Universitetet i Bergen

Master's Thesis

A novel collaboration platform: Structuring books as networks

Supervisor

Rogardt Heldal

Author:

Ole Eskild Steensen

June 2018

Abstract

Ever since Gutenberg introduced the art of printing in Europe in the mid 1400s, the way we consume and author books has remained somewhat static. In spite of several technological advances since that time, the very structure of books and the writing process has not been subject to change. Neither have the reading experience. This thesis explores a new way to read and write novels by applying a network structure to books and allowing collaboration on a big scale by leveraging the vast number of people connected to the internet. The thesis uncovers challenges in applying such a structure and proposes solutions to a number of them. Additionally a prototype under the name WebNovel is developed alongside the thesis.

Acknowledgements

First, I want to express my gratitude towards my supervisor Rogardt Heldal as well as John Grieg, who conceived of the idea from which this thesis is based on, for being immensely helpful throughout the process. Without the two of them, this thesis would not exist. Their patience, guidance and fruitful discussions has proven invaluable.

I'd also like to extend my thanks to the informatics department at both UiB and HVL for allowing me to be included in a stimulating environment, both academically and socially. They have made my years there highly enjoyable.

I'm immensly greatful to my girlfriend, Ida Sofie, for her help and patience in the weeks before deadline. Lastly, I'd like to extend my gratitude to my family and friend, for their encouragement, help and support throughout the entire process.

List of Figures

List of Tables

Contents

1 Introduction

1.1 Motivation

The first wave of digitalization managed to digitalize data like text, audio and images. The second wave digitalized work processes, a process we are currently in the middle of. Companies like Uber and Lyft changed an entire industry in a matter of years. AirBnB has allowed homeowners to share their house with travellers. However, one process that has yet to be changed in a radical way by technology is the traditional novel. We were curious as to what would happen if we could harness the power of large crowds to create novels where the number of authors and contributors is beyond what is currently feasible. Will completely new genres emerge? Will it change the way we read books? This thesis will not try to answer these questions. Rather, our goal is to propose solutions on how to create a platform allowing this kind of collaboration, thus creating an opportunity for the former questions to be answered. This thesis takes a closer look at the challenges in creating such a platform, and proposes solutions to a number of these. A prototype is developed alongside, giving us the possibility to try the solutions in practice.

1.2 Research objectives and purpose

In the last few years we have seen several examples of digital tools emerging that allows hundreds or thousand of people to collaborate and share content on an unprecedented scale. There are success stories, like Wikipedia, that has cracked the code on how to collaborate on objective texts. Our goal, big scale novel collaboration, seemingly face the same challenges that Wikipedia needed to overcome. However it presents a very different set of problems. Writing articles on Wikipedia is an iterative process that improves on a single piece of (mostly) objective text. A novel does not provide much leniency for objectivity. A chapter loved by Alice might not be appreciated as much by Bob. Should the chapter change to what makes Bob satisfied, or should it stay the same so that Alice remains happy? With this project we want to understand a particular strategy for creative collaborative writing. The purpose is to explore what happens when one changes the very structure of novels by imposing a **network structure** to the chapters instead of the more traditional linear and fixed structure. Furthermore it should be in the context of complete strangers collaborating in creating this network. We aim to identify challenges linked to using this technique and propose solutions to a subset of them. Our objective can be summed up by the following

research question:

What challenges arise upon developing a digital collaboration platform for writing novels, and how should we approach them?

1.3 Related works

While investigating past literature we found that little research had been conducted about collaboration between strangers on creative, fictional texts. We tried several search terms, all of which can be found in Appendix E. As we were unable to find any previous research done on the topic this thesis is breaking new ground on a previously unexplored subject.

Similarly there does not exist any products with functionality similar to what we are proposing. There are some that offer part of the functionality. Twine [2] is a software application allowing users to write non-linear stories in the form of a "choose-your-own-adventure" story. Although the structuring of books in a network is similar to what we are proposing it is not built to offer collaborative functionalities. Additionally it does not offer a publishing platform: If one would like to publish work created in Twine, one would need to export it and self publish. As our prototype is focusing on collaboration between strangers, it must include a publishing platform so that others who want to contribute can browse the content of others.

Another tool implementing part of our proposal is WattPad [3]. This is a writing community in the form of a website. Here users can share their stories with other users on the platform and receive feedback on their creations. However the novels are structured in the traditional way. Moreover they do not offer or encourage ways of collaboration.

A third comparison that comes to mind are tools like Google Docs [4]. Although both Docs and our proposed platform offer text-collaboration functionality, there are some major differences. The texts produced with Google Docs are not necessarily created with the intention to publish it. It also does not offer any publishing platform, preventing users from "stumbling" upon the content. Moreover, the collaboration is usually limited to a single linear piece of text. The tool does not allow for any change to the very structure of the content in the way we are proposing. Lastly the problems that these tools are meant to solve are completely different from each other. While Docs aims to be a tool allowing teams to collaborate on any type of textual content, our platform's focus is for people to collaborate

on fictional novels.

As there does not exist a platform with the equivalent functionality we deem the contribution of our prototype and research to be of value.

1.4 Target audience

The audience of this thesis is mainly people involved in computer science, as it gets technical and detailed about the technologies used. However it may also have some value to an audience that are not necessarily software developers due to its exploration of some conceptual subjects that most people should be comfortable to follow along with. It is also worth noting that due to the implementation of the prototype we have the potential to also reach an audience outside of the academic bubble. That is, people interested in creating and consume content made possible by our platform.

1.5 Thesis structure and outline

Apart from this introduction, the thesis is structured as follows:
Section 2 presents the methodology we used to perform our research. In Section 3 we explain concepts and technologies used to create the platform, that the reader needs to be comfortable with to properly understand the implementation of the application. Section 4 describes and justifies the concept of structuring books as networks. The next section, Section 5, outlines the questions we want to answer in this thesis. These are questions that were not set from the start, but emerged during the development process. In Section 6 we describe an overview over how the application was designed, without diving into implementation details. Section 7 discusses some legal aspects of the platform, such as the licensing of texts. Section 8 summarizes how we solved the questions posed in Section 5 by referring to the relevant functionality described in sections 4 and 6. In Section 9 we discuss the feedback received from surveys and user tests. Lastly, Section 10 summarizes our findings and proposes what further work may be done.

Appendix A contains instructions as to how to use the application. Appendix B explains how a developer may set up a local development environment for the prototype. In Appendix C we discuss the architecture and implementation details of the application. In Appendix D we share the unedited answers from the user tests and surveys we have conducted. Lastly,

Appendix E presents a list of what search terms we have used in our attempt to locate any previous research done on the subject.

2 Methodology

Our research uses a very practical approach, focusing mainly on the development of the application. More specifically, we employ the design science research methodology (DSMR) as presented in Peffers et. al [1]. According to their definition this methodology has six distinct activities or process iterations that are illustrated in Figure 1. This section describes how we carried out these activities.

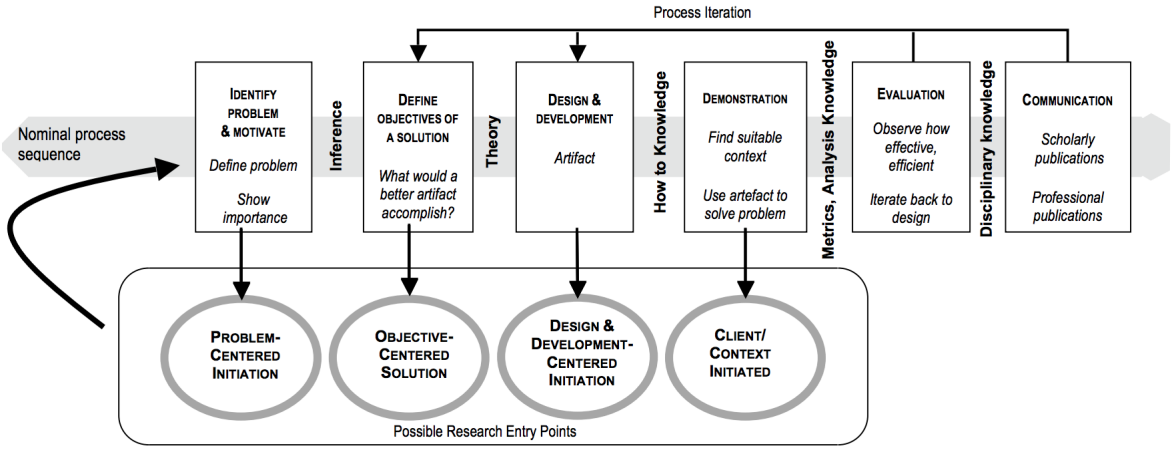

Figure 1: Design Science Research Process Model (from Peffers et. al [1])

2.1 Research entry point

As can be seen in Figure 1 Peffers et. al [1] define several possible research entry points. (Consequently it follows that there are no set order of activities.) In our case the point of entry was client-initiated. An individual presented a rough sketch of a novel-collaboration platform that he wanted to be developed and released to the public. The idea of using a network as the data structure of each book was also part of the original proposal.

2.2 Problem identification & motivation

This activity describes the process of "[defining] the specific research problem and justify the value of a solution" [1]. During this process the researchers describes the problem that they

aim to solve and offer motivation for why this is worth investigating. We have summarised how we approached this process in the next paragraph. For a more detailed description see Section 1.

Although the internet has provided us with the possibility to collaborate with people all over the globe, the process of writing a novel largely remains the same. Moreover the format of a novel is mostly linear, despite this not necessarily being the only way to structure novels. With this project we wanted to allow the use of an alternative method of authoring novels by both enabling hundreds of people to collaborate on the same book and propose a new way to structure these books.

2.3 Objectives of the solution

During the "Define the objectives for a solution" phase we aim to present a solution to the identified problems. Our "objectives of the solution" is described in the next paragraph.

We wanted to create a working software platform, implementing our proposal for a truly collaborative writing process between multiple parties. Our aim was to produce a fully functioning artifact that could be published and released to the public. The entire project should additionally be released as open source, creating a truly collaborative platform where not only the content is created in a collaborative manner, but also the software itself.

2.4 Design & development

This activity is where the actual artifact is created, be it a model, method or an instantiation. In our case the artifact is the prototype developed under the name **WebNovel**. Most of our effort has been focused on this activity, working towards our goal to produce a working artifact alongside the thesis. We borrowed concepts from agile software development, using iterative and incremental development methods. This part closely ties in with the evaluation activity in DSMR. After each development cycle, we improved on the design and architecture based on the feedback, results and conclusions we got from our evaluation methods as described in Section 2.6. The basic functionality and concepts were already in place from the start, however some requirements changed over time as the idea and product matured.

As explained further in Section 2.6 one of our key development techniques was to arrange

frequent meetings to discuss the current situation and the road ahead. During these meetings requirements were subject to change, and new features were suggested. Ahead of each meeting a new working version of the prototype was ready and could easily be evaluated by the participants. This enabled us to develop the software in an iterative, incremental and evolutionary way.

2.5 Demonstration

Demonstration is described as the activity where the researchers "Demonstrate the use of the artifact to solve one or more instances of the problem." [1]. As the scope of this thesis is limited in time the demonstration process has been done in a somewhat informal fashion. The prototype has been demonstrated to acquaintances of the author, both informally throughout the development and more formally during user tests as talked about in Section 9. This in turn has provided us with iterative feedback during the continued development of the prototype. Additionally the software has been published in various Facebook groups that we thought might have an interest in such a product.

2.6 Evaluation

In the evaluation process the researchers are supposed to "Observe and measure how well the artifact supports a solution to the problem." [1].

The first established evaluation method in this project was biweekly discussions between the supervisor, the student and the person who conceived of the original idea. These mostly functioned as brainstorming sessions, where both improvements to existing features as well as suggestions to new ones were discussed. The goal of this exercise was to gain insights about the product by allowing us to participate in an open forum, discussing its shortcomings and addressing problems that arose under its continued development.

Implementing the prototype was also a big part of the evaluation process. Implementing software is key to noticing shortcomings as well as impossibilities. Upon completion, this project should ideally have produced a working web-application implementing most of the functionality described in this paper. This not only provides us with valuable information about the development process itself, but also gives us the opportunity to receive feedback from users interacting with a concrete and tangible product. This helps uncover caveats that

might not be recognized due to a developer's tunnel vision. Additionally it paves the way for improvement suggestions by people not immersed in the project.

Lastly we conducted surveys and user tests as a form of evaluation. This is discussed on further detail in Section 9.

2.7 Communication

The last activity, communication, references how the work done during the research process should be communicated to the relevant audiences. The results and findings in this thesis will be published in the Bergen Open Research Archive.

3 Background

This chapter gives the reader an introduction into what technologies and concepts the reader should be familiar with upon reading this thesis. The technologies used to create the implementation of our proposed platform are introduced in addition to any relevant concepts. If the reader is comfortable with the concepts and technologies presented here, they may skip this section.

3.1 Tree & network structures

The data structures **tree** and **network** will be mentioned a number of times in this thesis. The reader should be familiar with these concepts to properly understand what is presented. What follows is a short introduction to what a tree and a network is.

A network, synonymous with the word **graph** for our intents and purposes, is a set of vertices or nodes that have some relation to one another. These relations are often visualized and referred to as **edges**, as illustrated in figures 2 and 3. What the nodes and the relation between them represent is arbitrary. Generally there are two distinct kinds of networks: directed and undirected. An undirected network contain relations that is true for both nodes on either end of the edge. The edge could, for example, represent friendships between a set of people, or which people has collaborated on science papers in a set of published authors. A directed network is often visualized by drawing arrows on the relevant

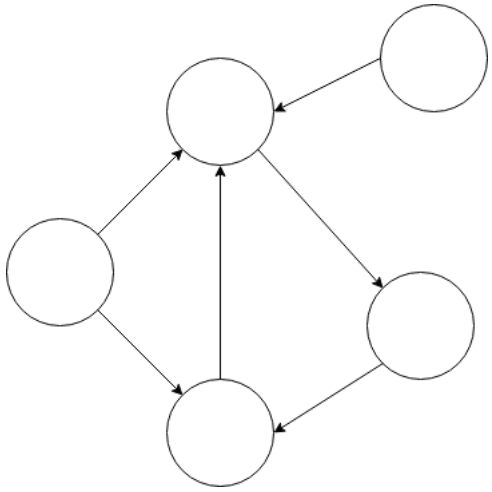

Figure 2: Example of a directed network

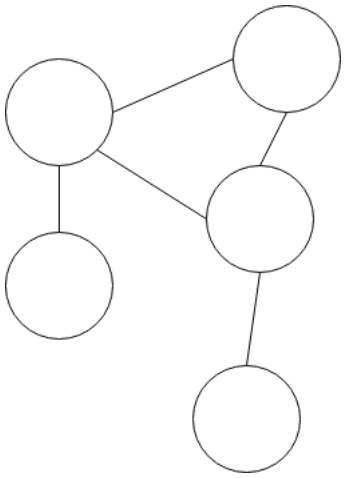

Figure 3: Example of an undirected network

end of the edges. A network is directed if the relations between the nodes are valid one way but not necessarily the other. A natural example would be the World Wide Web, where the nodes would represent web pages and an edge from one node to another would mean that the first node has a hyperlink pointing to the second one.

The tree data structure is similar to the network structure, however it has some constraints not found in a network. While in a network one can create a relation between any two nodes, the tree structure only allows a relation between two nodes when certain properties are true. Also it can be said to be a hierarchical structure, meaning the nodes are arranged in a way so that one can say that they are above, below or on the same level as one another. There are some terms one need to familiarize oneself with: **child**, **parent** and **sibling**. If a node is a parent to another node, it means that it is above that node in the structure. The node directly below that parent is what is called a child. Two nodes are siblings if they are children of the same parent. In a tree structure every node may add a child below itself, thus there can exist an arbitrary number of children to a node. However child nodes may not add another parent. A node will always have a single parent. Furthermore, one may not add a relation, or edge, between two siblings.

3.2 NoSQL

The prototype uses a NoSQL database. The reader should therefore be comfortable with the basic functionality and terminology used in this context. We are, more specifically, using the MongoDB [5] implementation of NoSQL, however the specific implementation is not

important as our explanation of the technology in this section will apply to most NoSQL database implementations.

