Using Storybook as a Design System for the Angular Web App TreeTest

Georg Niess, Arwin Roubal, Stefan Thurner, Enrique Barba Roque

Graz University of Technology
A-8010 Graz, Austria

02 Feb 2020

Abstract

Modern information hierarchies require testing to yield the best experiences for users navigating through them in search of content. One such approach on testing is so-called Tree Testing where users navigate through information hierarchies. Ajdin Mehic has written an Angular Web App called TreeTest for this purpose. In future work, the current TreeTest implementation can be built upon and extended in various ways to provide an even better open-source alternative to existing commercial tree testing applications. For improving, adapting and even for customizing the software further, a design system like Storybook can be very useful. This paper focuses on the usage of Storybook with the TreeTest project to find out if it is suitable as a design system for the further development of TreeTest.

Contents

List of Figures

Chapter 1

Tree Testing and TreeTest

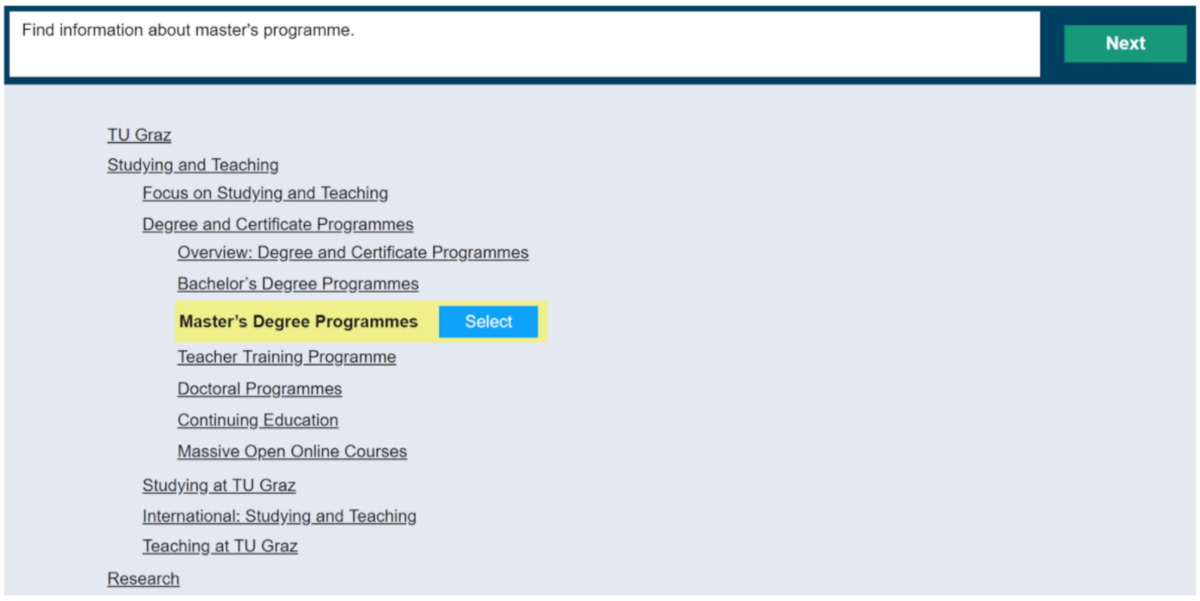
1.1 Tree Testing

In many cases, websites are dedicated to providing content to their users. They click on tabs and buttons to reach subcategories and find the content they desire. It is necessary to design a hierarchy that helps the users find what they are searching for, a so-called information hierarchy. But the designer of a hypothetical site encounters a problem here. Categories and labels are not interpreted by every person in the same way. This leads to users looking for content in the wrong places. In the worst case, this is annoying enough for them to leave the site without having found the content they desired in the first place. An information hierarchy, like everything linked to a user interface, is subject of testing in human-computer interaction. But how to test if the categories we structured our site with are indeed "good" categories? For this purpose, the praxis of Tree Testing was created.

In Tree Testing, users navigate an information hierarchy top-down, reaching child nodes and deciding in each step which category they want to navigate to. If the path they take through the tree leads to the desired content the navigation is considered a success. But if the chosen path does not lead to the content this is considered important feedback for the creators. This is because the user was sure to find the content in a place unexpected by the creator. Tree Testing is efficient because it cuts out distractions in an UI that might cause the user to leave the correct path. It instead focuses only on language and if the categorization of content made by creators actually matches what a typical user might expect. There are actually a few known methods of doing Tree Testing. Users could categorize the content written on cards to see if they choose according to the current design, this would then be filmed and evaluated. But for efficiency reasons, it is preferable to do such studies on devices or even online nowadays. And of course, there are commercial tools that achieve this, but never before has there been an open-source tool. That was until Ajdin Mehic created the first open-source alternative, the Angular Web App TreeTest.

