writing-tests/visual-testing>.

Additionally, design systems provide UI components with a consistent visual design, making it easier to compare baseline screenshots for visual testing. By capturing an image of each component in the same browser and comparing those images to accepted baseline screenshots, visual testing can save your frontend team time and prevent costly UI regressions. When discrepancies are detected, you will be notified, allowing you to ensure that UI changes are correctly implemented without the need for manual review.

## Set up visual tests using Chromatic

1.Create repository in GitHub

2. yarn add -D chromatic -in the project terminal

3. Then add your project as illustrated bellow

Graphical user interface, text, application

Description automatically generated

4.In the project terminal install the token given as shown below

Graphical user interface, text, application

Description automatically generated

5. Then a link will be given to share for the teams to check remotely .

6. Continues deployment on chromatic is achieved by adding chromatic.yml file as follows

Text

Description automatically generated

6.The chromatic web interface contains Builds, Library and Manage sections whereby we can see the different builds and components and manage the project.

Graphical user interface, application

Description automatically generated

7.For continues storybook publishing we need to add the secret token in the GitHub repository’s Security setting.

Graphical user interface, text, application, email

Description automatically generated

8. Whenever there is change chromatic will let us know automatically and act accordingly.

Graphical user interface, text, application, email

Description automatically generated

9.The visual changes as illustrated below can be accepted based on the design or denied if need be. The difference and the pull requests can be done from the web interface.

Graphical user interface, application

Description automatically generated

## Visual Test Summary

Visual regression testing helps to make sure that UI components stay the same, allowing us to catch accidental changes. When unexpected differences are detected between two versions of a component, we determine if it is intentional or not - if it is intended, the baseline can be updated to reflect the changes, or else it must be fixed.

Diagram

Description automatically generated

# Coverage tests

Coverage testscheck whether the test fully covers the code. It gives a fast feedback and gives us an advantage of examining our code with the best industry best practices. It works as QA(Quality Assurance). More info about coverage tests and how to install it to your project. <https://storybook.js.org/docs/react/writing-tests/test-coverage>.

# Import stories in other tests for other tools

There is a possibility to write tests with different tools if they are standard JavaScript modules. The stories can be imported with tools such as Jest, Testing Library, Puppeteer, Cypress and Playwright to save time and maintain work. More info can be found from <https://storybook.js.org/docs/react/writing-tests/importing-stories-in-tests/>.

# Ideas for further development

Not able to use database, testing types that could have been researched more for example coverage tests, snapshot tests, import stories in other tests for other tools.