The first concept to familiarize oneself with is **collections**. This is a collection of data that has a shared conceptual model. That is, all the data in a collection should, but is not forced to, be the same type of data. For example if one where to store a database containing the breeds of different animal species one could create a collection for each species, such as "Dogs" or "Cats".
The data contained in a collection is in the form of a number of discrete objects known as **documents**. Going back to our animal example, each document in the "Dogs" collection would contain the info of one breed. This information is stored in the form of a JSON, or JavaScript Object Notation [6], document. Querying the database is done through an API in a supported programming language. Querying for the document holding information about a specific breed in our animal example could look something like this:

```
1    db.dogs.find({"breed":"Norwegian Lundehund"})
```

More generally the syntax looks similar to this:

```
1    db.collection.find{data}
```

The last concept one should be familiar with is the no schema philosophy that NoSQL databases implement. That is, the database does not validate or force the user to have a set static schema for each collection. Thus, as system requirements change, it is easy to change the data type to fit the new needs. If one wishes for some form of validation of documents being updated or inserted into a collection one generally has two options: Manually implement validators that run before any altering in the database takes place, or use a software library that acts as a layer on top of the original APIs. If one uses the latter option, one ordinarily has the ability to define schemas that are used to validate raw data on an application level before entering the database.

3.3 HTML & CSS

HTML (Hypertext Markup Language) [7] is the standard markup language for the web. The reader should be familiar with this technology as the prototype is a web application. In short, whenever one is viewing a rendered web page in a browser, that render is based off of a HTML document. Such a document consists of building blocks called HTML elements, which are presented by **tags**. An example of such a tag is the <img> tag that display

an image. Each tag may contain a number of predefined attributes. Going back to the image example, it would be required to define a "src" attribute holding the file path to the image's location. While HTML defines how content on a page is structured it does not provide any styling functionality. The styling of webpages is done through CSS (Cascading Style Sheets) [8]. CSS allows a developer to style the HTML document however they please, defining attributes such as height, width, color, position etc.

3.4 DOM

The DOM [9], or Document Object Model, is a programming interface that structures an HTML document as a tree structure, where each node in the tree represents a part of the document. This tree can be manipulated through APIs via JavaScript. Thus, a webpage can dynamically change throughout its lifecycle by dynamically changing the DOM the web browser uses to render the webpage.

3.5 JavaScript & Typescript

As the entire codebase, both front end and back end, is written in JavaScript [10] and Typescript [11] one should be comfortable with how the two languages function. We will introduce what the reader would need to know to follow along in this thesis, starting with the differences between the two.

Typescript is a superset of JavaScript. Thus, all valid JavaScript code is also valid in Typescript. What Typescript adds is the ability to define variable types. At compile time, the Typescript transpiler will check for any wrongfully assigned variable values and output any relevant errors. As this is the only aspect that separates these two languages in any meaningful way, at least in the context of this thesis, the rest of the information presented in this section holds true for both.

JavaScript is a multi-paradigm language. It supports event-driven, functional and imperative programming styles [10]. One important aspect of the language is the asynchronous behaviour it exhibits. As JavaScript is often used in the context of the World Wide Web it needs to be able to wait for a database query to return information while simultaneously processing user events, such as a click. This is done via events-handlers and callback functions. An event is an action or activity that is recognized by an event-handler. A typical

example in the context of a web page is when a user clicks a button. Normally the button has an event-listener attached to it. When the button is clicked, the event is sent to the event-handler defined in the software by the developer. The function defined in this event-handler is then executed. The concept of a callback function is somewhat similar. It is a function passed into some method that behaves asynchronously. It is often used in conjunction with input/output behaviour. Whenever one executes a method that starts an I/O procedure it is the norm to also provide a callback function. This is a function that the method will execute at a given time. Typically it is called whenever the input or output of the relevant data is complete. Listing 1 illustrates a typical example of a method using a callback function when retrieving data from a database.

Listing 1: Example of a callback function

```
1    database.find(query, function(result){
2        console.log(result);
3    });
```

The function provided as the second parameter will be called whenever the database retrieval finishes. As it is an asynchronous function, other code may execute while this is happening.

3.6 Angular

The reader should be comfortable with Angular, as the front end of the prototype uses this technology. This section is a quick introduction into the basics of Angular.

Angular [12] is a Single Page Application (SPA) framework written in Typescript. SPA frameworks allow web pages or web apps to only load a single HTML page and then dynamically manipulate the local DOM using JavaScript. When a user navigates to another part of the application, the framework will dynamically update the page to correspond with the desired page. If any external data needs to be loaded one typically uses AJAX (Asynchronous JavaScript and XML) [13] to retrieve the desired data from a web server through an API.

The most important concept one needs to be familiar with in Angular is **components**. These are the basic building blocks of the user interface. Each page in Angular is itself a component, often containing a number of other components. A typical example of the hierarchy can be seen in Figure 4.

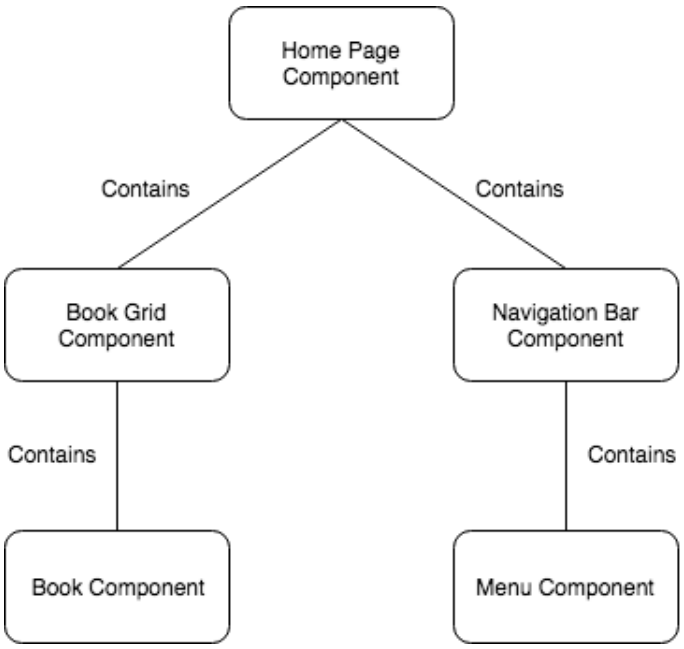

Figure 4: Example of a component tree

Each component in Angular is a Typescript class, and can thus contain functions and variables. It also contains some metadata meant for Angular. We will introduce the three most fundamental fields. These are **selector**, **template** and **style**. The first, selector, holds a custom string value that will be used when referring to and inserting the component into any other component. The insertion is done by using an HTML-tag with the tag-name corresponding to the selector value. If one were to define a component with the selector value "bullet-list", it could be inserted into HTML documents by using a <bullet−list></bullet−list> tag.
The next field, style, contains any custom CSS styling desirable for this particular component that are not defined in any global stylesheet. The last field, template, holds the markup template that the component uses. In addition to supporting ordinary HTML, the markup template extends the language to include some new features. One of these is interpolation. These are JavaScript expressions wrapped in double curly braces ({{...}}). These expressions will be evaluated at runtime and the result is converted to a string that can be displayed on the web page. Angular also automatically sets up listeners for the variables used in these expressions. Whenever a variable value changes, the changes are mirrored in the UI without needing to manually instruct the UI to update.

The components also support in- and outputs. This concept is easiest to explain by de-

scribing a concrete example. Imagine creating a component displaying a vertical list of strings. We could create one and hard-code the values into the template, or we could use input and outputs to create a reusable component. We would then define an input that is to receive an array of strings. The template is then made to loop through every string and create a line in the list containing the relevant string. We now have a functioning read-only list. However, if we also want interactivity we can take it one step further by defining an output event. One can define this output to trigger upon any event happening in the component. A natural output would be whenever one of the items in the list is clicked. The output can contain any arbitrary data, in our example this could be the index of the item being clicked.
This component can now be used by any other component that should display a list. The array of strings would then be provided by the parent component as input to the list component. The parent component also has the ability to define what happens whenever one of the items in the array is clicked. Thus we have created a reusable component using outputs and inputs allowing for dynamic content and behaviour.

An important concept in any SPA is how the navigation between pages is handled. Traditionally navigating to a new page is done by linking to another HTML document, leading the browser to issue a GET request towards the web-server, asking for the relevant file. As SPAs are just manipulating the DOM, this method becomes obsolete. However the behaviour is desirable to emulate as most users expect this behaviour when navigating a webpage.

The router module [14] is Angular's solution to this. Whenever the user navigates to a new page, by clicking on what is called a routerlink, the following happens at a high level:

1. Angular modifies the URL in the browser's URL bar
2. The DOM is manipulated to reflect the desired changes
3. Angular adds the navigation step to the browser history

These steps ensure that the navigation functionality in the browser behaves as the user expects in spite of it actually just manipulating the DOM.

The last concept one should familiarize oneself with is Angular's dependency injection system [15]. Angular encourages to use special classes called **services**. These are used as an abstraction layer to retrieve data for the application, i.e. from an API. Upon creating a

class defined as an injectable service, it can be injected into any component through the constructor. The developer does not create their own copy of the service. Rather, the service is injected in runtime. Thus, if one wants to change the service implementation to another API or maybe to a mock service used for testing, one can simply configure the relevant services to be the implementation one wants to currently use.

3.7 SVG & D3

The visualizations in the prototype uses the D3 software library. Hence, a basic understanding of how this library functions is desirable for anyone reading this thesis. Also we briefly introduce SVG as this is used in conjunction with D3.

SVG [16], or Scalable Vector Graphics, is an XML-based vector image format. SVG can be inserted into the DOM by using the <svg> tag. As the graphics are vector based it is possible to zoom indefinitely without the graphics becoming pixelated. Inside the SVG element there is a number of possible tags to put. Some of these are circle, rectangle and line. These all have assignable properties such as height, width, radius etc. Moreover, one can attach event listeners to any of these elements as they are all just DOM elements.

D3's official website describes the library as "[...]a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG, and CSS. D3's emphasis on webstandards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation." [17] Put swiftly, D3 offers functionality to make it easier to create visual representations from various data sources. For example, the library has a number of transformation functions. Depending on the function, they accept specific data structures that are transformed to a structure containing information that are useful when visualizing the data. To better understand how this works in practice we will explain how we can use D3's tree function to visualize data structured as a tree. First we need to make sure that our data is structured in the correct way. In this case it should be in the form of a JSON object, where the root node of the tree is first defined, containing whatever data one would want. The root node has a children field that is an array of new objects representing nodes. These objects, or nodes, may also contain their own children array. If our JSON file is correct we can input it into the tree function. The function then produces an array of nodes. Each of these nodes are populated with several new attributes: [18]

- parent - the parent node, or null for the root
- children - the array of child nodes, or null for leaf nodes
- depth - the depth of the node, starting at 0 for the root
- x - the computed x-coordinate of the node position
- y - the computed y-coordinate of the node position

Now we have coordinates for each node in the tree. Therefore we can use another core functionality of D3 to draw our tree. The selectAll(selector).data(array).enter() pattern. This lets us create elements in the DOM for each data point in the array. Illustrating this concept is the example in Listing 2, copied from the D3 documentation:

Listing 2: Example of code using the data(array).enter() pattern

```
1  d3.select("body").selectAll("div")
2        .data([4,8,15,16,23,42])
3     .enter().append("div")
4        .text(function(d){ return d; });
```

Assuming that the body is empty the code in Listing 2 will create six new div-elements, append them to the body and assign text content equal to the number it corresponds to in the data set as illustrated in Listing 3:

Listing 3: DOM element generated by d3

```
1  <div>4</div>
2  <div>8</div>
3  <div>15</div>
4  <div>16</div>
5  <div>23</div>
6  <div>42</div>
```

It is worth noting that the data corresponding to each div is passed as a parameter in the text function. Thus it is easy to apply the desired properties on each element. In our tree example we could use this pattern to create an SVG circle element for each node and then assign each circle the appropriate coordinates based on the data produces by the tree transformation function.

3.8 Socket.IO

Socket.IO is a software library that "enables real-time bidirectional event-based communication" [19]. Essentially this means that by using this library one is able to send an arbitrary payload to either a client or a server. When using a more traditional technology, a client usually has to ask the server for information whenever it is needed. With socket.IO the server acquires the capability of sending information to the client without the client first asking for it. The library also has reconnection logic, so that if either party disconnects for some reason, socket.IO will try to re-establish the connection.
Two concepts in socket.IO the reader should be familiar with are **rooms** and **namespaces**. It is possible to "namespace" sockets, essentially assigning different endpoint or paths. This allows the developer to separate concerns within the application without having to set up a TCP connection for every endpoint. Clients have the possibility to join any of the namespaces defined on the server, and will receive messages that are emitted in this namespace only. Within each namespace one can define arbitrary rooms or channels. Sockets can subscribe to these rooms to receive whatever messages are emitted in them. Socket.IO automatically creates a room for each socket connected to the server that is identified by a "random, unguessable, unique identifier" [20]. Thus, to send data to a single client, one can use this id. Upon emitting a message one must define what event it is part of. This is done by using a string identifying the event. The event is typically defined to be human readable text such as "notification" or "private message". The sockets receiving the events has the ability to create event handlers that fire whenever the specified event occurs.

3.9 Redis

Redis [21] is "[...]an in-memory data structure store, used as a database, cache and message broker[...]". It supports different kinds of data structures such as strings, lists, maps, sets etc. It can be used to store information that does not necessarily need to be persisted on disk such as queues or caches. However it also provides persistence capabilities through snapshots. Redis acts as a server so that any process may connect to it, and is faster than traditional databases due to it using memory as the main storage.

3.10 Node.js

The reader should be familiar with Node.js if one wishes to understand how the back end of the prototype works, as this is the runtime used. This section quickly introduces Node.js

for readers not familiar with the technology.

Node.js is a "[...] JavaScript runtime built on Chrome's V8 JavaScript engine." [22] Thus, any artifact to be run in node, is written in JavaScript. The runtime uses an event-driven model, where inputs and outputs are non-blocking. Essentially, event-driven means that when programming for Node.js, one defines what procedures should be executed upon various events. In the case of a web server this could be events such as GET and POST requests. For example one could define that upon a GET request to the server on a specified URL path, the server issues a query towards the database retrieving data which is sent back to the client that initially issued the HTTP request. Due to the input and output being non-blocking the server is able to serve several users concurrently using only one thread, because code continues to be executed while waiting for the database to return the result. The data that is returned is handled by a callback function as explained in Section 3.5.

3.11 Docker

Docker [23] is a platform that allows applications to be run in **containers**. A container is a general construct that exists outside of the context of Docker. Wikipedia summarizes it the following way: "[...] containers is a generic term for an implementation of operating system-level virtualization [...]" [24]. Put more simply, Docker is a piece of software that runs on top of some host operating system enabling apps to run isolated as if they are running on different physical hardware. In contrast to virtual machines, which is a comparable technology, the time it takes to start an application is significantly smaller, as it does not need to spin up an entirely new instance of an OS. Additionally one does not need to allocate CPU, storage or RAM for a guest OS. This translates into using less resources as one no longer needs to run virtualization of an entire OS for every app in production. Moreover Docker offers the capability of sharing bins and libraries between different apps in separate containers, allowing for even better use of resources.
The concept of linking together containers is also available. Linking is the term used to describe a connection between two separate containers. A typical example is that we have two containers in production: a web server and a database server. We would like the web server to have access to the database server to retrieve and insert data, however for added security we might want to prohibit a connection from anywhere else. To achieve this we may use the linking feature.

4 Novels as a network

The structure of the novels is our main focus in this thesis. Instead of the traditional fixed linear structure, we wanted to try something more complex. More specifically a directed graph. This, in our opinion, enables a great foundation for a platform allowing users to collaborate in writing novels on a big scale. This structure allows for several features that is key to implementing our proposal. The purpose of this chapter is to give the reader a clear picture as to why we decided on this structure and what benefits it brings when it comes to crowdwriting novels. It also briefly touches on the challenges that occur when applying this structure to novels.

4.1 Creating new paths

One of our earliest insights in the process of deciding how to structure the novels was that we wanted to provide the content creators with an option to create alternative story paths if they didn't particularly like or agree on how a story unfolded itself. This, in practice, meant that a writer would have the ability to create a fork from a chapter, where his or her version creates an entirely new direction that new readers can explore instead of the existing one. People familiar with software development will immediately recognize this structure as a tree, branching outwards and growing exponentially. This was also our first conception of the idea. Each novel was to be structured as a tree, allowing creators to add forks in the existing tree to create new story paths. The first prototype implemented this feature, and only allowed this action. We later realized, however, that this presented some constraints that we were not happy with.