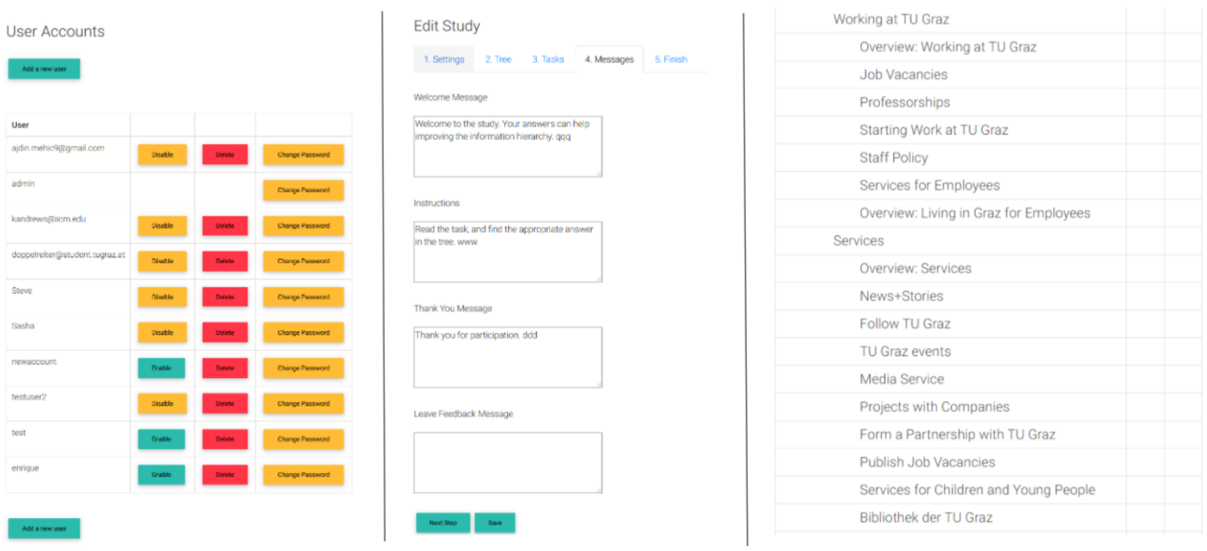
1.2 The Open-Source Web Appliation TreeTest by Ajdin Mehic

TreeTest is an open-source web application for tree testing. An example of a survey can be seen in figure 1.1 It uses Angular 7 as a frontend framework, Node as a server, Express.js as a Node framework and MongoDB as a database. These technologies are part of the so-called MEAN stack, a JavaScript development solution. There are three different user roles in TreeTest. The Admin can add and delete users, change user passwords, and enable study access. There is only one predefined admin account. The Study Owner creates and runs studies, and of course, there is the Participant, the role of the user that takes part in tree testing studies. As mentioned, one main advantage of TreeTest is that studies can be shared through a link and be done online, which is much more practical than other approaches. TreeTest features multiple interface components, as seen in figure 1.2, which leads to the question if the software can be improved by a design system.


Find information about master's programme.

- TU Graz
- Studying and Teaching
  - Focus on Studying and Teaching
  - Degree and Certificate Programmes
    - Overview: Degree and Certificate Programmes
    - Bachelor's Degree Programmes
    - **Master's Degree Programmes**
    - Teacher Training Programme
    - Doctoral Programmes
    - Continuing Education
    - Massive Open Online Courses
  - Studying at TU Graz
  - International: Studying and Teaching
  - Teaching at TU Graz
- Research

Figure 1.1: A Simple Example of TreeTest by Ajdin Mehic.


User Accounts

User			
ajdin.mehic9@gmail.com	Disable	Delete	Change Password
admin			Change Password
kandrews@iicm.edu	Disable	Delete	Change Password
doppelreiter@student.tugraz.at	Disable	Delete	Change Password
Steve	Disable	Delete	Change Password
Sasha	Disable	Delete	Change Password
newaccount	Enable	Delete	Change Password
testuser2	Disable	Delete	Change Password
test	Enable	Delete	Change Password
enrique	Enable	Delete	Change Password

Edit Study

Welcome Message

Welcome to the study. Your answers can help improving the information hierarchy. qqq

Instructions

Read the task, and find the appropriate answer in the tree. www

Thank You Message

Thank you for participation. ddd

Leave Feedback Message

- Working at TU Graz
  - Overview: Working at TU Graz
  - Job Vacancies
  - Professorships
  - Starting Work at TU Graz
  - Staff Policy
  - Services for Employees
  - Overview: Living in Graz for Employees
- Services
  - Overview: Services
  - News+Stories
  - Follow TU Graz
  - TU Graz events
  - Media Service
  - Projects with Companies
  - Form a Partnership with TU Graz
  - Publish Job Vacancies
  - Services for Children and Young People
  - Bibliothek der TU Graz

Figure 1.2: Some of the Frequently Used Components of TreeTest.

