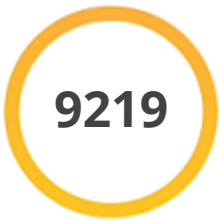
Raport de verificare a conținutului plagiat

Scanned on: 12:54 June 24, 2022 UTC

Overall Similarity Score

Results Found



Total Words in Text

Identical Words	118
Words with Minor Changes	71
Paraphrased Words	119
Omitted Words	0

Raport de verificare a conținutului plagiat

Scanned on: 12:54 June 24, 2022 UTC

Results

Sources that matched your submitted document.

Building Imaginary Worlds: The Theory and History of Subcr... *https://dokumen.pub/building-imaginary-worlds-the-theory-and-history-o...* building-imaginary-worlds-the-theory-and-history-of-subcreation-201...	1%
Building Imaginary Worlds - PDFCOFFEE.COM *https://pdfcoffee.com/building-imaginary-worlds-pdf-free.html* building-imaginary-worlds-pdf-free.html	1%
React: The Virtual DOM | Codecademy *https://www.codecademy.com/article/react-virtual-dom*	1%
World Anvil Worldbuilding tools & RPG Campaign Manager |... *https://www.worldanvil.com/*	1%
'The Subtle Art of Worldbuilding' by Jennifer Harwood Smith... *https://blog.bham.ac.uk/poplit/the-subtle-art-of-worldbuilding-by-jennifer...*	1%
Cloud live streaming System based on Auto-adaptive Overla... *https://dl.acm.org/doi/10.1145/2962564.2962571*	1%

IDENTICAL

Identical matches are one to one exact wording in the text.

MINOR CHANGES

Nearly identical with different form, ie "slow" becomes "slowly".

PARAPHRASED

Close meaning but different words used to convey the same message.

Unsure about your report?

The results have been found after comparing your submitted text to online sources, open databases and the Copyleaks internal database. For any questions about the report contact us on support@copyleaks.com

Learn more about different kinds of plagiarism here

Raport de verificare a conținutului plagiat

Scanned Text

Your text is highlighted according to the matched content in the results above.

IDENTICAL MINOR CHANGES PARAPHRASED

SUMMAR
Sinopsis
Abstract
Acknowledgements
1 Introduction1.1 Context
1.2 The problem
1.3 Objectives
1.4 Thesis structure

2 Requirement analysis and specification2.1 Target audience
2.2 Use cases

3 Market Research3.1 World Anvil
3.2 Legend Keeper
3.3 Notebook.ai

4 Existing technologies4.1 Front-end technologies
4.2 Back-end technologies
4.3 Database management systems

5 Suggested solution5.1 The MEAN Stack architecture
5.2 Application architecture5.2.1 Front-end application
5.2.2 Back-end application
5.2.3 Database

6 Implementation details
7 Evualuation
8 Conclusions and subsequent developments
9 Bibliography
10 Appendix

UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE DEPARTMENT
DIPLOMA PROJECT
Lore Map: A web tool for writing and organizing a fantasy universe
David Cornelia-Gabriela
Thesis advisor:
Conf. Dr. Ing. Anca Morar
BUCHAREST
2022

SUMMAR
Y

SINOPSIS
Crearea unui univers fantastic cu istorie, geografie și persoanje coerente și credibile poate fi dificil pentru scriitorii amatori. Acest proiect propune o soluție care ajută la organizarea unei lumi imaginare complexe, permițând crearea de colecții de documente structurate, hărți interactive, cronologii cu evenimente istorice și diagrame de relații între personaje. Documentele sunt bine conectate prin hyperlink-uri. Soluția este o aplicație web construită cu tehnologii populare: Angular pentru partea de client, Node.js și Express.js pentru partea de server și MongoDB pentru gestionarea datelor.
ABSTRACT
Developing a fantasy world with coherent and believable history, geography and characters may prove challenging for amateur writers. This project proposes a solution that helps with organizing a complex imaginary world by allowing the creation of structured document collections, interactive maps, historical events timelines, and character relationship diagrams. The documents are well-interconnected via hyperlinks. The solution is a web application built with popular technologies: Angular for the client-side, Node.js with Express.js for the server-side, and MongoDB for data management.