4.2 Adding alternative chapters

While testing the tree structured version internally we realized we wanted some functionality that simply were not possible with the current solution. For some chapters we wanted to be able to create a single alternate chapter, without having to start an entirely new path. With a tree structure this is not possible, as it would require an edge to point back to the graph as illustrated by the dotted line in Figure 5. Thus we needed to change the entire database presentation. Before, each chapter contained a list of foreign keys which represented the children of the chapter, while simultaneously having a foreign key pointing back to the parent chapter. Changing to a network model meant that a single vertex could have multiple par-

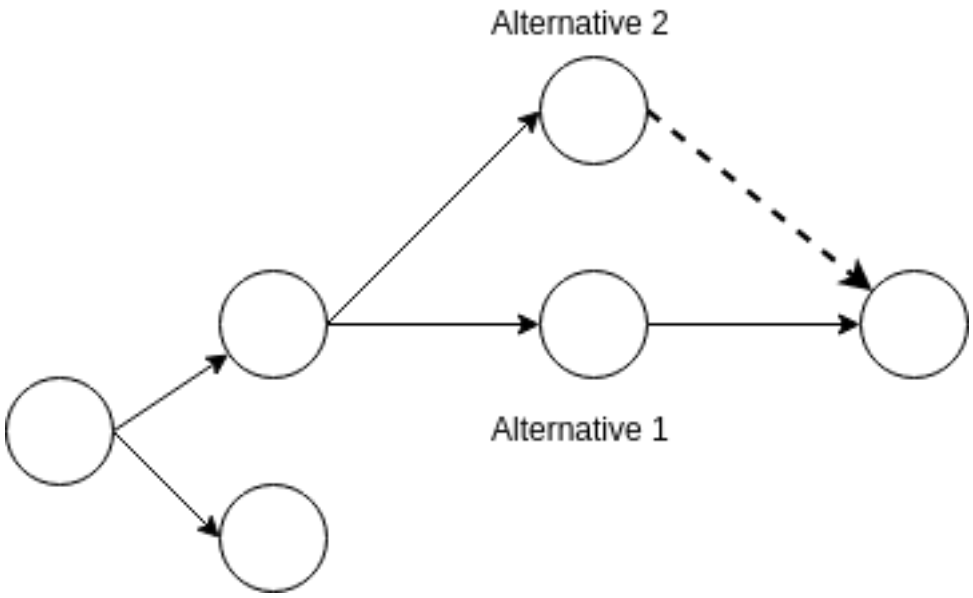

Figure 5: The dotted line is not possible using a tree structure

ents, or rather multiple vertices pointing to it. This meant that the entire database structure needed to be redesigned. Thus we opted for a design with greater decoupling: one collection of vertices, which in this case is just chapters, and another containing edges connecting the vertices. This way the network could take any form we wished. In hindsight this would probably be the better solution from the start, as it allows for more agile development as requirements change, not forcing us to use any given structure.

4.3 Challenges

During our discussions a number of challenges arose quickly, given our new perspective of the structure. This section address these issues and propose solutions.

4.3.1 Continuity

The first problem that became obvious was the problem of continuity. Maintaining continuity even in a traditional book with only one author has proven to be hard throughout literary history [25]. How then should one address this problem when faced with hundreds of authors? There are several approaches to solve this. One of our earliest suggestions was to make use of tags. More specifically letting people attach tags to each chapter they write, containing key information about what happens in each chapter in a condensed form. Although this seemed like a good solution at first, we quickly concluded that it would be hard

to do in practice. Each writer would be required to not only tag their chapters appropriately, they would also need to understand how others have tagged previous chapters. In conclusion the system requires too much of the user.

After much discussion we decided on letting this issue get taken care of by the rating system. If a particular path or chapter has continuity errors, or plot holes, users will not like it, thus giving it a bad rating. Therefore it follows that this chapter or path will not be promoted to other users and will eventually fade away amidst all the other chapters. This means that we allow a system where continuity errors may exist. This is why we ultimately decided not to have a separate system for keeping continuity, but rather letting the users of the site curate what content should be highlighted to other users and what should be hidden away.

4.3.2 Presenting it linearly

After switching to a network structure, using vertices and edges to define relations instead of pointers to and from children and parents, we also had to change the way we rendered our visual representation of the graph in our web client. In the past we used a horizontal tree structure, where the nodes were positioned by the D3 library, similar to the one in Figure 6. We could use a traditional visualization for the network, where the position of each node is random and arbitrary. An example of such a visualization can be seen in Figure 7. However, the position of the nodes is of some importance in our case. It can get visually confusing to follow the direction the story unfolds if we were to position the nodes arbitrarily. Thus we still need a structure similar to Figure 6. Our solution to this was to transform the network data on the client side so that we were still able to apply D3's tree structure transformation to the data. To be able to do this we run through an algorithm that creates a copy of the network structure, but removes any edge that can not be a part of a tree. When this process is finished we can input the new structure into the tree function provided by D3. We now have positional data for every node and can render them. Next, to render the edges, it is just a matter of looping through every edge in the original network and look up where the nodes are positioned by finding the corresponding node ids.
Hence we have combined the neat, linear structure of trees and the added complexity of a network to create an orderly visualization of the network. An example of the final product can be viewed in Figure 17.

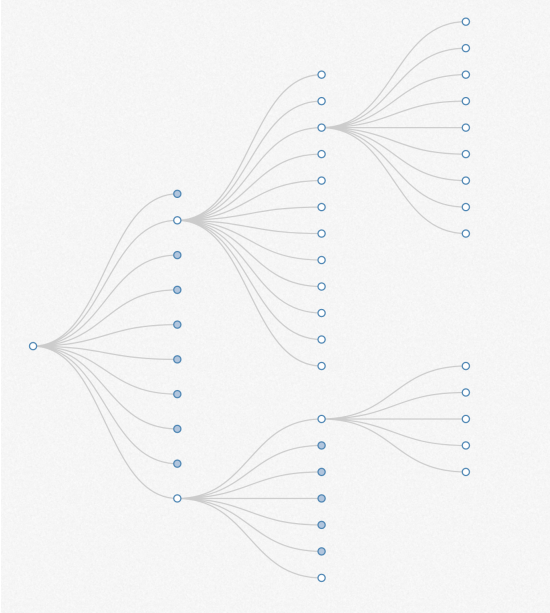

Figure 6: Example of a network using the tree layout in D3

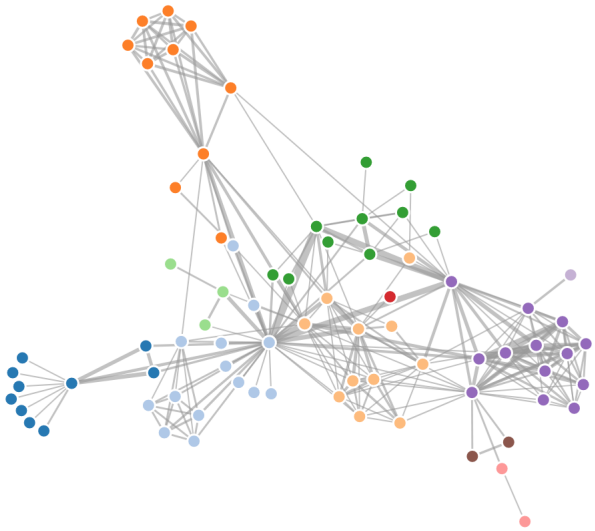

Figure 7: Example of a network using the force graph layout in D3

4.3.3 The paradox of choice

As more and more continuations are added to a chapter a new problem arises: How should the user be presented with all of the chapter alternatives following the chapter they just read. If there only exist a few chapters to choose from one can just show them all to the reader. However if the number of choices gets larger, say in the tens, a great burden is put on the reader. Forcing a user to choose one of these without any assistance stops the flow in which the reader may be in, and also gives them a huge task as they have no idea which alternative is best suited for them, forcing them to choose at random or reading every single one. That is why we decided there need to exist some algorithm that dictates what the user would most likely want to read next. Right away it seems obvious to take advantage of some feedback mechanism, like stars, thumbs up/down or any other rating system, to get a feel of what most other people like. However if this is the only criteria for which chapters should be presented, older chapters have a great advantage. The older a chapter is, the more feedback it has the possibility of getting. Hence the algorithm should also implement some way to promote the more recent chapters so that they too stand a chance to be read regardless of their young age. The solution and design we eventually settled on is described in Section 6.10.

5 Emerging questions

Our process for developing this platform was an emerging one, meaning we started out with little knowledge of how we wanted the final implementation to look like. Rather, we employed an iterative development cycle where problems and questions arose as we progressed. Some of these were adressed in our implementation, others not. This section summarizes the questions that *were* adressed in the implementation. We will restate the questions along with our solutions in Section 8, after presenting a description of the high level implementation design in Section 6. The unanswered questions will be presented in Section 10.2.

5.1 The big five

The bulk of our questions can be summarized by the following five broad ones:

- How should users give and receive feedback to each other?
- What incentives should be put in place for people to contribute with content to the platform?
- How should the users navigate the chapters?
- What restrictions and permissions should be applied to the process of adding new chapters?
- How should the network of chapters be structured?

For each of these we have a number of more specific subquestions we want to address.

5.2 Giving and receiving feedback

As our platform matured we came to the conclusion that we wanted to offer users a feedback system. Drilling down on this concept we wanted to solve the following problems:

- How should the system alert users when they receive feedback on their submitted content?
- What specific mechanisms should be implemented in order to bring a proper feedback system to the users of the platform?

5.3 Incentivizing contribution

Some ever present questions during our development were the ones concerning incentivization. For the platform to be successful, production of content is needed. Thus we needed a way to encourage this. To solve this we figured we needed to answer the following questions:

- What mechanics are other platforms successfully employing in order to incentivize contributions?
- What sort of reward should users receive in exchange for their contributions?

5.4 Navigation

As the books on our platform has a rather complicated structure it was important to provide users with an intuitive way to navigate it. In order to find a solution to this we wanted to solve the following sub problems:

- How should we present each chapter option a reader has upon reading through a chapter?
- In what way should the users be able to explore the entirety of the book graph?

5.5 Adding new chapters

Similar to the problem of how navigation would work, we were determined to make it easy and intuitive for users to add new content to a book regardless of how complicated the structure might be. Thus we developed the following questions, designed to address this issue:

- Should we allow multiple methods of adding chapters, or just one to avoid confusion?
- What tools should be offered users in order to add new complex, non-trivial paths?
- What tools should be offered users in order to add a single chapter?

5.6 Network structure

The last two questions in Section 5.1 are somewhat coupled together. Decisions made in regards to either of them affects the other one. Thus we bundle them together in search for a solution to both. To arrive at a solution we figured we needed to answer the following questions:

- Should the book graph be structured like a tree or a network?
- Which paths are allowed to be added in an existing book?

6 High level design

This section describes the high level design of the platform implementation. A more detailed description of the implementation can be explored in Appendix C.

6.1 Authorization

As the platform allows users to create content it is obvious that we need an authentication system. The implementation itself is discussed in Appendix C.3. When designing an authentication system a big part of the design is what authorization one should give authenticated and non-authenticated users. We wanted users to be able to explore as much as possible of the platform without having to create a user. Thus we opted for a design where every consumable part of the application is accessible by anyone. Actions under this category include, but is not limited to, reading chapters, viewing user profiles and reading comments. All parts of the application where the user has the ability to submit their own content is only accessible to an authenticated user. This includes features such as writing chapters, submitting comments and editing your own profile. In the version, as of this writing, these are the only two authorization rules applied.

6.2 Profiles

When a new user registers they are given a new profile page. This page is public and can be viewed by anyone. However, upon registering, one can choose a pseudonym. This will be the name shown publicly to others. We offer this functionality to encourage users to remain anonymous if they so choose to. The aim of the profile is to present the author to others. Each profile has a section called "bio". This section can be edited by an author to describe themselves in their own words. Furthermore all the contribution made by that author is displayed for others to explore. Currently this is the chapters the author has written, and the books they have created. Additionally there is a section displaying what badges the user has earned. (The concept of badges is explained further in Section 6.3.) Lastly some basic statistics about the user is displayed, containing metrics such as number of words written and the total number of chapters created.

6.3 Badges

To incentivize users to write for our platform we introduced the concept of badges. This is a virtual award given to users whenever they achieve some given milestones. Such a feature is often referred to as "gamification". This is defined by the Merriam-Webster dictionary as: "the process of adding games or gamelike elements to something (such as a task) so as to encourage participation" [26]. Examples of milestones that currently award you badges are the ones that has to do with likes. Whenever the chapters an author has written have amassed a total of hundred likes, a badge by the name "Likeable" is assigned to them. This can be viewed by anyone on the author's profile page, and is comparable to a trophy.

6.4 Feedback

To provide authors with feedback on their chapters we implemented two mechanisms. The up- and downvote buttons, and the comment system. The number of upvotes and downvotes is displayed on each chapter and can help determine the quality of them. Similarly it is possible to comment on any chapter. The comments to a chapter is displayed underneath the text and can be viewed by anyone. These features are implemented for several purposes. First, it provides feedback for the author so that they might improve on their writing. Secondly, it offers readers a platform to discuss the content of a chapter or a book with other users. Third, it helps our system to rank chapters and authors so that we can build a recommendation engine as discussed in Section 6.10.

6.5 Notifications

The platform features a notification system. This is designed to let users know when someone interacted with their content on the platform. Whenever a user likes, dislikes, comments on or creates a continuation of a chapter, the author of the relevant chapter is immediately notified. In the current version the button used to view the list of notifications is visualized by a bell in the menubar that is visible throughout the application. Whenever a user receives a notification a number appears over the bell, displaying the amount of unread notification that user has received. When the user clicks the icon, a list of notifications appear. The unread ones are marked with a small orange dot beside the notification. If a user receives notifications while not logged in, the notifications will be displayed upon login in the manner just described.

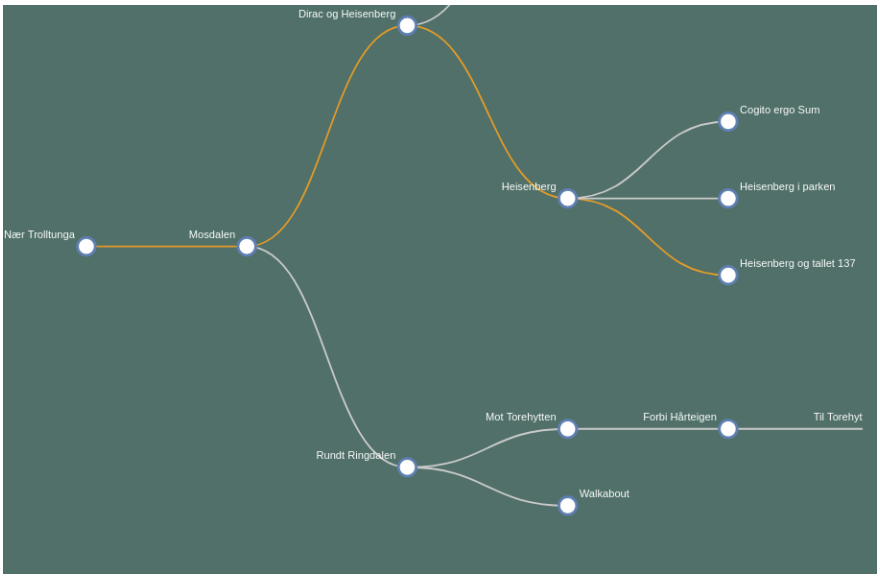

Figure 8: Screen shot of a part of a graph in WebNovel

6.6 Visual presentation of the graph

The platform is designed to offer the functionality of visualizing the graph of a book and give users the option to interactively explore it. The option to explore a book is available on all chapter pages. Thus, upon finishing a chapter the reader may choose to explore the entire book if they choose to. An overlay will slide in, displaying the entire network. The chapters are visualized as circles and the relation between them are displayed as lines. The title of the chapter is shown beside each circle. Additionally the path that the reader has walked to the chapter they are currently reading is highlighted, so that they may trace their steps. This is possible due to the "Remember History" feature, explained further in Section 6.9. Moreover it is possible to navigate the book by clicking on the nodes. Screen shots from the version as of this writing can be viewed in figures 8 and 9.