Chapter 2

Modular Building

2.1 Introduction

When developing software, each new element brings complexity and with more complexity, it becomes more and more complicated for developers. “Without a unified Language, all products drift toward inconsistency”, Yesenia Perez-Cruz writes in her Book “Expressive Design Systems”. That means, the larger a project becomes, the more inconsistent and complicated the workflow becomes too if you don’t use some form of modular building. Perez-Cruz says that teams don’t choose themselves to be inconsistent, it is a natural process when a huge number of developers are working on one project. Another problem is the formation of smaller teams within a software department. Many of them may form around specific areas and the development process of different products (e.g. website, apps, etc.) begin to split up even more. [Perez-Cruz 2019]

One possible solution for this issue is the usage of modular building. This means, that companies try to design and build with reusable components in different forms. They want to unify the development process across multiple teams. This should be a benefit for the developers and for the customers using the final products. To achieve this kind of design unification there are multiple methods which can be applied. [Perez-Cruz 2019]

- Pattern library: A pattern library is basically a compilation of reusable user interface components, called “building blocks” that are shared within the company. All developers from the different teams should be able to access them to use in their individual product development process.
- Style guide: A style guide is a textual documentation of how products should be developed concerning design, functions or basic program logic. Such a style guide is usually text-based and available as a document on a website or as a pdf-file.
- Design system: A design system is basically a combination of a pattern library and a style guide. On the one hand, it provides a unified library of building blocks. On the other hand, it describes these components in detail in a connected documentation. An example of such a design system is Storybook.js, which will be explained in the next chapter.

2.2 Basic Principles

According to Yesenia Perez-Cruz, such design systems do have some basic principles [Perez-Cruz 2019]:

- Design systems are intentional. Which means that you should have a strict plan on how to approach the development process.
- Your team is already working in systems. Meaning, that your team needs to get the concept clear what it means to use a design system. Maybe they already stick to some form of documentation?

- Strong systems are collaborative. Design systems must be shared across all teams and should be a benefit for all developers.
- The human aspects of your system are more complicated than tooling. This means, that many challenges will come up within the development team and not within a specific software. Creating a unified language for all your developers can be more important than the tool itself.

2.3 Advantages and Challenges

After defining what a design system is, a next step is to show, why design systems like Storybook can be a huge benefit. In short: What are the main advantages of using a design system within your development team? According to Yesenia Perez-Cruz, there are the following advantages [Perez-Cruz 2019]:

- Faster builds, through reusable components and shared rationale.
- Better products, through more cohesive user experiences and a consistent design language.
- Improved maintenance and scalability, through the reduction of design and technical debt.
- Stronger focus for product teams, through tackling common problems so teams can concentrate on solving user needs.

These aspects show, that design systems can bring many advantages. However, using a design system can also bring some new challenges. Yesenia Perez-Cruz describes them as following [Perez-Cruz 2019]:

- Rigid systems that stifle creativity.
- Monotonous systems that lead to generic, cookie-cutter experiences.
- Overly specific systems that can't be adapted to enough use cases.
- Complicated, unexplained, and unsupported systems that lead to fragmented user experiences.

As a conclusion could be said, that design systems can be a very powerful tool for improving your development process. Especially on a larger scale with multiple teams working on different products, a design system can help to deliver products faster, more stable and in a unified design. However, when using this form of modular building, you must know exactly where the challenges and base principles are to gain the most value out of it.

Chapter 3

The Design System Storybook.js

3.1 Introduction

3.2 How to use Storybook

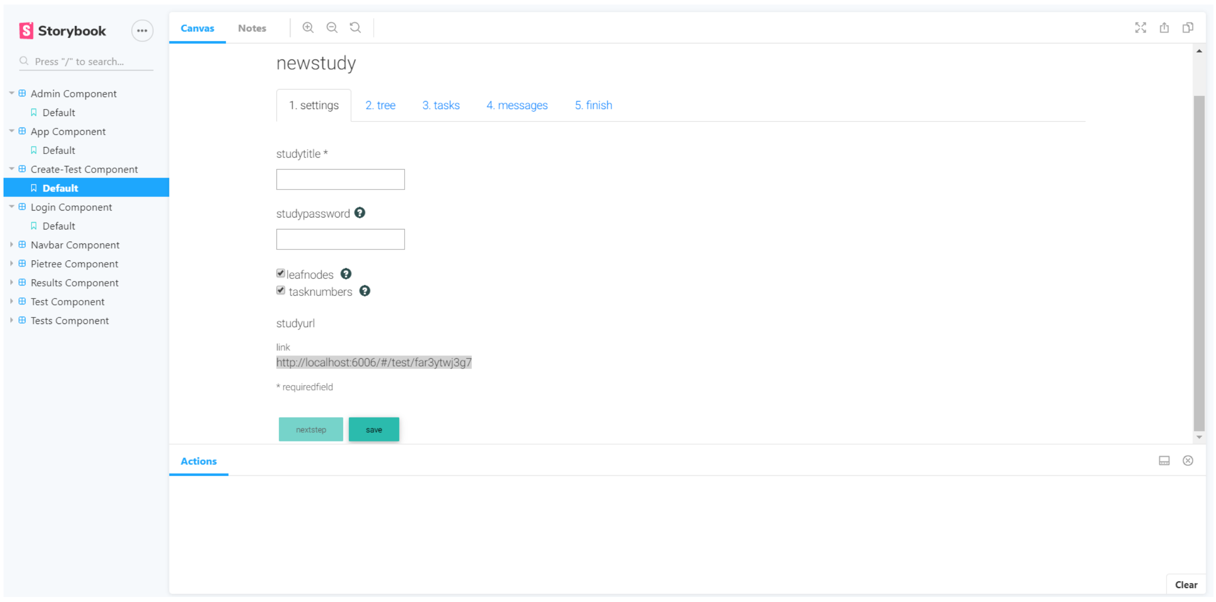
After explaining in general, what a design system is and how it works, this section will focus on giving you a short introduction to Storybook.js. It is an open-source tool for UI development, and it helps you creating modular user interfaces. Supporting different frameworks like React, Angular, Vue and more, it offers you a selection of useful functions to create, maintain and share your building blocks within your team. You can also think of it as a playground, where you can test out new options for your interface, before adding it to the real application. The main purpose of Storybook is to develop components in isolation. This should give you more control, reusability, organization and efficiency. A typical hierarchy in Storybook can be seen in 3.1. On the left side is the hierarchy which contains the different building blocks from the main app TreeTest. The current story is rendered in the Canvas Window.