ACKNOWLEDGEMENTS
I would like to thank Professor Anca Morar who offered valueable guidance and feedback that has helped me in completing this project.

1 INTRODUCTION
1.1 Context

The modern fantasy genre has been around since the 18th century [1], only becoming more popular and entering the mainstream in the late 19th century. J. R. R Tolkien's The Lord of the Rings established the epic fantasy subgenre and influenced many fantasy writers. It was then that the fantasy genre could be seen as a profitable market. [2], [3]

As the fantasy genre has considerably grown worldwide, so did other forms of fantasy besides literature such as movies, video games, and board games. Notably, fantasy lead to the release of the first-ever role-playing game, Dungeons & Dragons which amassed immense popularity. The game entails creating a character and going on adventures in a fantasy world created by a separate player, the Dungeon Master [4]. This represented the precursor of role-playing games (RPG) such as The Elder Scrolls series, The Witcher series, Final Fantasy franchise, and many more.

Given the popularity of the fantasy genre in multiple industries, many creative individuals wish to build fantasy worlds, whether they might be for books and novels, video games, or tabletop role-playing games such as the one mentioned above.

1.2 The problem

Building a credible fantasy world is difficult, considering how complex ideas can get. Mark J. P. Wolf mentions in his book "Building Imaginary Worlds": "Imaginary worlds, built of words, images, and sounds, can be tremendous in size; for example, as of summer 2012, theTStar TrekTuniverse consisted of over 500 hours of television shows, 11 feature films, and hundreds of novels, not to mention several decades worth of video games, comic books, and other books including technical manuals, chronologies, and encyclopedias. And since it is an open-ended and still-growing universe, moreTStar TrekTmaterial appears every year. Worlds of this size, even closed ones that are no longer being added to (though they may still be adapted and interpreted), can often be difficult to see in their totality, and much time must be spent to learn enough about a world to get an overall sense of its shape and design. In this sense, an imaginary world can become a large entity that is experienced through various media windows ... Experiencing an imaginary world in its entirety, then, can sometimes be quite an undertaking. An imaginary world is more difficult to encapsulate in a description or analysis than a particular story, character, or situation." [5]

The geography, political context, and overall history of a world should be understood not only by the creator but by the audience as well. As a result, many amateur writers and their readers are overwhelmed by the complexity of the universe and the chaotic structure it might have at first. Having played Dungeons & Dragons and seeing that creating a world for the players is not an easy feat, I was inspired to develop a web application to help creators with laying out all the information of their world in an understandable manner. The application is built with the following technologies: Angular for the frontend, Node.js for the backend, and MongoDB for the database.

1.3 Objectives

The project proposes a content management web application for creating a fantasy world that is well structured and interactive via hierarchies of documents that are interconnected by hyperlinks, and interactive maps. The application also has an interface that is easy to use. With this tool available on the Internet, creators will be more encouraged to start building their fantasy worlds and also share them with their audience.

1.4 Thesis structure

The thesis discusses all the development stages of a web application. Chapter 2 will outline the specifications of the application based on the target audience and questionnaire results. Chapter 3 will analyze similar products on the market and compare them to this project. Chapter 4 will describe popular web technologies and list a few advantages and disadvantages and why certain technologies were chosen for the development of this project. This chapter will also motivate the choice of certain technologies. Chapter 5 presents the architecture of the application on different levels: front-end, back-end, and database. Chapter 6 describes the implementation of the application in more detail by analyzing snippets of code from core features. [...de continuat la 7]

2 REQUIREMENT ANALYSIS AND SPECIFICATION

2.1 Target audience

The main target audience for this application is represented by players of the game Dungeons & Dragons, especially Dungeon Masters whose role is to create the world in which the events of the game unfold. However, the application's functionalities can be useful for any kind of writer, and the target audience can easily be extended to fantasy writers and game designers.