6.7 Drafts

Whenever someone authors a chapter they might want to work on it later and not publish it right away. For this purpose we designed our platform to include drafting functionality. There are essentially two types of draft: A standalone draft with no pre-set connection to any other nodes, and a draft being attached to an existing node. The standalone draft is used when a user employs the "Create New Path" functionality, explained in detail in Section 6.8.2. The other type is created when someone uses the "Continue Story" functionality,

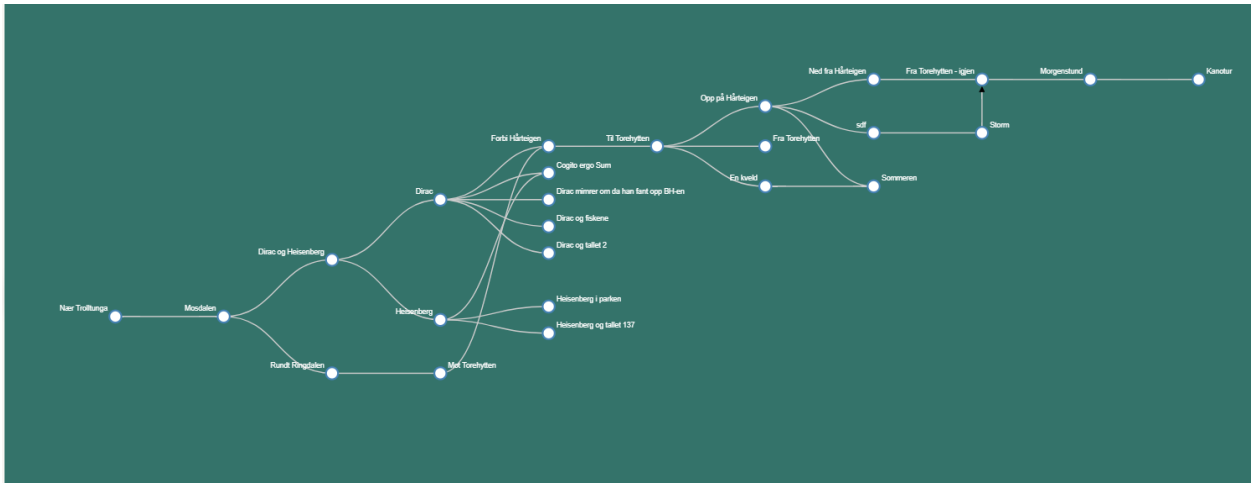

Figure 9: Screen shot of the version as of this writing

explained in Section 6.8.1, and wishes to save the chapter as a draft rather than publishing it. The link between the parent chapter and the new one is then created but is not published until the author chooses to. A user can go back and edit the draft as much as they would like. It will be hidden from others until it is published.

6.8 Inserting new chapters

If someone wishes to contribute to a book by writing and inserting a new chapter they are presented with two choices at the end of every chapter: "Continue Story" or "Create New Path".

6.8.1 Continue story

If a user decides to use the "Continue Story" option, they are presented with a text editor. Here they can write their continuation of the chapter they chose this option from. After giving the chapter a name and some content they are presented with two options: they can either "Save As Draft" or "Publish". If they pick the former option, the chapter is, intuitively, saved as a draft. This entails that it will not be visible for anyone else on the platform yet. It will only be visible to the author. The author then have the capability of editing or publishing it later. When the chapter is published it gets added as a child to the chapter it was a continuation of as illustrated in Figure 10.

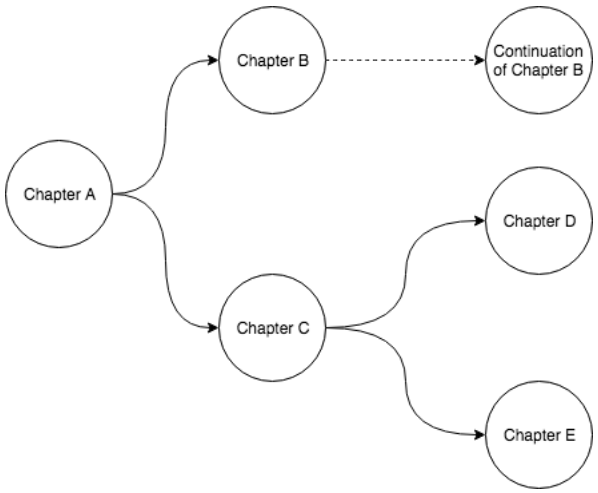

Figure 10: Illustration of how a continuation of chapter B would be structured

6.8.2 Create new path

Upon choosing this option the user is presented with the visual presentation of the graph. The user may now draw new nodes and edges under the restriction that one may not draw a new edge between two already existing nodes. Thus one can draw edges between an existing and a new node or between two new nodes. The reason for this restriction is that contributing with new chapters translates into adding new paths, however all the old paths are still the same. Nothing is altered, something is just added. If one were to draw a new edge between already existing nodes the story would be altered, and there would be a link between two chapters that was not intended by the original authors.
When the user has added new nodes and edges they can switch from draw mode to insert mode. Upon switching modes the user is presented with a sidebar containing a list of all the draft they have created. Each of these drafts can then be inserted into one of the new nodes. When every newly added node has a draft attached to it the user may publish it and the new path will be visible to everyone. Screen shots of how the current version implements this visually can be explored in figures 11 and 12.

6.9 Remembering history

We designed the platform to be able to remember what users had read. This way the reader can pick up where they left off. In the version as of this writing we use this functionality to offer several features. First it enables us to present the user with the books they are currently reading on the welcome page, making it easy to jump exactly to where you left off.

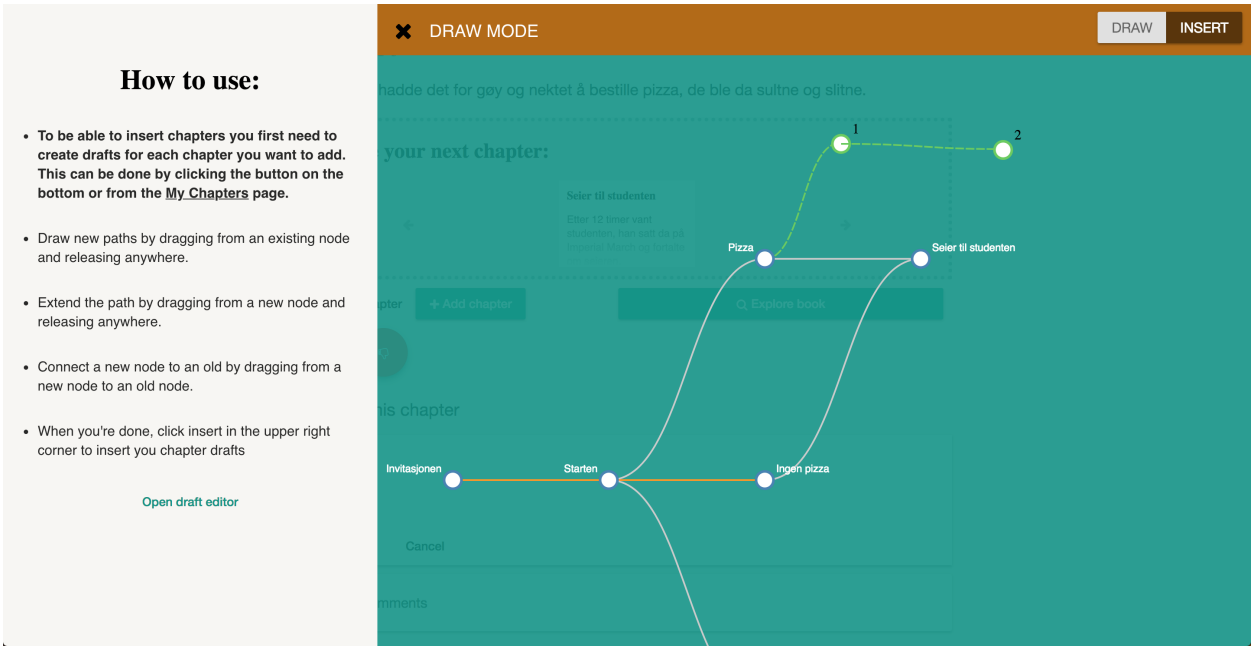

Figure 11: Current implementation of draw mode when creating a new path

When a user navigates to a book they have begun reading, they are immediately navigated to the chapter they last read. Secondly, with the history of all the chapters that has been read by the user, the path they have walked in a book can be visualized in the explore graph functionality, as mentioned in Section 6.6. Thirdly, we need to remember the history to be able to navigate the user backwards in the book. Due to the network structure of the novels there are possibly several chapters that lead to the current chapter, and accessing the history is the only way to know what chapter the user read before the current one.

6.10 Recommendation engine

After reading through a chapter, the user is presented with up to three options of what chapter they might want to read next. To guide the user in making a somewhat informed and meaningful decision we created a simple recommendation engine, selecting what three chapters to display to the user. If there are more than three continuations of the chapter, the user may click an arrow button revealing three new chapters. (Also, the user will always have the possibility to use the "Explore Graph" functionality, explained in Section 6.6.) The order of the first three chapters are decided by applying the following rules[1]:

- The first chapter is selected based on the author. The algorithm looks through every

[1]Unfortunately, due to time constraints, we were not able to implement this algorithm before deadline.

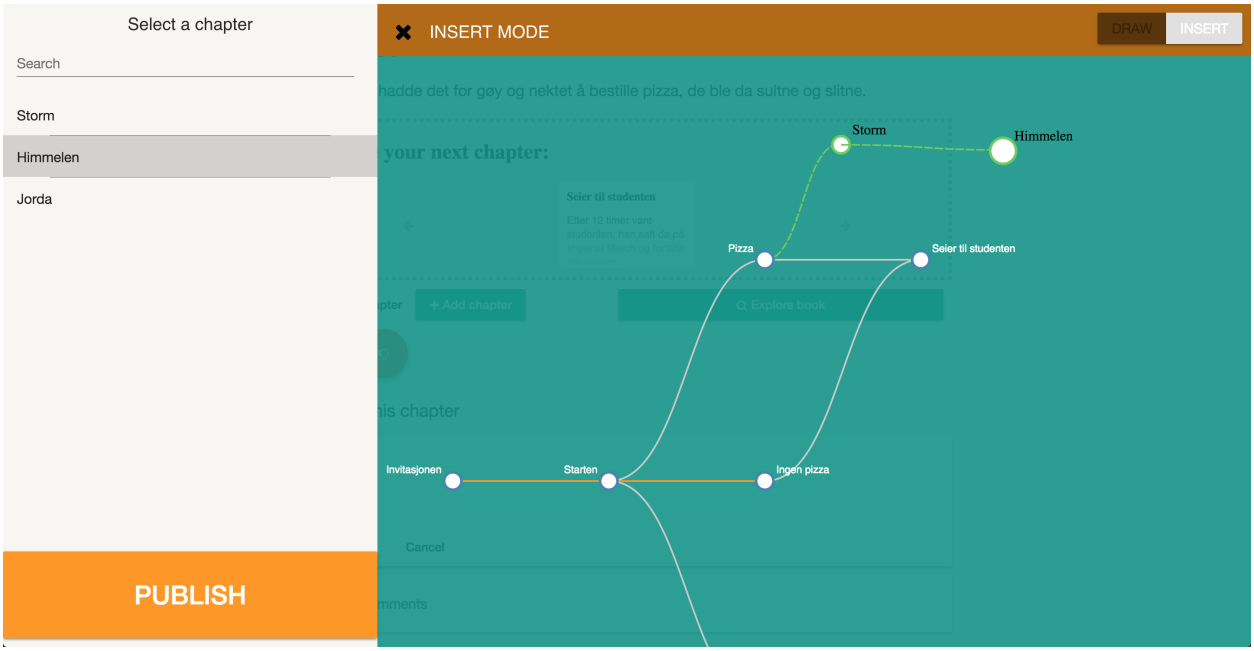

Figure 12: Current implementation of insert mode when creating a new path

available chapter and rate each author. The details around this rating is discussed in Section 6.10.1. The chapter with the highest rated author is then placed first.

- The decision of what chapter comes second is based on the chapters themselves. The algorithm looks at the feedback each chapter has received, and located the one with the most likes. If some are equal we pick one of these at random.
- The third and last chapter is selected using weighted random selection. For every chapter left the algorithm applies various weights corresponding to the chapter's creation date. The newer a chapter is, the more likely it is to be picked. Thus new chapters that has not yet received enough feedback also has a chance to be displayed to the user. This should limit the rich-get-richer phenomenon somewhat.

The rest of the chapters, after the first three, are put in completely random order.

6.10.1 Rating authors

The rating of the authors are based on what badges the authors have received. Badges implying good quality improves their rating. What badges implies good quality is subject to change as badges are added and removed from the system. Currently the "Likeable" variant of badges, that is given to a user upon reaching a certain number of total likes on all of their chapters, are the ones considered to imply good quality. The author in possession of

the highest number of these badges receives the highest rating. In the future this rating might use more data points to determine the highest rated author, however, due to time constraints, our proposal is to use this rather simple rule set.

6.11 API documentation

Included in the platform are documentation for every API endpoint. These includes a description of what schemas are accepted in POST and PUT endpoints, what type of data the user can expect in return and a short description of what actions the endpoint performs. The documentation is accessible by anyone on a specified URL. The reason for this being part of our design was to encourage others to contribute to the open-sourced codebase by making it easy to explore and understand the APIs implemented.

7 Legal considerations

As the WebNovel platform deals with user generated content we need to define some licensing that applies to the texts for legal reasons. Furthermore, as we are releasing the code as open source we also need to apply a license to the source code. This section discusses some options before stating what license we eventually decided to use.

7.1 Licensing the content

As the stories written on WebNovel is put together by a large number of people we wanted to apply a single license on every text. If we were to let the authors decide the license for each chapter they write, confusing and complicated situations could come about. If we ever wanted to do something with the stories outside of the WebNovel platform conflicting licenses might occur and so on. Thus the next sections explore what license fit our need the best by looking into two types of licenses: Public Domain and Creative Commons. The cause for the decision being between these two are that we wanted an open license, encouraging people to get inspired by other authors without fearing that they might be infringing upon their copyright.

7.1.1 Public domain

There exist several licenses that can be said to be in the category of Public Domain. However they all originate from the the same core idea: Anyone can use your content for any purpose [27]. They may print it, sell it, create a movie out of it, create music from it etc. We considered licensing the texts this way, however after some discussion we thought better of it. We wanted authors to get the credit they deserved, and feel that they had some control over their creation so as to not disincentivize publishing on our platform.

7.1.2 Creative Commons

Creative Commons is, similar to Public Domain, a generic term describing several specific licenses. The one we will consider in this section is the "Creative Commons Attribution 3.0 Unported (CC-BY)" license [28]. This license states that anyone are free to [29]:

- "**Share** — copy and redistribute the material in any medium or format"
- "**Adapt** — remix, transform, and build upon the material for any purpose, even commercially"

However one is only allowed these freedoms under the following terms:

- "**Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use."

These freedoms and constrictions are ideal for our purposes. Other authors are encouraged to build upon the material of previous works, but the author who created the original text is credited. Thus we propose this license to be applied to the content created in WebNovel.

7.2 Publishing paths as books

As books on the platform get bigger and receives a number of contributions there might emerge some paths that are, according to the rating system, considerably better than others. We have discussed the possibility of publishing part of book networks as a traditional, linear book. However, if this was to happen, how should we provide the proper compensation to those who actually wrote the book? What measure should we use for deciding how the income might be distributed? We had several discussions regarding this issue, however we did not arrive at a satisfactory solution. We discuss some of our ideas and their shortcomings in Section 10.2.3 where we outline what further work needs to be done.

7.3 Licensing the source code

Upon completion of this thesis we would like the prototype to be improved upon. In an attempt to achieve this we are publishing the code as open source so that anyone interested might help to better the system. To do this we need to decide upon a license. We wanted the licensing to be as free as possible. This included the possibility for others to copy the work and create their own version. Thus we decided to use the MIT license [30]. This license gives anyone permission to "deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software" under the condition that this license is included in "all copies or substantial portions of the Software" [30]. By applying this license to our implementation we hope to encourage improvement in the software so that it might be enjoyed by a larger number of people.

8 Solutions

The reader should now be familiar with the high level design of the platform. This section will answer the questions posed in Section 5, in the form of referring to where in we have described the solution in former sections.

8.1 Feedback

How should the system alert users when they receive feedback on their submitted content?

To address this question we introduced the concept of notifications in Section 6.5. In addition to alert users of received feedback it also functioned as a communication tool to let users know whenever they received a badge and when someone created a continuation of their chapter.

What specific mechanisms should be implemented in order to bring a proper feedback system to the users of the platform?

This question was addressed in Section 6.4. We discussed a comment system and a like and dislike system for rating chapters.

8.2 Incentivization

What mechanics are other platforms successfully employing in order to incentivize contributions? What sort of reward should users receive in exchange for their contributions?

Exploring other platforms dependent on content creation from their users, such as Stack Overflow [31], we noticed a common pattern. Users would receive some virtual reward upon achieving certain milestones. Our version of this concept was to use badges as presented in Section 6.3.

8.3 Navigation

How should we present each chapter option a reader has upon reading through a chapter?

This question was addressed in Section 6.10. Following each chapter we display up to three contuination options of which the order is determined by the algorithm explained in

the mentioned section.

In what way should the users be able to explore the entirety of the book graph?

Our solution to this was explained in Section 6.6. We offer users the ability to interactively explore a visualization of the book graph that includes a highlighted part representing the story path they have walked so far. Additionally, the user is able to navigate the book by clicking on the nodes representing the chapters.