Adding Storybook to a project is easy. It can be achieved by calling the command “npx -p @storybook/cli sb init” within the root directory of your project. This creates the complete structure which is needed for Storybook. To run the application with Storybook, the command “npm run storybook” can be used. The application is now reachable at "http://localhost:6006/”.

As a next step, it should be defined, what a so-called “story” is. A story is one specific state of one component of your application. It contains basic component information and is used to organize these building blocks or to cover edge cases. Each story is written in an own file where all the classes and dependencies needed, must be included. It is also possible to add different states for each component.

3.3 Writing Stories

Stories are a way of adding the information necessary to classify different visual states of a user interface component. There is a story file per class, and the application loads the stories from there. Of course, all necessary includes must be taken care of. But in the story files (cleanly separated from the main application code), the developer can add as many stories as wished. Each story then shows up in the Storybook application. To write a story one must create a story file (only files with the extension .stories are recognized) and use the storiesOfApi. This is the API to match visual components to the data that forms their different stories. It is worth to mention that currently (status from 2.2.2020) the API is already deprecated according to the development team as they are working on a new solution that will be used in Storybook in the future. However, until the current day, the storiesOf Api is the most widely used one that is present in most of the applications using Storybook up until now. With the use of the API, the developer adds all the necessary stories. Technically, a story is a function that returns something that can be rendered to screen. How one uses stories depends on the application and used frameworks. A very basic example of how stories work can be seen in figures 3.2 and 3.3.



newstudy

1. settings 2. tree 3. tasks 4. messages 5. finish

studytitle *

studypassword

☑ leafnodes
☑ tasknumbers

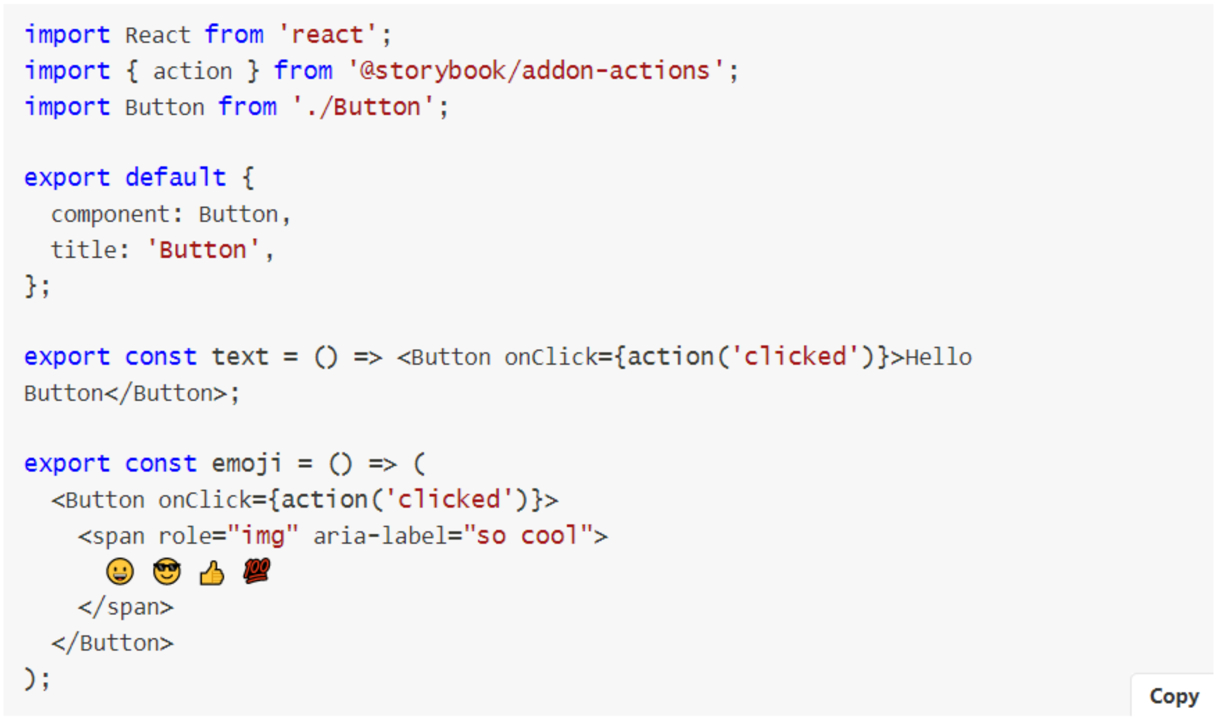
studyurl

link
http://localhost:6006/#/test/far3ytwj3g7

* requiredfield

nextstep save

Figure 3.1: Loading TreeTest Components into Storybook.



```
import React from 'react';
import { action } from '@storybook/addon-actions';
import Button from './Button';

export default {
  component: Button,
  title: 'Button',
};

export const text = () => <Button onClick={action('clicked')}>Hello
Button</Button>;

export const emoji = () => (
  <Button onClick={action('clicked')}>
    <span role="img" aria-label="so cool">
      😀 😎 👍 💯
    </span>
  </Button>
);
```

Figure 3.2: A Simple Story with Storybook.