8.4 Adding new chapters

Should we allow multiple methods of adding chapters, or just one to avoid confusion? What tools should be offered users in order to add new complex, non-trivial paths? What tools should be offered users in order to add a single chapter?

We settled on offering two distinct methods to add chapters, as explained in Section 6.8. We thought it best to implement an option to just add one continuation to a chapter, thus making this process quick and easy to do if this is all a user wants. Additionally, users are able to use the more comprehensive graph editing tool when they want do something more complex.

8.5 Network structure

Should the book graph be structured like a tree or a network? Which paths are allowed to be added in an existing book? The answer to this question was discussed in detail in Section 4. We eventually settled on using a network structure and letting the user create new paths between two nodes if either of the are a newly inserted chapter.

9 User tests & surveys

As a part of the evaluation process of the platform we performed user tests with several participants. These were done over three development iterations, each time tweaking some part of the application in an attempt to address the shortcomings revealed during the user tests. This section will first explain how we designed the study. Following that we discuss the results collected in the tests and survey.

9.1 Survey design

The participants in the survey were mainly acquaintances of the author. Thus the type of sampling is a convenience sample in which respondents are chosen based on their convenience and availability. While random sampling is what is most sought after, we did not have the resources or time to get participants retrieved in a random or probabilistic way. The participants are in the age range of 20 to 28. The sample size is also somewhat limited, due to the nature of the gathering of the participants. The total number of participators are 19. Despite these shortcomings, we found the tests and surveys helpful, as they shed light on pain points in the application that was shared across participants.

The survey instruments used were a combination of asking the users to complete certain tasks in the application while the author was watching them, and asking several questions in an online form that were to be answered after using the application for some time. Six of the participants completed both methods, while thirteen explored the website unsupervised before answering the online survey. The unedited answers are presented in Appendix D.

The first version exposed to user testing were used by only two people, however this was enough to uncover a number of bugs and unintuitive parts of the application. Fixing some of these issues, we let two new users test the platform. This uncovered several new issues we had to fix. Thus, after implementing some changes addressing the issues, we arranged two more supervised user test with yet another two people. This is also the version that the unsupervised participants were using. The located issues, and implemented solutions are discussed in the Section 9.2.

To gain an understanding as to how we designed the questions in our survey, and what purpose they have, we present each question in Table 1 along with a short explanation as to

what information we are trying to obtain from the answers to that question.

Table 1: Questions posed in the survey and their accompanying explanation

Question	**Explanation**
What was your first impression when visiting the website?	The first impression of something is always important, and can be major factor in telling whether new users are intrigued and want to explore the application or leave it. If this receives positive feedback, we kan maintain the overall design of the application. If this receives negative feedback, we might need to change the welcome page and some of the overall design elements.
Explain shortly what sets WebNovel apart from traditional novel writing tools.	This question is designed to figure out if they have grasped the core concepts of the platform, namely that novels are structured as a network and not in a linear fashion, as well as the collaborative aspect.
What did you like about the application?	Retrieving information about what we did right is of pivotal importance to know that what we are doing is of interest to people. Also, knowing what parts are enjoyable makes it easier to create new features in the future that is inspired by what people have found enjoyable before.
What did you not like about the application?	This question should reveal what parts of the application are hard to use or understand. Ideally it should also give us some insight into what we should change in our next development iteration.

What do you think should be improved or be removed from the application?	This is somewhat similar to the last question, however it asks for a more concrete answer. While the question above makes room for answering it in a somewhat abstract manner, this asks the participant to name specifically what functionality should be improved or removed.
Do you find the application fun or boring to use? Why?	This question is designed to figure out if people actually enjoy using the application, which is key in it being successful, as it being an enjoyable experience determines whether users will return.
How easy is the application to use?	Obviously we want users to be able to use the application and find it intuitive to use. The answer here is on a scale from 1 to 5 where 1 is very hard to use and 5 is very easy to use. If the answer tends to be low we know that we need to do some major changes in regards to usability.
What functionality do you think we should add?	This question exists in case the participant has some good idea of some functionality that we have not thought of, enabling us to implement it in our platform.
What score would you give if you were to rate WebNovel from 1 to 10, where 1 is garbage that you never want to use again, and 10 is that you would gladly visit the application daily?	This should act like an overall rating, giving us some insight into how the overall usability, user experience and enjoyment is. If this is low we need to do some fundamental changes. However if it is high, we know that we might be onto something.

9.2 Results from supervised test sessions

We evaluated the platform, using the supervised approach, on three different occasions, improving our design based on the feedback each time. The first version was tested by two users. They were first asked to explore the page by themselves. Afterwards we asked them

to try to create a new book. They were then asked to create a new chapter in that book. Lastly they were asked to use the advanced editor, adding a more complex path to the book. In the following section we describe how each of them followed the instructions and how we addressed the issues:

Kristoffer:
When asked to create a book, he first appeared confused, as he did not find any place to create books. After a few seconds he realized that he had to register a user. The application failed to give enough feedback to make this point obvious. After creating a new profile he quickly managed to create a new book. After publishing it he was asked to create a new chapter following the one he had just added. He immediately figured out how to do this, and quickly added another chapter. Lastly he was asked to create a new path, where he added two new chapters simultaneously. He quickly found the button that was supposed to be clicked to allow this, however he was confused when presented with the visual editor. Although an explanation as to how to use the editor was provided on the sidebar to the left, he did not seem to read it before starting to add new nodes. Eventually, after not figuring it out he noticed the explanation text to the left. Upon reading this he managed to insert new nodes and attach chapters to it. However he had a problem figuring out how to publish the newly added path. After looking around a bit he eventually found a button reading "Save". Clicking this made a little message pop-up, saying the path had been added successfully, however he was unsure if it had been saved properly, as the editor remained the same after the message disappeared.

Ida Sofie:
When visiting the site she went straight to the sign up page, and did not read the welcome page containing an introduction to the application. After creating the profile she was navigated to the "Discover" page. Although she was already on the page she tried clicking the button navigating to that page, as there were no indication that this was the page she was already on. Next she clicked on one of the books and started reading it. After reading a chapter she wanted to give it a "Like". She tried clicking the statistics showing the number of likes, under the title. She was confused when nothing happened. (The button to actually give likes and dislikes are located at the bottom of the page). When asked to create a book,

she quickly managed to figure out how to do this. When asked to navigate an existing book she first used "Choose your next chapter" box presenting her with three chapters. After a while she tried clicking the "Explore book" button. She was at first confused by the fact that although the path she chose was highlighted, she did not seem to actually navigate to the chapter she clicked on. This was due to the navigation happening in the background, while the graph was still overlaying the chapter page. Thus she did not notice the change happening.
Next, when being asked to add another chapter to the book she created, she immediately clicked the "Create chapter" button. However when clicking it she seemed confused about the two options. Still she eventually chose the "Continue story" option and had no issues from that point adding a new chapter. Lastly she was asked to try to add several chapters at once. She quickly found the "Create new path" button, and was presented with the visual editor. Similarly to Kristoffer in the former section, she did not seem to read the instructions found on the side, before starting to add new nodes. This resulted in confusion. She exited the editor and then opened it to try again. This time she noticed the instructions and followed them. Then, after drawing a new path and getting to the part where she inserts chapters into the new nodes, she was confused as to how to do it, although a new message had popped up telling her to select one of the new nodes. After a while she figured it out and clicked one of the new nodes. She then was presented with an empty list, telling her that she had to create a draft first, followed by a button to write new drafts. She clicked this, wrote a draft and saved it. After saving the draft she was navigated back to the "My Chapters" page. She thought this confusing, as she figured she would be taken back to the editor afterwards. However she quickly managed to navigate back to the editor, and were now in fact able to add the new draft by using the visual editor. However, similarly to Kristoffer, she had a hard time finding the "Save" button, and was unsure if the path had been published as the editor still looked the same.

These tests, although few, was enough to uncover a number of flaws. We made the following changes for our next version:

- Whenever navigating using the graph visualizer, the graph now disappear when clicking on a new chapter, revealing the chosen chapter, to clearly inform the user that the they have successfully navigated to the desired chapter.
- The explanation on the left were altered to use more screen real estate. Additionally

we added a big header on the top, reading "How to use". Lastly we added a point in the beginning, telling the user that they need to have drafts available to use this functionality, providing a link to the draft page. This was an attempt to combat the confusion that both participants had experienced when using the visual graph editor.

- We added the ability to be able to like or dislike a chapter by clicking on the statistics on the top of the page, as this seemed to be an intuitive way for one of our participants to use this functionality.
- The save button was changed to read "Publish". Furthermore we enlarged it and changed the color so that it would be more visible. This was done in an attempt to make the publish functionality easier to use and understand.
- We changed the "Create chapter" button to read "Add chapter", as this seemed to better describe the actions that can be performed when clicking that button.

The new version, exhibiting these enhancements, were then tested with two new, different people, under supervision.

Oda:

When first visiting the site she first clicked the "Start reading" link, which navigates the user to the "Discover" page, showing most recent books added to the platform. After clicking on one of the books, and reading the first chapter she immediately clicked the "Explore book" button. She seemed to think that the navigation using this was intuitive, in contrast to the last version, as it was more clear that the chapter now had changed, as the graph visualization was closed. When asked to create a book she managed to do this quickly, as she immediately figured she needed to first create a user. When asked to create a new chapter in the new book she figured that she had to use the "Add chapter" and then "Continue story" button quickly, and was able to add a continuing chapter. Next, when asked to add several chapters at once she quickly opened the graph editor. She seemed to notice the explanation right away, and started by creating two new drafts. However, similar to the others, she was confused by the fact that she was brought to the "My chapters" page after creating the draft instead of back to the editor. Eventually she figured she needed to go back to the book and open the graph editor again. She picked up how to draw new paths quickly, as she read through the explanation first. However, when switching to insert mode she did not immediately realize that she had to click one of the new nodes, despite the sidebar reading "Select one of the new nodes". However she eventually figured it out, and added the two

drafts to the book. She had no problem publishing the new path, however she was confused about it actually being published or not.

Connor:
When first visiting the welcome page he read the short introduction displayed there, before immediately creating an account. Similarly to Oda, he found navigating the books to be easy with the visual graph explorer. Furthermore he had no problems creating a new book. When asked to add a chapter to the book he used the "Create new path" functionality instead of "Continue story". He noticed the explanation, and started creating new drafts. After creating the drafts, he too were confused by the fact that he was not navigated back to the graph editor but to the "My chapters" page. However he figured how to get back to the book, and managed to add new paths and publish them with ease, except for the fact that he was unsure about the path actually being published after completing the steps. He also made sure to tell us that he really enjoyed using the "Explore book" functionality to navigate the more complex books in the system.

Again, these tests helped us uncover more faults with the design of the application. What follows is a list of the improvements we made for the next version.

- We altered the behaviour when clicking the publish button to close the editor, in hope that it would better communicate that the path had been added successfully.
- When switching to insert mode in the graph editor we changed the directions on the side from "Select one of the new nodes" to "Click on one of the newly added chapter nodes". This was to combat the fact that some were confused about what they had to do. We figured changing it to using "Click" instead of "Select", and adding "chapter" before "nodes" would help people not familiar with the system to better understand the directions.

Similar to the last iteration, this new version was tested on two new participants.

André:
When visiting the page he quickly glanced over the introductory text before signing up for an account. He explored some books, and seemed to find the "Explore book" functionality both

enjoyable and intuitive. Similar to other participants he found it easy to add a new book, and add a continuing chapter to the book. Also similar to others he found it confusing to add new chapters using the "Create new path" functionality, especially the way that adding new drafts discarded what he had done in the graph editor, where he had to navigate back to the book and open it again. However, after creating the drafts he managed to fairly easily use the editor to insert the drafts into the graph.

Thomas:
Upon being presented with the welcome page he did not seem to read the introduction displayed there, and went straight for the "Start reading" link. After reading some books and chapters, he went to sign up for an account. Next, he tried to add a chapter to an existing book, however he was confused about the two options under "Add chapter". He first clicked the "Continue story" option. He then seemed to understand how that option worked, and successfully added a continuation chapter. Similar to the others he managed to create a book with ease. When asked to use the "Create new path" functionality he read the instructions first, before clicking the link to create new drafts. As all others he was confused by the fact that he was not directed back to the graph editor afterwards. Additionally he mentioned that upon clicking "Save draft" he did not expect to be navigated away from the page. After navigating back to the graph editor he managed to insert the new draft. However he noted that he felt that the process was somewhat unintuitive and cumbersome.

Throughout the testing, it seems the necessary draft creation when using the graph editor has been subject to most of the confusion. Thus in the last version we changed the behaviour somewhat. When presented with the explanation upon opening the graph editor we instead offer the user to create drafts from the same page, by using a dialog. The user can click a button reading "Open draft editor", which will open a dialog on top of the application with a text editor. When finished creating the draft, the dialog is closed and the user can return to using the graph editor. Unfortunately we did not have the time to test if this new functionality made the process easier for users, due to time constraints.

9.3 Results from online survey

In addition to supervising users while they used the application we asked several people to test the application by themselves, before answering an online survey with the questions presented in Table 1. Their responses can be viewed in Appendix D. In this section we discuss the answers received from each question.

What was your first impression when visiting the website?
Overall the respondents seem to like the design of the page and found it pleasing to the eye. We interpreted this as the general design being on the right track. There were some feedback saying that it did not look to good on a phone. We were aware of this, but have put our efforts into creating a working desktop version without presently having phones in mind.

Explain shortly what sets WebNovel apart from traditional novel writing tools?
Originally we designed this question to reveal whether the user had understood that the novels were structured as a network. Although some answers indicate that they have indeed understood the concept, a large number of people have answered that they had not tried any other similar tools. In hindsight we should have replaced "novel writing tools" with "writing tools" or "word processor". The participants seemed to think we meant other tools that provide similar functionality, when what we really meant was any other word processing tool. However, putting those responses aside, many answers mention the collaboration aspect as well as the network structure. Thus it seems like the general idea of the platform are properly conveyed to the users.

What did you like about the application?
The response here were somewhat scattered. However it seemed to mainly be about three things: The general design of the application, the fact that one can cooperate on chapter and create different paths, and the "Explore book" functionality allowing them to navigate books visually.

What did you not like about the application?
The most recurring pain point in the responses is the difficulty in adding new paths using the "Create new path" functionality. The rest of the responses are a bit scattered in their feedback, mentioning several minor issues.

What do you think should be improved or be removed from the application?
The answers to this question do not share one common aspect. Besides the people answering that they can not think of anything, each answer is distinct from person to person.

Do you find the application fun or boring to use? Why?
Apart from two responses, saying it was "okay" and that it would be fun for those who "like to write", every other response said the site was fun to use. Several people noted that it was fun to be able to read what others had written on the site.

How easy is the application to use?
This question was answered in the form of a scale from 1 to 5, where 1 was very hard to use, and 5 was very easy to use. A histogram can be explored in Figure 24. 79 percent of the participants answered 4 or 5 in terms of ease of use. Also, upon closer inspection of the people who gave 2 or 3, they were the ones mentioning the difficulty of using the visual graph editor in some of their other answers. Thus, it seems like the website in general is easy to use for most people, with the exception of the graph editor. A fact that has been apparent throughout the feedback.

What functionality do you think we should add?
Here the participant had the option to leave the answer blank if they could not think of anything. In total we got 7 non-blank responses, each suggesting different functionalities.

What score would you give if you were to rate WebNovel from 1 to 10, where 1 is garbage that you never want to use again, and 10 is that you would gladly visit the application daily?
The histogram displaying all the scores can be explored in Figure 26. The average score given by the 19 participants was a score of 6,8.

9.4 Conclusion

What stands out most in the feedback is the difficulty in using the visual graph editor. Further work should therefore be focused on creating a better experience in regards to this functionality. Additionally there are some additional minor pain points expressed by the survey participants. Luckily these should be easy to address, as they are more concrete in how the application should be improved. Moreover, the overall impression of the result

is that most people seem to enjoy using the application. Most people have expressed that they believe the idea to be exciting. This indicates that we have created a product that people want to use, and that it has a place in the world, which should encourage further development of both the concept and the application.

10 Results & lessons learned

10.1 Results

Our research question was defined in the introduction as:
What challenges arise upon developing a digital collaboration platform for writing novels, and how should we approach them?

By using an iterative development process, both in terms of implementation and requirements, we have come to and presented challenges that arose and suggested solutions to them. This resulted in the WebNovel platform, that is available to use as of this writing. However, in addition to the challenges and questions where we proposed solutions, we uncovered other challenges that we think should be addressed. The time constraint present during this thesis lead to us not being able to suggest proper solutions to these. Thus we encourage others to continue the research.

10.2 Further work

During development we uncovered two big challenges that we wanted to solve but where unable to, due to time constraints. They are summarized by the following four questions: **Is the technology applicable in an educational context?**, **Who has the rights to edit and delete texts?**, **Is there a feasible way to publish paths as books, and ditribute the possible income fairly?** and **How should we address the shortcomings revealed in the user tests?** We have some thoughts regarding all questions, thus we will discuss them here so as to provide a better starting point for anyone who wishes to continue the development.

10.2.1 Education

We realized that this technology might be a valuable tool in education. It could be used in a way where a class together could create a network book. The teacher would have complete control over the content and would be able to delete and edit chapters. Each student should be able to receive feedback on their chapters. This feedback would only be accessible by them and the teacher.
We discussed several models of how this should be integrated into the school day. We propose two possibilities. One is to use it in a somewhat traditional way, where teachers assign every

student to contribute to a book created by the teacher. This would then be homework for everyone during a limited time period.
The other possibility is to use the technology throughout the school year, incentivizing a more natural growth during a longer time period. The teacher could then assign a number of students each week to contribute with a chapter to the book. When the school year is over they are left with a network book that has received contributions by every one in the class over the course of a year.

10.2.2 Editing texts

We spent a number of brainstorming sessions discussing who in the system should have the privileges to edit and delete chapters. We did not arrive at a proper solution, thus the current implementation only allows the original author to edit their own chapters.

One of our proposals during these sessions were the possibility of giving editing abilities to users with a moderator role. There could exist multiple types of moderators. One could be a super-moderator that are given the ability to edit and delete whatever chapter they choose to. The other type of moderator could be users with a more limited privilege scope. They could be assigned by super moderators to moderate some group of books, perhaps books that they themselves have been a main contributor to. We did not arrive at any solution as to how the super-moderators should be assigned their role.

Another possibility would be to create functionality allowing people to create both private and public books. A public book would function the same way as they do in the version as of this writing. The private ones, however, could have a limited set of authors allowed to contribute. It should also be discussed whether these books would be required to be public for anyone to read or if they have the option to limit readers to invited users. Our proposal is that the original creator of the book would have editing and deletion rights, as well as the possibility of assigning the moderator role within the book to any other users, offering them the same privileges.

10.2.3 Publishing paths as books

As previously mentioned in Section 7.2 we had some discussions revolving around the possibility of publishing particular paths as a traditional book. This should be trivial to do in a technical sense, the main concern is the legal and logistical aspect of it. More specifi-

cally, how should the income be distributed. We developed some very simple models such as splitting it evenly between every chapter. That is, if one author has contributed with five chapters and another author has contributed with one, the former receives five times as much money. However this seemed unfair, as some could write a number of chapters that might be significantly worse in quality compared to someone writing less chapters. This argument also holds for paying authors based on number of words. As we did not arrive at any satisfactory solution, we encourage others to develop a viable model.

10.2.4 Shortcoming revealed from user tests

As mentioned in Section 9 the most confusing part of the application seems to be the adding of new paths. Consequently further work should address this issue, and create a solution making this process more intuitive. Additionally, reading the answers from the survey may help locating some minor changes that would need to be implemented in the application.

10.3 Conclusion

By conducting user tests we concluded that this type of application is something people are eager to use and explore. Consequently we hope that this project are to be developed further, even though our work on this thesis is completed.

Appendices

A User Manual

This section describes how to use the various features in the application. As of this writing the application is located on the http://strys.xyz domain. This is subject to change in the future. The updated link can be found in the documentation on the public GitHub repository: https://github.com/oleeskild/webnovel

A.1 Registering a new user

To register a new user one needs to click the "REGISTER" button in the upper right corner on the toolbar that is visible throughout the application. This will navigate the browser to a form. Input the desired values and click the "REGISTER" button at the bottom of the form. You should now be logged in and navigated to the home page.

A.2 Logging in

If one has an existing user in the system, one can log in with the credentials created in the registering process by clicking the "LOGIN" button in the upper right corner on the toolbar that is visible throughout the application. This will navigate the user to a form where one can input one's credentials.

The following sections assumes that one is logged in when using the application.

A.3 Creating books and drafts

To create a new book, click the "CREATE" button in the upper left corner of the toolbar that is visible throughout the application. The browser will navigate to a new page containing two sections. There are two buttons visible, reading "NEW BOOK" and "NEW DRAFT".

Clicking the "NEW BOOK" button navigates the browser to a new page. A form with various inputs should be visible. Fill each input with the appropriate values and click the "PUBLISH" button at bottom. The book, including the first chapter, should now be visible to everyone on the platform.

Clicking the "NEW DRAFT" button navigates the browser to a new page containing a form with two input fields. These are the chapter title and the body of the chapter. Fill these with the appropriate values and click the "Save as Draft" button at the bottom of the form in order to save the chapter as a draft. It should now be visible under "My Chapters" as explained in Section A.5.3.

A.4 Reading books

To view the recently created, as well as your previously read books click the "DISCOVER" button in the upper right corner of the toolbar. This will navigate the browser to the home page. A number of books should be displayed. Clicking on any of them should navigate the browser to the chapter the user was previously reading in that book. If the user has yet to read the book, the first chapter will be displayed. At the bottom of each chapter there is a box displaying up to three possible chapter continuations. Clicking on any of them will navigate the browser to that chapter. If there are more than three chapter continuations, it is possible to click the arrows on either side of the chapter alternatives. This will reveal up to three new chapter continuations.

A.5 Creating and inserting chapters

In order to insert new chapters to a book, one first need to navigate to the relevant chapter one wants to continue from.

A.5.1 Continuing a story

To continue a story, that is, adding a new possible path from the current chapter, click the "Add chapter" button. This is located on the bottom of the page. This will reveal a menu. Click the "Continue story" option. This will navigate the browser to an editor. After finishing the chapter on can either save it as a draft or publish it right away. Saving it as a draft means that it will not be visible to anyone else, however the system will remember what chapter it was a continuation of. Thus, upon publishing, the chapter will automatically be placed correctly.

A.5.2 Creating a new path

To create a new path, that is draw a new path in the graph containing an arbitrary number of new nodes, click the "Add chapter" button. In the menu that reveals itself choose the "Create new path" option. Clicking this reveals the visual graph presentation of the book on the right and a sidebar on the left. Initially one is in draw mode. The explanation of how to draw is provided in the right sidebar. Following this explanation, one can create any number of new nodes, each with a number displayed beside them. It should now be possible to switch to insert mode by clicking the "INSERT" button in the upper right corner of the

graph window. Clicking this will bring in a new sidebar on the left. This will ask the user to select one of the newly created nodes. Clicking any of them will cause the sidebar to display a list of the drafts created by the logged in user. Selecting one of these will insert the chapter into the selected node. This is reflected in the graph as the text on the side of the node change to the chapter title. This should be done for every new node. Completing this enables the "PUBLISH" button in the bottom left corner. Clicking this will publish the new path.

A.5.3 Editing a chapter

To edit a chapter start by clicking the pen name visible in the upper right corner of the toolbar. This reveals a menu. Click the "My Chapters" menu item. This navigates to a page displaying every chapter created by the logged in user, including unpublished drafts. Clicking any of the chapters will navigate to a page displaying the chapter. On the top of the chapter content there is a "EDIT CHAPTER" button. Clicking this enables the user to edit the chapter inline on the page. When finished, one can either discard or save the changes by clicking the relevant button.

A.6 Exploring a book graph

To explore a book graph one first have to view one of its chapters as explained in Section A.4. The "Explore book" button is located on the bottom of the chapter body text. Clicking it will reveal an overlay displaying the entire book as a graph. It is possible to zoom by using the scroll function of the mouse. Panning is also possible by holding and dragging the background with the mouse. The path that is highlighted in orange displays what path the user has walked so far in the book. Lastly, clicking on a node that is either a child or a parent of the current chapter will navigate the browser to that chapter.

A.7 Profiles

Every user on the platform has their own profile displaying various information about them. It is possible to edit one's own profile as well as view anyone else's.

A.7.1 Editing your profile

To edit one's profile click the pen name in the upper right corner of the toolbar. A menu will reveal itself. Clicking on the "My Profile" menu item navigates the user to one's own profile

page. To edit the bio section click the "EDIT BIO" button on the left of the page. This allows the user to edit the content. When done one can either save or discard the changes by clicking the appropriate button.

If one wishes to edit any other information one should click the "Account settings" menu item. This navigates to a page allowing the user to change pen name, full name and password.

A.7.2 Viewing the profile of others

To view the profile of others one first need to be on a chapter page. Depending on the browser window size there will either be a byline under the title, or an author card on the left of the text. Clicking either of these will navigate the browser to the relevant profile page.

A.8 Viewing and submitting feedback

To view comments, number of views or number of thumbs up or down one needs to navigate to a chapter. The number of views and thumbs up or down are visible underneath the chapter title. To display comments, scroll down to the bottom of the page. Click the "Show comments" button, and the comments made to this chapter should reveal themselves. In order to submit a comment one can use the text field above this button. To submit the comment for others to see, click the "Publish" button at the bottom of the text field. If one wishes to give the chapter a thumbs up or down there exists two buttons above the comment text box that allows for this when clicked.

A.9 Notifications

In order to view notifications, click the bell icon in the upper right corner of the toolbar. This will reveal a list displaying every notification received. Clicking on one of them will navigate the browser to the relevant page. I.e. if one clicks on a notification regarding a comment on a chapter, the browser navigates to that chapter. If one has any unread notifications, the amount will be displayed in the form of a number on top of the bell icon. The notifications in the list that are unread will display a small orange dot on the left side to indicate this.

B Development environment setup

This section describes how one can setup the developer environment in order to run develop the application locally.

B.1 Prerequisites

The setup described here will work on all three major operating systems: Windows, macOS and Linux. We assume that node and git is installed on the system and in the path. This can be checked by issuing the following commands in the command line: "git –version" and "node –version" respectively. If any errors occur, one needs to install these packages before continuing by following the instructions accessible at https://nodejs.org/en/download/ and https://git-scm.com/book/en/v2/Getting-Started-Installing-Git.

B.2 Retrieving the code

The entire implementation is published through a public git repository on GitHub. Thus to download the entire code base enter the following command:
git clone https://github.com/oleeskild/WebNovel.git
Now change into the newly created directory "WebNovel". This will be the root directory of the project.

B.3 Installing necessary tools

In the root folder there should exist a file under the name "package.json". This files specifies what node packages are required in order to compile and run the code. These can be installed automatically by using node's package manager, npm, which should be installed if node is installed correctly on your system. To install the packages run:
npm install
If no errors occurred, all packages should be installed successfully. (There might occur some warnings, however this usually can be ignored.) However, two other applications need to be installed as well. These are the MongoDB server and the Redis server. The instructions on how to install these are found on the following URLs: https://docs.mongodb.com/manual/installation/ and https://redis.io/topics/quickstart. In order to run the WebNovel server, these both needs to be running.

B.4 Environment variables

To successfully run the application there are some environment variables that need to be set. Each will be explained in the following paragraph.

- **MONGODB_URI**: This variable hold the connection string to the Mongo database. It should look similar to: mongodb://username:password@hostname:port/databasename
- **SECRET**: This is a variable used when hashing the passwords in the database. This can be any string, however remember that if it is lost all passwords needs to be reset.
- **RECAPTCHA_SECRET**: When registering new users, the user needs to pass a recaptcha test. If one wants to use this feature one can obtain a recaptcha secret from https://google.com/recaptcha.
- **NODE_ENV**: This can either be "development" or "production". When set to "development" a more development friendly log is printed in the application log.
- **REDIS_URL**: This variable holds the connection string of the Redis database. In a local context this, by default, is: redis://127.0.0.1:6379
- **PORT**: This is the port where the web server will listen for incoming connections.
- **PROCESS_TYPE**: This can hold two values. Either "web" or "badge-worker". If the value equals "web", the web and API server is started. If the value is "badge-worker" the process that determines if anybody has achieved any badges is run. Normally both processes are started in parallel. However, if one doesn't need the badge feature while developing it is not necessary to start this process.

B.5 Transpiling client code

The client code for the front end is written in Typescript and thus needs to be transpiled into JavaScript. It also needs to be bundled in the correct way. This is an automated process, however we need a new package in order to do so: angular-cli. It is possible to install this globally in your system by using npm: "npm install -g @angular/cli" (On *nix systems one typically have to run this command with the root user). In order to transpile and bundle the client code run "ng build" in the root directory. The "–watch" flag can be added for live, continuous builds.

B.6 Running the server

If everything is done correctly we should now be able to run the application. Inside the root folder run
node index.js with the appropriate variables defined in your environment. In order for badges to work, one is required to start two processes. On with environment variable set to "web", the other one set to "badge-worker". In the latter configuration only the REDIS_URL variable needs to be set. The application should be accessible at 127.0.0.1 on the port specified by in the PORT environment variable.

C Implementation details

In this section we describe in more detail how some of the non-trivial parts of the platform was implemented. We will not present any code here as the code base in its entirety is published and accessible at https://github.com/oleeskild/webnovel. We will instead attempt to provide the reader with an intuitive understanding of how the implementation works, and how the different parts all work together to create a cohesive platform. We will first, in Section C.1, illustrate the architecture of the entire application and how it all fits together. In the latter sections we will go into more detail in how some non-trivial components are implemented.

C.1 Architecture

Figure 13 illustrates how the architecture of our platform is set up.

Components in the front end define what services they want injected by the Angular injector. These services are an abstraction layer on top of the API calls. The API calls are done via a custom service under the name of "WNHttp". This service is an extension of Angular's own http library. However it always adds the authentication token to the header before the requests is sent. Additionally there exists a service not using WNHttp. This is the **Socket** service. This service is an abstraction layer on top of the Socket.IO connection to the server, and can be injected into any component just like any other service. When the client issues requests to the server, the server will communicate with the database to update or acquire the relevant data before responding to the client with the result. Additionally the server communicates with a Redis server, that is used to store a user-to-socket mapping as explained in Section C.4. The badge worker process periodically checks if there are any users that are to be awarded a badge. If it finds any, it communicates this to the relevant users through a socket connection.

C.2 Badges

A badge should be awarded as soon as possible after the event that triggered it happened. Implementing this functionality can be done in a number of ways. The most obvious one is to check whether a badge should be awarded any time a user performs an action that might result in a badge award. This however is very resource intensive. Depending on what the requirements of the badge is, this process may need to check several data points from various collections in the database. Doing this each time a user like, views or does

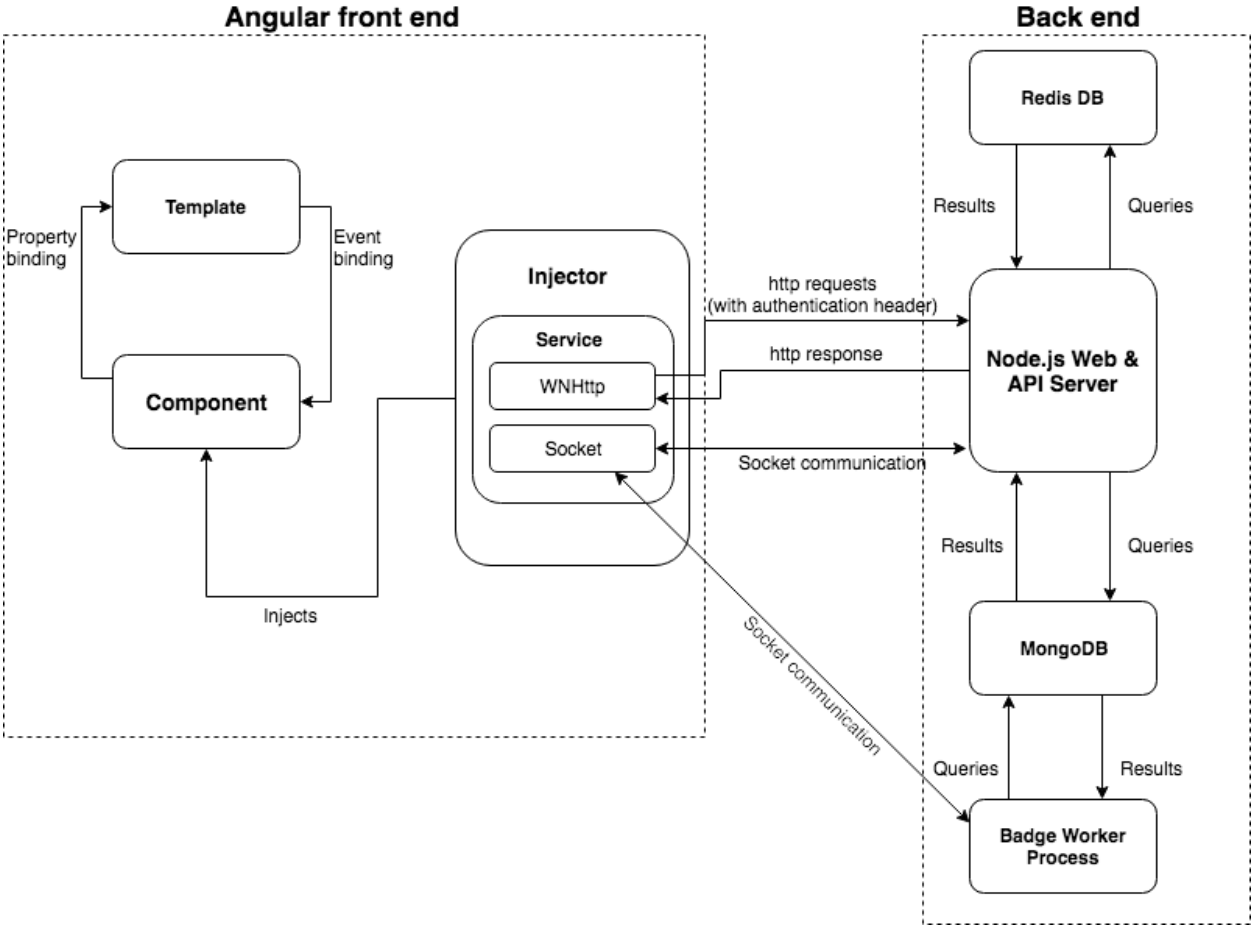

Figure 13: General architecture of WebNovel

any other quantifiable action does not scale. Especially given that this is a synchronous process and node runs on a single thread. One would need to check this in some other, more resource friendly, way. The algorithm we decided to use should scale better. To understand this algorithm we first need two introduce two new data models: "Currency" and "User Currency". Their structures are illustrated in Figure 14. We will first outline the Currency data model. It contains a name field and a description field. This model aims to define an action a user performs that is quantifiable. As an example, in our current implementation we use a Currency with the following information: Name:"likes", Description:"The total number of likes the user has received on all his or her chapters". The idea is that each user can obtain any number of each currency. The number of like currencies a user has reflects how many likes all of his or her chapters has. This quantity is presented using the UserCurrency data model. This contains four fields: "Currency Id", "User Id", "Amount" and "Checked". The