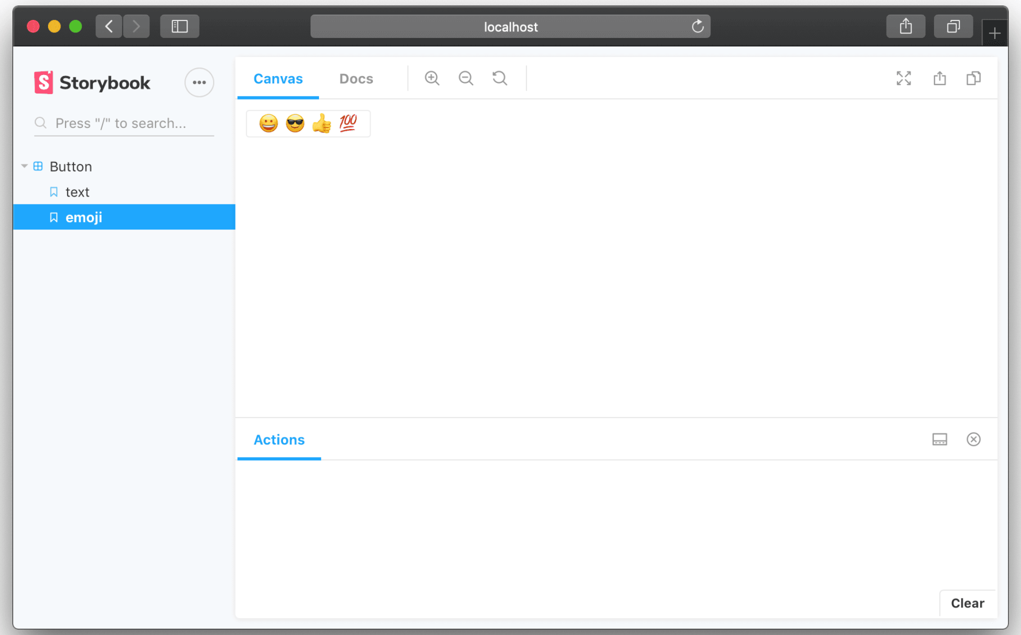


Figure 3.3: The Outcome of the Simple Story shown in 3.2.

3.4 Advantages of Storybook as a Design System

Storybook comes with all the advantages and challenges that were addressed in the last chapter. However, it also offers some exclusive features. A huge advantage is the support for many different frameworks like Angular, React, Vue or more. Most of the other open-source systems like Atellier [Sprinklr 2016], react cosmos [Skidding 2014] or react style guide generator [Pocotan001 2015] are specialized on one framework. However, the support of the different frameworks in Storybook is not equally good, but this aspect will be discussed later in the issues chapter.

Another advantage of Storybook itself is a large library of addons and extensions which can be integrated into the application. There are addons like the "Knobs addon", which will be introduced later in this paper. Storybook also delivers a safe area for testing different components and allows you to use markdown for documentation purposes. If it is integrated right from the start of a project, it can be a powerful tool for comparing designs, showing different states and aiding at presentations for the project's stakeholders. Overall, Storybook is especially recommended for React and delivers many functions to improve the development process. The main issues and conclusions for Storybook as a design system for TreeTest will be covered in a separate chapter.

Chapter 4

Knobs Addon

This chapter will take a look at the Knobs Addon for Storybook, which allows change of the properties of the component under testing dynamically using the Storybook UI. This feature is useful to show possible aspects for the UI to the investors or development team during meetings or demonstrations.