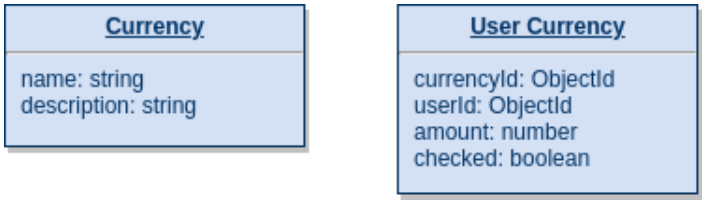

Figure 14: Currency and User Currency objects as presented in database

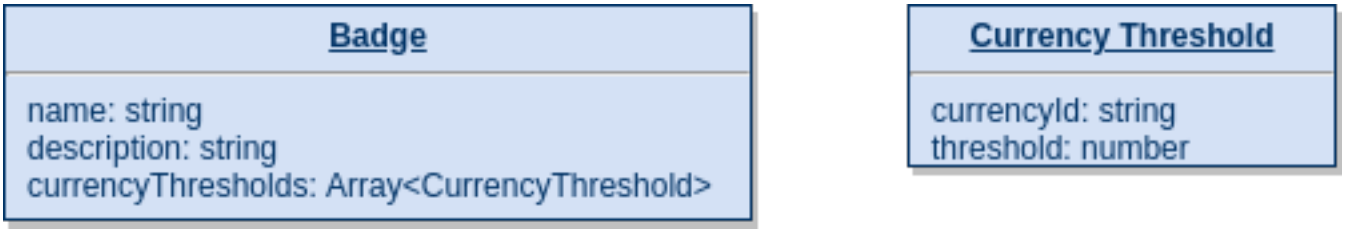

Figure 15: Badge and Currency Threshold objects as presented in database

first three fields are obvious: "Currency Id" contains the id of the currency being quantified, the "User Id" represent the id of the user that has obtained these currencies and "Amount" is the number of the given currency the user owns. We will return to the field "Checked" later when explaining how the system updates the user currencies.

Next we introduce a new data model: Badges. The purpose of this should be self-explanatory: a badge object contains the definition of the milestone that needs to be passed in order to be awarded this badge. The fields are outlined in Figure 15. As can be seen from the figure, apart from having a name and description field, it also contains an array consisting of a new data model called "Currency Threshold". The data model for this object can also be explored in Figure 15.

The purpose of "Currency Thresholds" is to formalize the requirements for a particular badge. Each object in the currency threshold array has a currency specified, in the form of an id from the currencies collection, and the threshold that need to be reached. A badge can contain an arbitrary number of these thresholds. To explain further we will describe a simple example: There exists a badge that is awarded to a user when he or she has obtained both hundred views and hundred likes on a chapter. This is then formalized in the system by creating a badge object containing two thresholds. One specifying the relevant view currency and the number that needs to be reached, i.e. hundred. The other one specifying the relevant like currency and the quantity a user needs to have. This system should make the introduction of new badges trivial.

This summarizes how all the data models are designed. Next we need to explain how the

user currency objects gets updated, and how the system checks whether any users deserves a new badge.

We will first explain how a user currency is updated: We have implemented a general algorithm that is to be invoked with the relevant parameters each time an action incrementing a currency is completed. These parameters are: "currency name", "incrementation" and "user id". The parameters mentioned should be self-explanatory. The first specifies the currency to be updated and the second one specifies how much the currency should be increased. Lastly the user id specifies which user should receive the currency incrementation. After incrementing the relevant user currency, the "Checked" field is set to false. This boolean is used later by the algorithm that awards badges, which is explained later in this section.
This process is invoked every time the relevant action is performed. For example in the API endpoint responsible for adding new comments to a chapter, there is an asynchronous call to this algorithm with the relevant parameters. This means that if one is to introduce a new currency one has to update the codebase as well as adding the desired currency. Although this may seem cumber some we deemed it necessary to obtain the desired functionality. Also, this process is greatly simplified by providing a general algorithm so that all that is necessary is providing the correct parameters in the correct place. Furthermore, instead of actually running an algorithm to verify whether anybody is to be awarded a badge every single time a relevant action is made, we simply update the database.

The last piece of the puzzle is to explain how the algorithm awards the badges. We decided that we wanted to run a separate process independently from the web server. This means that the API calls will not suffer with delayed response time due to a synchronous check that needs to be performed to award badges. The only drawback by doing it this way is that it is a bit less spontaneous. Instead of running each time it might be relevant it now runs every minute (this time period, of course, is arbitrary). Thus, in the worst-case scenario, a user might wait up to one minute before the badge is awarded. However the performance enhancements outweighed the drawbacks in our opinion.
Finishing this section we explain a bit more in detail how this separate process is implemented. We will also touch upon why the "User Currency" object has a "Checked" field.

The algorithm that is responsible for figuring out which users deserve a badge runs, as previously mentioned, every minute. Or, if it uses more than a minute, right after it finishes.

Each time it goes through the following process: The outermost loop, loops through every badge defined in the badge collection. This means it looks at one badge at a time to figure out who is to be awarded with it. A nested loop then loops through all the currency thresholds defined in this badge. For each threshold it will query the "User Currencies" collection for objects having an amount greater than or equal to the threshold. It also adds another requirement in the query, namely that checked is set to false. This helps with performance, as it only cares to check the currencies that actually has been updated since the last time it ran. It then sets the checked field to true. Now, for each user currency object that is returned by the query, the user id is stored in a hash set, which then is added to an array. After this process has been run for every threshold there should exist an array containing a number of hash sets, each containing the user ids of the users having enough currency to pass one of the thresholds. This means that, in order to figure out which users have successfully passed every threshold, we simply need to find the intersection of these hash sets. We have now successfully found which users are to be awarded a badge. It is then simply a matter of storing this information in a collection in the database. Lastly, to let the user know he or she has been awarded a badge we simply use the notification system as explained in Section C.4.

C.3 Authentication

The authentication process consists of two distinct parts: registering a new user and authenticating an existing user. A high level description of these parts are presented in the following sections.

C.3.1 Registering new users

When visiting the registration page, the user is presented with a blank registration form. Additionally there is a recaptcha widget [32] to ensure that the user is human and not a bot. Their desired username, password and other information is then sent to the API endpoint responsible for new registrations. After validating the fields and ensuring that the email is unique and not already a part of the user collection, the password is hashed. The hashing is an extra security step to ensure that, in the unfortunate event of a database leak, the passwords cannot be extracted as plain text. Our hashing algorithm uses bcrypt [33] because of the advantages it has over more traditional hashing algorithms. Among the advantages is the technique to have the rekeying rounds configurable. This way one can make it arbitrarily

slow to decode, thus discouraging brute-force attacks. Secondly it uses a salt. Put shortly, a salt is a changing value that is used as a seed for the hashing function so that two passwords that are equal in plain text will not have an equal hash value. This makes it resistant to something like a rainbow table attack, in which an attacker would use a precomputed table of plain text and their corresponding hash values to extract passwords.

C.3.2 Authenticating existing users

To authenticate requests from users we use JSON Web Token (JWT). JWT is "an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed." [34] Such a token contains three pieces of information: First there is a header containing meta information. Next is an arbitrary payload containing whatever information the developer chooses to. The last part is a digital signature. This is what makes this technology ideal for authentication uses. Whenever a user successfully logs into the website her or she receives a token that not only contains user information such a as name and email, but also a signature. This signature uses the header, the payload and a secret only known to the server, as seeds to get generated. Thus, due to the use of the secret, the server is the only actor capable of providing the signature. After a client has received this token it is set as a header so that it will be available in every request made. For every request that requires the user to be authenticated, the signature goes through a validation process to determine its validity.

C.4 Notifications

Each user connected to the website, has a socket connection to the web server. In order to understand how the notification system is functioning we must first explain how a user id is mapped to the relevant socket. When the server is started, part of the setup process is setting up event handlers for Socket.IO. Socket.IO automatically emits events both when a new socket is setting up a connection and when it disconnects. In addition to these we also take advantage of custom events. We use an event with the name "user-auth". This event is emitted from the client immediately after the socket connection is set up, given that they are logged in. If not, the event will be emitted after the user has logged in. This event sends a payload containing the authentication token. On the server side there is an event handler set up for to handle this event. It uses the process described in Section C.3 to verify that

the user is authenticated. If the authentication is successful, the user id gets extracted from the payload part of the JWT. We also have access to the id of the socket emitting this event, hence we can insert the mapping into the Redis database. Thus we have a mapping from the user id to the socket id which allows us to send messages to specific users.
A second event handler is run when a socket emits the "disconnect" event. This event is emitted automatically by the client when the socket is disconnected. The handler does a quick cleanup, by deleting the mapping from the Redis database, so that only the relevant connected sockets is mapped.
This summarizes how the system is able to send notifications to a user by only knowing their user id. Next we go into more detail as to how the notifications are propagated through the system.

A notification is represented in the database as illustrated in Figure 16. The "Actor Id" field holds the id of the user performing the action that warrants a notification, while the "Subject Id" field contains the id of the user that is to receive the notification. The "Object Id" represents the id of the object that has been interacted with, this could be something like a comment or a chapter. The type of object the id is referring to is stored in the field "Object Type". The "Verb" field simply contains a string describing what the actor did to create the notification i.e. "commented on" or "liked". Lastly the "Read" field holds information about whether the notification has been read by the subject or not. Whenever a user completes an action that warrants a notification the following process happens: First the actual action is completed, for example saving a comment in the database. After this a notification object is created containing the relevant information. This object is then saved in a Notification collection in the database. When the insertion is completed a shared static function called "emitMessage" is invoked. This function receives a user id, an event and the payload that is to be sent as parameters. In this case the user id is the subject id from the notification, i.e. the receiver, the event is "notification", and the data is the notifications object. On the front end the client has set up an event handler for when a notification event is emitted. Thus the client receives this notifications and handles it appropriately, showing some visual feedback to the user. We have now described how a user receives a notification in real time if he or she actually is logged in, but we still need to provide an explanation as to how the user receives notifications that are created when the user is offline. The user should, upon log in, be presented with all notifications that he or she has received while being logged out. This is achieved by an API-endpoint that returns every notifications that

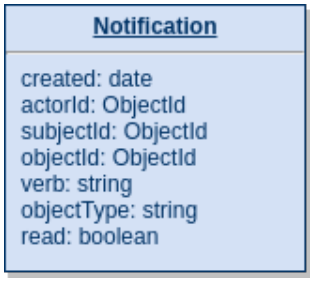

Figure 16: A notification object in the database

has the "Read" field set to false. Thus, upon login, the client sends a request to this endpoint and in turn receives the notifications created when the user was offline. Lastly, whenever the user opens up the list of notifications, each notification is updated with "Read" set to true. Putting all these parts together thus provides us with a fully functional real time notification system.

C.5 Graph visualization

At first we used D3's tree function to visualize the book graphs. However when the decision was made to change the representation of each book to a network instead of a tree, we could no longer use this approach directly. D3 has the functionality to visualize both undirected and directed graphs, however these all result in a somewhat random positioning of the nodes. We needed the graph to still be orderly, as there exist a proper order of which chapters should be read before others. Hence we wanted a similar visualization to the former tree structure, but also allowing for edges between nodes that would not be allowed in a tree. This problem was solved by going back to using D3's tree functionality. The way we were able to do this was the following:

We start by creating a tree structure with the initial chapter set as the root node. Then we iterate through every node in the graph. For each node we look at the outwards pointing edges and the node in the other end. These nodes are added to the children array for the node we are currently looking at in the iteration. The id of these children are then added in a hash set. When the iterator moves on to the next node we check which nodes on the outward edges are not yet in the hash set, i.e. that has not yet been added as children to another node. If there are any that are not added to the tree yet, these are added as children to the current node. This continues until every node is in this hash set. If we now feed this tree structure into D3's tree function it should output data points that is somewhat orderly positioned. Lastly we need to paint the edges between these nodes. This is done by iterating

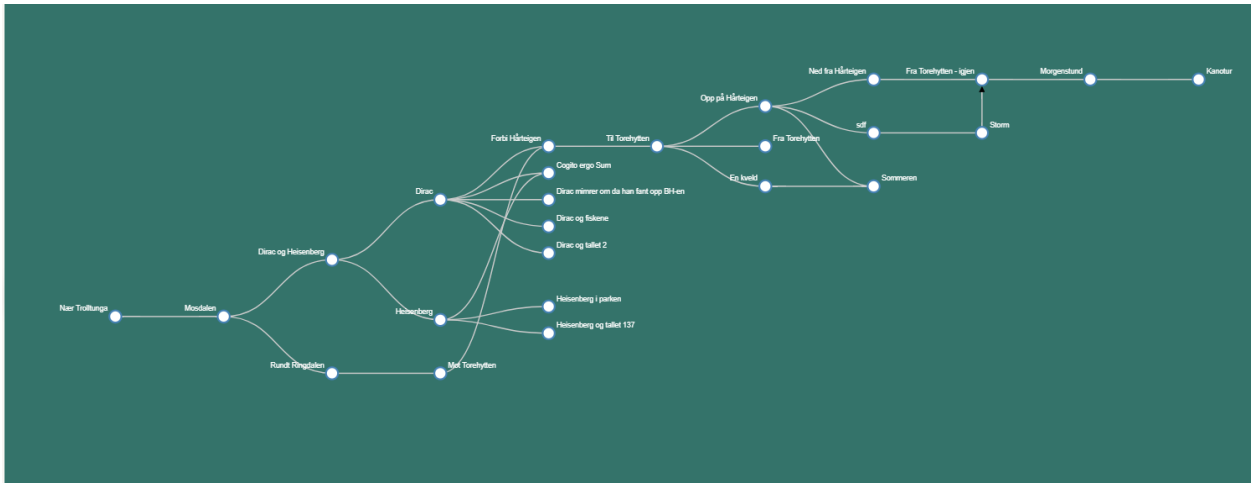

Figure 17: Screen shot of the version as of this writing

through the original edge array of the graph. Each edge contains the id of the nodes on either side. Thus we can find their current position by finding the nodes with the given ids in the newly created array of positioned nodes. The start and end position is all we need to know to draw our line between the two nodes. The final result can be seen in Figure 17.

D Survey results

This section presents the answers received from the online survey. The following figures are all the unedited answers from the survey. The answers marked with * are translated from Norwegian.

Figure 18: Answers to the question: What was your first impression when visiting the website?