4.1 What is Knobs Addon

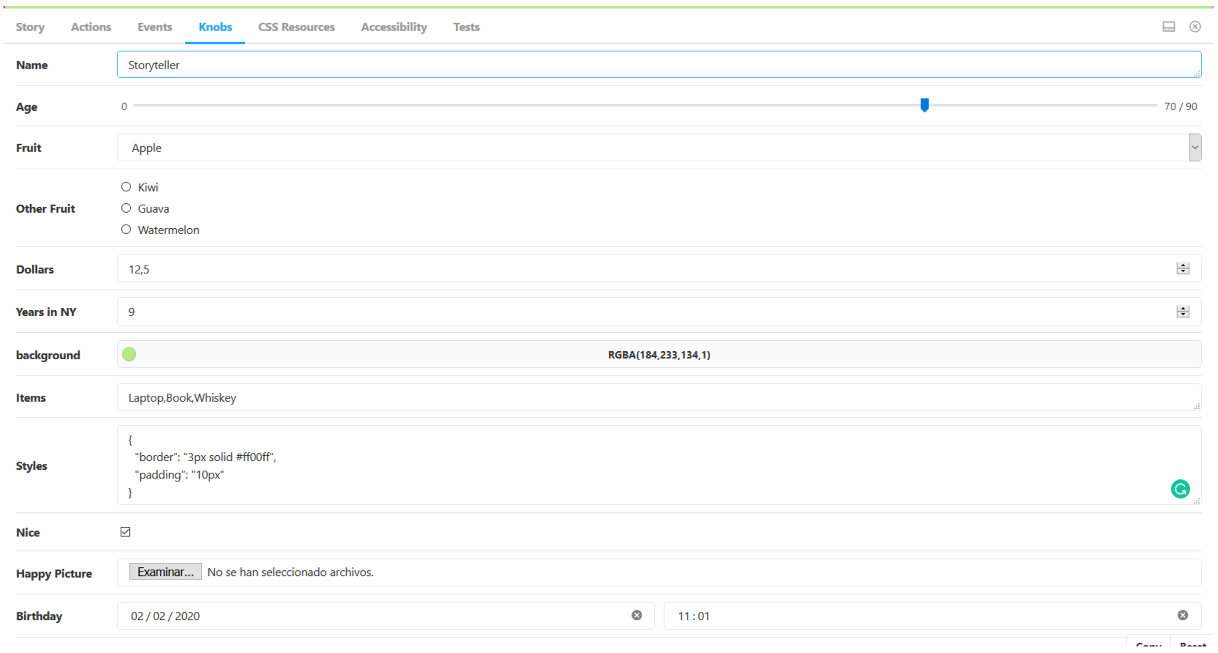
The Knobs Addon is one of the multiple addons available to add functionalities to Storybook, and it is compatible with all frameworks supported by Storybook. The addon is also part of the development of Storybook, but it is not installed together with the main package.

This addon works by defining a knobs object in the story and passing them as a property to the component. This object is then represented in the Storybook interface, and modifying its value changes the value in the visualized component, using basic string binding. This means that only string properties can be changed in the component, like the text of a section or the colour property in certain style expression.

There are different types of knobs objects, which changes how the knob value is visualized and selected in the Storybook UI:

- Text: Standard HTML text input from the user.
- Boolean: Gets a boolean value from the user.
- Number: Number input, can be standard or a range slider format.
- Color: Displays a colour picker and sends the HEX string of the colour selected.
- Object: Gets a JSON object representation.
- Array: Obtains an array of strings from the user.
- Options: Offers different options of the value in different forms: radio, select, multi-select, checkbox...
- Date: Gets a date input from the user.

All these objects get as parameters a name that shows up in the Storybook UI, a default value and optionally a group ID, to group them in tabs in the UI.



Name	Storyteller
Age	0 — 70 / 90
Fruit	Apple
Other Fruit	○ Kiwi ○ Guava ○ Watermelon
Dollars	12,5
Years in NY	9
background	RGBA(184,233,134,1)
Items	Laptop,Book,Whiskey
Styles	{ "border": "3px solid #ff00ff", "padding": "10px" }
Nice	✓
Happy Picture	Examinar... No se han seleccionado archivos.
Birthday	02 / 02 / 2020 11 : 01

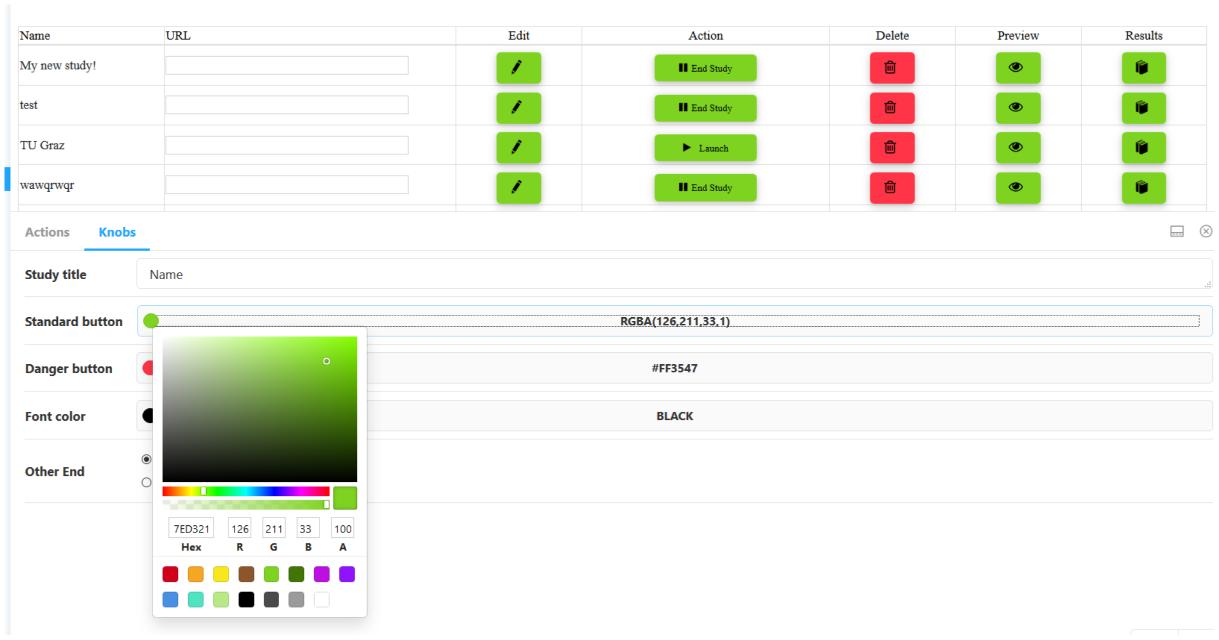
Figure 4.1: Different Knobs Types and Aspect in the Storybook UI.

4.2 Knobs Addon in TreeTest

The use of knobs in TreeTest could not be directly implemented due to unsolved dependencies and the need to update the version of some tools, like Typescript. However, a small demo was implemented taking the generated HTML and CSS from TreeTest and some knob objects were added to change the colour of buttons. This way, the implementation of some of the properties of the addon could be tested in the target application.

To make this work, a new component was created in a Storybook demo project with the Knobs addon installed and all dependencies up to date. The component used the HTML and CSS from the tests table generated by TreeTest as a template for the component, and passing the properties of the component to the template using *ngStyle* for the colours. Then, the knobs are defined in a story and passed to the component, which embedded them in the HTML.

For this test, three colour knobs, one text knob and one options radio knob were defined. The colour knobs controlled the colour of standard buttons, danger buttons like *remove* and the font colour of the buttons. The text knob changes the name of a column of the table, and the radio offers two options of the label of the ending study button.



Name	URL	Edit	Action	Delete	Preview	Results
My new study!			End Study			
test			End Study			
TU Graz			Launch			
wawqrwqr			End Study			

Actions Knobs

Study title	Name
Standard button	RGBA(126,211,33,1)
Danger button	#FF3547
Font color	BLACK
Other End	

Hex	R	G	B	A
7ED321	126	211	33	100

Figure 4.2: The Knobs Demo for TreeTest, Showing the Interface for the Color Knob.

Chapter 5

The Issues of Storybook and TreeTest

Some issues are creating obstacles for a clean and quick integration of Storybook in TreeTest. This section will examine and try to give solutions to them.

5.1 Legacy Projects

Storybooks support for Angular is still in development. At the moment, React is the main development branch. While most features work under Angular, they need more lines of code than in React and are not as optimized.

Storybook is also hard to integrate well in already existing projects. A lot of changes must be made for every element to not only load, but also support the dynamic features of Storybook. Every property needs to be made visible to the story, and in the case of TreeTest a lot of components would need to be split up to allow more granular stories and more control over their visual design.