1. The layout and design is pleasing to the eye.
2. I liked the colors and the layout. It makes me want to explore the page. Navigating the application was okay, I understood that the smart thing to do first would be to create an account *
3. Easy to navigate around the application, interesting concept, but I think there should exist more information about how to create a new path.*
4. It was a cool design and I liked the overall idea.
5. Nice frontpage explaining the idea.
6. Very nice website.*
7. Orange. But otherwise easy to understand!*
8. Clear and straightforward.*
9. Easy to use. It might benefit in having a more exciting design/look.*
10. Positive. I quickly understood what the site offered. I also thought the landing page was clean and easy to understand.*
11. Simplistic and fresh look, looked clean and proper.*
12. The site seems simple, streamlined and easy to navigate.
13. That it had a welcoming design and that it was exciting to be able to read the stories of others. Also it was easy to understand. The only thing that I found hard was that it did not work so great on a phone (iPhone 6). PS: there might be something I don't realize how to do in regards to what I mentioned last.*
14. Not very mobile friendly, but looks good on pc
15. Cool concept. It was a clean and good page.
16. Okay, the design wasn't all that exciting.*
17. Good*
18. Interesting*
19. Clean layout and straight forward

Figure 19: Answers to the question: Explain shortly what sets WebNovel apart from traditional novel writing tools.

1. The forking paths are essential to the WebNovel, allowing collaborative writing combined with interactivity.
2. The fact that it is on the internet. Others are able to see your work, and one can receive feedback. Exciting that others can read it.*
3. The fact that I can create my own paths if you feel like a story can have several viewpoints.*
4. Webnovel is mote interactive and it opens up for a different narrative that fit for different people.
5. Writing with others to collaborate on the same book with many stories.
6. I have no experience with other tools.*
7. I have not been exposed to other forms of a webnovel before.*
8. Easier to use.*
9. Unsure, I've never used anything similar.*
10. I found it very easy to create a book and add chapters. It was easyto read what others had written. Also I really liked the fact that it was possible to discuss each chapter.*
11. Feels easy to use, few things that can be confusing.*
12. The way WebNovel gives you the opportunity to co-create and take part in other peoples stories is what mainly sets it apart from other writing tools. You can make your own stories and give the reader a way to choose what direction to take in the story by making different continuations. But other people can do this as well, and build further on your works, or make alternative storylines for your novel. This creates a freedom in writing that is not seen on a larger scale before.
13. I found the website easy to use. I have not tested any similar websites.*
14. It let's the users cooperate on stories, and by allowing multiple "next chapters"", give stories an extra dimension, by turning a storyline into a "storytree"
15. It can be edited by others. Easier to write a story. More creative than traditional novel writing.
16. Haven't tried a lot of others.*
17. Haven't tried any other sites.*
18. Nice design, easy to navigate.*
19. I have no experience with other novel writing tools

Figure 20: Answers to the question: What did you like about the application?

1. I like what the application promises, the idea is solid.
2. I liked the colors. It was interesting because it was something new, something unlike anything I've used before.*
3. I like how the page is structured, and how you can build on and comment on the different books.*
4. I liked the idea and I also liked the design. It opens up for some fun and great stories.
5. Fun to write and come up with stores with many different paths.
6. The fact that you can build upon the stories of others. *
7. The fact that you can get inspired and read the novels of others. Additionally one can create novels where others may give feedback.*
8. It was easy to use and clean.*
9. It was easy to use, easy to register.*
10. The clarity and the colours.*
11. Very easily accessible, one click and you are view the thing you wanted to read/look at, easy and clean.*
12. The sense of freedom and the possibility to make your own story or "spin-off" of the stories are a nice feature, and i think it is very welcome in the world of Fan-fiction
13. I liked the fact that it was easy to use, and that was very understandable. The instructions were also clear in how to log in, create an account and use the website.*
14. It's simplicity
15. I think it is nice that it challenges the idea about what a book is, that there is not just one ending to a story.
16. Simple buttons and headlines.*
17. Easy to get started. The explore book functionality seems nice.*
18. Easy to know where to click, (easy access).*
19. The "explore book" tool

Figure 21: Answers to the question: What did you not like about the application?

1. The execution is riddled with shortcomings that renders the experience arduous. While the flaws are not enormous, they make the experience of inserting and navigating chapters annoying.
2. It was hard to understand how one were to navigate around the page. Hard to understand how one should accomplish different task. I.e. how one can create a new path in the graph. There are instructions on the sidebar, but I can't be bothered to read it.*
3. How to create a new path could be explained better, but I got it eventually.*
4. That it was a bit difficult to understand that you should make drafts before you create a new path.
5. Very confusing to connect the different paths and chapters.
6. Some aspects might be a little unintuitive, among others "Create New Path" and "Continue Story".*
7. What I didn't like about the application is that it is hard to figure out if it is possible to delete chapters and books, if it is possible at all.*
8. Did not manage to change the title of the book afterwards.*
9. That you full name is shown if you input it when you create an account.*
10. I'd wish thath were some form of inspiration pages. More genres for the different books one were to write would also be good.*
11. I did not manage to leave a comment on books/chapters, and the "choose your next chapter" box looked a little weird. Didn't really blend in with the rest of the design of the page.*
12. The site in itself was a little bland and boring, the colours weak and the missing option do add illustrations to your stories would be nice. Also the site is not optimised for handheld devices (only tested on android) and some of the functions did not work on android
13. Just that it didn't fit phones that well? But there might be something I'm not getting.*
14. It's responsiveness in accord to screen size
15. Cannot think of anything now
16. Hard to find all of the chapters after they were written.*
17. A little hard to navigate through the chapters at first, but it gets easier after trying it for a bit.*
18. Don't know.*
19. Nothing that comes to my mind.

Figure 22: Answers to the question: What do you think should be improved or be removed from the application?

1. The user experience should be made easier and more streamlined.
2. The graph could have been simple to use. The graph should be prettier. The graph is a little bit boring.*
3. One could have the ability to respond directly to another comment.*
4. That you should be able to easily make drafts when creating new paths.
5. The logic for connecting paths should be improved.
6. Maybe it should be a little more streamlined when it comes to building upon a story.*
7. Easier to understand what some things means. Like pen name, I had no idea what that meant. Maybe some description or another word would be easier to understand. And a delete button would be great.*
8. The ability to fix or change a book title.*
9. The fact that your full name is displayed (alternatively a question if you want to show all of it.)*
10. As one is able to receive likes, it would be nice to be able to see who one has gotten likes from so that one might contact them, as they have already shown interest in what one has done.*
11. Unsure, but maybe change the design of the "choose your next chapter" box.*
12. The possibility to add pictures or illustrations to stories and "bookcovers" could be implemented, and the mobile-site fixed or an app created. Also the navigational buttons are a bit hidden, and there should be a way to sort stories by genre, language, etc
13. That I do not know.*
14. The scroll position is conserved between homepage and readpage, which feels weird when you are scrolled far down on homepage
15. Cannot think of anything now
16. Even easier to navigate, bigger fonts and headlines.*
17. It misses comedy?:P *
18. I don't know *
19. I would like the ability to choose what type of genres I want to read in the discover tab.

Figure 23: Answers to the question: Do you find the application fun or boring to use? Why?

1. It was okay. It's just writing, you can't really make it more engaging.
2. Fun. The concept is new, so I want to try it out and learn more about it.
3. Fun, because it is possible to get a varying reading experience.*
4. No, it was fun.
5. I have never written stories with different paths before. I definitely liked that. Its also fun to read stories from others and contributing with your own paths.
6. Fun to use, much because of all the weird things others have written.*
7. Fun and exciting.*
8. Fun to use if one sits down to write properly.*
9. Fun. Get to be creative.*
10. It was fun to write. Precisely because it was easy and one could share it with others. Of course one gets nervous about the fact that others may read it, but as this is the case for everybody I find this positive.*
11. FUN! I see the potential in the application, perfect for both reading, writing and sharing with others.*
12. I found the application fun to play around with and to read through other stories and the way people creates their different versions of stories.
13. I found it very fun! I liked that the stories from others are so easily accessible. It was fun.*
14. It's fun to use your imagination and cooperate with others.
15. It was fun.
16. fun
17. For those who enjoy writing, this is probably a perfect site.*
18. Fun! Because the design was good and easy to figure out.*
19. Fun. I like the navigation through the "world"

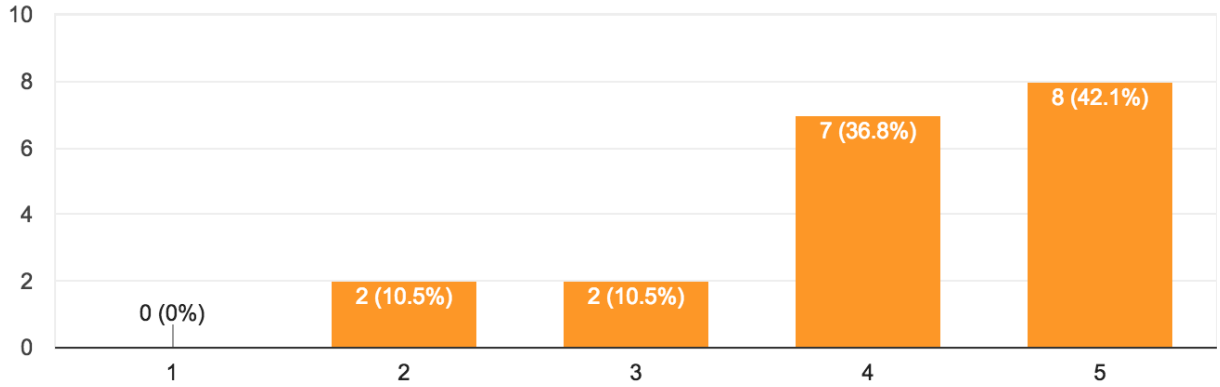

Figure 24: Answers to the question: How easy is the application to use? (Where 1 is very hard to use and 5 is very easy to use)

Figure 25: Answers to the question: What functionality do you think we should add?

1. The mapping of the chapters and how they interact with each other should be easier to use, maybe something akin to Twine, where creating the chapters and connecting them is a breeze.
2. That you can listen to the book like an audio book, to make it available to more people.
3. To add more genres to each book.
4. It would probably be cool if one could choose a particular path in the novel-tree and then export it to a PDF. (This might possibly already exist, however I did not find it)*
5. I think it should be possible to delete things.*
6. I really miss a sorting tool for the novels/stories
7. Maybe add some book covers

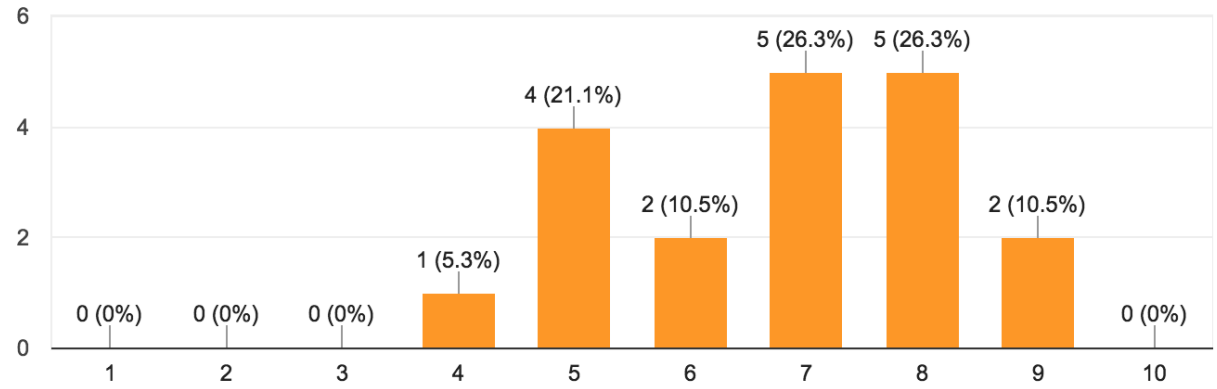

Figure 26: Answers to the question: What score would you give if you were to rate WebNovel from 1 to 10, where 1 is garbage that you never want to use again, and 10 is that you would gladly visit the application daily?

E Search terms

The list presented here are the search terms we used in an attempt to find similar research.

- novel collaboration
- graph novel
- novel as network
- collaborative fiction
- collaborative writing platform
- collaborative writing tools
- collaborative novel web
- fiction novel network structure
- story network
- story graph structure

References

[1] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of management information systems.*, vol. 24, no. 3, pp. 45–77, 2007.

[2] "Twine," retrieved 19. April 2018. [Online]. Available: https://twinery.org

[3] "Wattpad," retrieved 19. April 2018. [Online]. Available: https://wattpad.com

[4] "Google docs," retrieved 19. April 2018. [Online]. Available: https://docs.google.com

[5] "Mongodb for giant ideas," retrieved 17. April 2018. [Online]. Available: https://mongodb.com/

[6] "Json," retrieved 12. May 2018. [Online]. Available: https://json.org

[7] W3C, "Introduction to html," retrieved 24. March 2018. [Online]. Available: https://www.w3schools.com/html/html_intro.asp

[8] ——, "Css introduction," retrieved 20. May 2018. [Online]. Available: https://www.w3schools.com/css/css_intro.asp

[9] ——, "Javascript html dom," retrieved 20. May 2018. [Online]. Available: https://www.w3schools.com/js/js_htmldom.asp

[10] "About javascript," Mozilla, retrieved 19. May 2018. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript

[11] Microsoft, "Typescript," retrieved 22. March 2018. [Online]. Available: http://www.typescriptlang.org/

[12] Google, "Angular," retrieved 22. March 2018. [Online]. Available: https://angular.io

[13] "Ajax - developer guides," Mozilla, retrieved 19. May 2018. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX

[14] "Angular - routing & navigation," Google, retrieved 19. May 2018. [Online]. Available: https://angular.io/guide/router

[15] Google, “Angular - angular dependency injection,” retrieved 23 May 2018. [Online]. Available: https://angular.io/guide/dependency-injection

[16] W3C, “Svg tutorial,” retrieved 20. May 2018. [Online]. Available: https://www.w3schools.com/graphics/svg_intro.asp

[17] M. Bostock, “D3 - data-driven documents,” 2017, retrieved 13. April 2018.

[18] ——, “d3-3.x-api-reference,” retrieved 19. May 2018. [Online]. Available: https://github.com/d3/d3-3.x-api-reference/blob/master/Tree-Layout.md#tree

[19] Socket.IO, “Socket.io,” retrieved 18. March 2018. [Online]. Available: https://socket.io/

[20] ——, “Socket.io - rooms & namespaces,” retrieved 19. May 2018. [Online]. Available: https://socket.io/docs/rooms-and-namespaces/

[21] S. Sanfilippo, “Redis,” retrieved 20. May 2018. [Online]. Available: https://redis.io

[22] N. Foundation, “Node js,” retrieved 22. March 2018. [Online]. Available: https://nodejs.org

[23] Docker, “What is docker?” retrieved 20. May 2018. [Online]. Available: https://www.docker.com/what-docker

[24] Wikipedia, “Linux containers,” retrieved 27. March 2018. [Online]. Available: https://en.wikipedia.org/wiki/Linux_containers

[25] J. Somers, “5 famous novels that have huge mistakes,” retrieved 22. March 2018. [Online]. Available: https://www.barnesandnoble.com/blog/5-famous-novels-huge-mistakes/

[26] Merriam-Webster, “Gamification,” retrieved 11. April 2018. [Online]. Available: https://www.merriam-webster.com/dictionary/gamification

[27] ——, “Public domain,” retrieved 20. May 2018. [Online]. Available: https://www.merriam-webster.com/dictionary/public%20domain

[28] “Attribution 3.0 unported (cc by 3.0),” Creative Commons, retrieved 03 May 2018. [Online]. Available: https://creativecommons.org/licenses/by/3.0/legalcode

[29] “Attribution 3.0 unported (cc by 3.0),” Creative Commons, retrieved 03 May 2018. [Online]. Available: https://creativecommons.org/licenses/by/3.0/

[30] "The mit license," Open Source Initiative, retrieved 03 May 2018. [Online]. Available: https://opensource.org/licenses/MIT

[31] S. Overflow, "Badges - stack overflow," retrieved 22. May 2018. [Online]. Available: https://stackoverflow.com/help/badges

[32] Google, "recaptcha: Easy on humans, hard on bots," retrieved 23. May 2018. [Online]. Available: https://www.google.com/recaptcha/

[33] N. Provos and D. Mazieres, "A future-adaptable password scheme," retrieved 23. March 2018. [Online]. Available: https://www.usenix.org/legacy/events/usenix99/provos/provos_html/node1.html

[34] Auth0, "Json web token introduction," retrieved 19. March 2018. [Online]. Available: https://jwt.io/introduction/