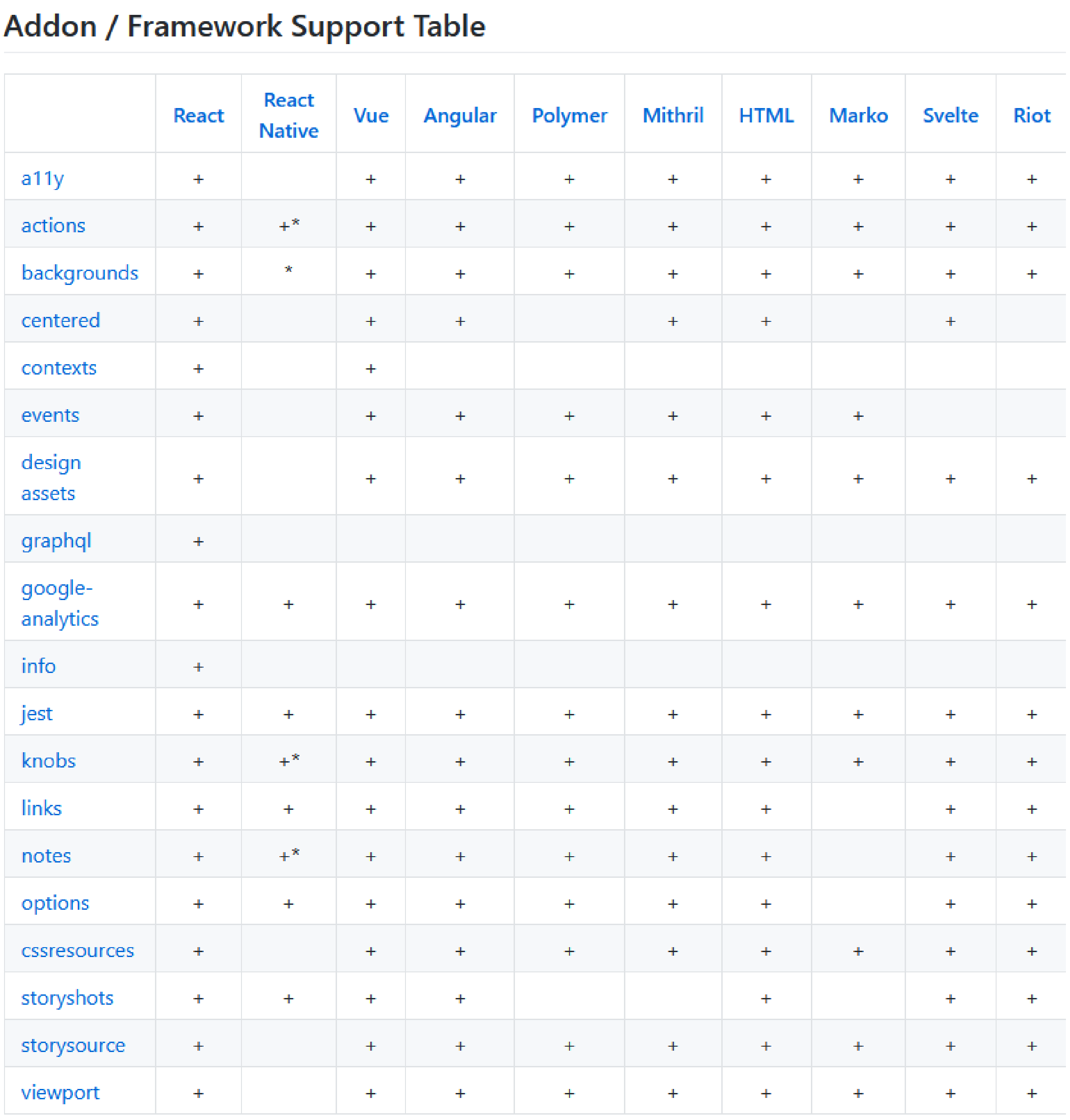
5.2 Knobs Addon

There are also problems with the Knobs addon. The addon uses basic string binding to bind properties to a knob. This means that every property of an element that should be able to change needs to be manually made visible for the addon. This is done by string binding a property to its class and then adding an individual handler to the respective story. Again, this process would take less lines of code and fewer changes in React and JSX, for which the addon seems to be designed for initially.

5.3 Exernal Templates

Another problem is that TreeTest uses global external templates for CSS and Bootstrap. They are loaded at runtime from a Cloudflare server and are hard to make changes to. To properly load the external CSS, it is necessary to use a workaround to inject the code into a global story a startup. This means that the CSS of stories is not isolated and that changes made by for example the Knobs addon get instantly overwritten by the global template. Instead, the CSS would need to be made available locally so that Storybook can create its own instances.

Another solution could be to create dynamic contexts. There is a Contexts addon for exactly this problem, but it is not compatible with Angular yet 5.1.



Addon / Framework Support Table

	React	React Native	Vue	Angular	Polymer	Mithril	HTML	Marko	Svelte	Riot
a11y	+		+	+	+	+	+	+	+	+
actions	+	+*	+	+	+	+	+	+	+	+
backgrounds	+	*	+	+	+	+	+	+	+	+
centered	+		+	+		+	+		+	
contexts	+		+							
events	+		+	+	+	+	+	+		
design assets	+		+	+	+	+	+	+	+	+
graphql	+									
google-analytics	+	+	+	+	+	+	+	+	+	+
info	+									
jest	+	+	+	+	+	+	+	+	+	+
knobs	+	+*	+	+	+	+	+	+	+	+
links	+	+	+	+	+	+	+		+	+
notes	+	+*	+	+	+	+	+		+	+
options	+	+	+	+	+	+	+		+	+
cssresources	+		+	+	+	+	+	+	+	+
storyshots	+	+	+	+			+		+	+
storysource	+		+	+	+	+	+	+	+	+
viewport	+		+	+	+	+	+	+	+	+

Figure 5.1: Addon Compatibility of Different Frameworks Supported by Storybook [Shilman 2019].

Chapter 6

Conclusion

Storybook could be a very useful tool for TreeTest. Basically all the benefits discussed in this paper would apply to the project. It could especially aid in the expansion of features and future UI components.

However, Storybook should have been integrated and planned for from the start. To integrate Storybook now into TreeTest, a lot of changes and reworks are necessary which leads to the question if the integration would be worth the time and effort. This largely depends on what the future of TreeTest is supposed to be and if the Software should be able to adapt to and speed up further development.

The estimated time for an integration into TreeTest with all features working would be comparable to a full UI rewrite. In that case, it also may be worth it to switch to React and a JSX interface in the process. On the other hand, it is worth it to consider an integration of Storybook in future projects from the start. There is not that much additional work if implemented in parallel with the UI, and it is also easy to plan for. However, for very small projects or projects with no plans for expansion, the benefits of Storybook would be too small to consider it. All other projects would benefit more than enough to go forward with a full integration.

Bibliography

Perez-Cruz, Yesenia [2019]. *Expressive Design Systems*. A Book Apart, 21 Nov 2019. ISBN 978-1-937557-84-3 (cited on pages 3–4).

Pocotan001 [2015]. *react styleguide generator*. URL. 2015. `https://github.com/pocotan001/react-styleguide-generator/graphs/contributors` (cited on page 7).

Shilman [2019]. *Addon / Framework Support Table*. URL. 2019. `https://github.com/storybookjs/storybook/blob/master/ADDONS_SUPPORT.md` (cited on page 14).

Skidding [2014]. *react cosmos*. URL. 2014. `https://github.com/react-cosmos/react-cosmos` (cited on page 7).

Sprinklr [2016]. *Atellier*. URL. 2016. `https://scup.github.io/atellier/` (cited on page 7).