2.2 Use cases

Before starting the implementation phase, I thought about what core features I would want the application to have and also what any web application needs. Next, I would conduct research regarding the features of the application and how demanded it would be.

What a user of this application primarily needs is the ability to write and save parts of the story on a multitude of documents and link the documents together by referencing keywords. For example, a user who wrote a document about a certain country in which they metion the name of a character can attach a link to the character's name wich leads to another document with details about the character. A user must be able to format their data and add pictures.

In addition, a user should have a personal account that safely holds the information that they wrote. Their credentials should be safely stored in a database and their passwords encrypted. The application mustn't be subject to attacks.

Besides writing and saving text, the user should also have the ability to provide maps which then they can interact with by adding pins that can link to certain documents themselves.

I have created a questionnaire that has 48 answers from students. I firstly wanted to assess how popular this application would be. Among the students, 80.9% have never used a similar application and out of these 76.6% would use one. Of the students, a majority of 62.2% would use the application for role-playing games, and the rest would use them for book writing or both. As the results suggest, applications such as this are in demand.

Additionally, the students have been asked about how relevant the features described above are. They were asked to rate from 1 to 5 how relevant they consider the features to be, 1 being least relevant and 5

being most relevant. The results have shown that the majority consider the features relevant, most of the answers having above a 3.
Finally, the students were asked what additional features they would like the application to have. 87.2% would like to be able to create timelines, 85.1% would like to visualize the relationship between characters and 57.4% would like to create family trees for their characters.
Following the results of the questionnaire, I elaborated on the following features of the application:
-The user must create a personal account to use the application
-A user can change the password if they have forgotten it and they will receive an email that prompts them to do as such.
-The user will receive an email with the confirmation of the password change.
-The user can add multiple worlds with different stories and they can see a list of these worlds.
-Within a world, the user has a main document page available in which they can write text, format it, and also add images.
-A user can add multiple documents such as the main document which they can group as desired in folders. The hierarchy of files is always displayed.
-A user can add documents
through the designated menu, by clicking on the „Add" button
by right-clicking on words for which the page will be created
which will also attach a hyperlink on the word that links to the new document.
-A user can create multiple maps by uploading images of maps and adding markers with specific information.
-A user can see a list of all the maps that they have created
-A user can select a word in a document and place a hyperlink to another page that is relevant for that word.
-When a user clicks a hyperlinked word, they are taken to the document that the user chose.
-A user can create timelines and can add cards with important events and their historic dates. The timeline is displayed vertically.
-The events in the timeline can also link to written documents.

-When adding a document, a user can specify that the document is a special character document that should only contain details about a character. When creating this type of document, there will be a prompt asking for the name of the character and the relationship it has with other existing characters.
- A user can create a graph that displays the relationship between characters. The nodes will be the characters and the links will be lines of different colors depending on what type of relationship the characters have: good, neutral, or bad.
-The user can delete any of the structures created: a map, a timeline, a relationship graph, a document, a folder
The use cases can be seen in the diagram in Figure 1:

3 MARKET RESEARCH

The idea behind this application is not new. Several web applications have built a solution. This chapter presents the advantages and disadvantages of a few of them compared to this project.

3.1 World Anvil

Fig. 1: UML Use Case Diagram of the application

World Anvil is a set of worldbuilding tools that helps a user create, organize and store their world setting. With wiki-like articles, interactive maps, historical timelines, and full novel-writing software, they have all the tools needed to run an RPG campaign or write a novel [6].
This application has a great variety of features, which include: organization charts, family trees, diplomacy webs, and the ability to create entries of different categories such as military conflicts, religions, myths, and species. The dashboard of this application can be seen in Figure 2.
Despite the number of features that this application offers, it is difficult to use. There are tutorials at each step and the whole process can be tiring and overwhelming for someone new to the platform. Lore Map has fewer features and it is not as complex as World Anvil, but it makes up for it with an intuitive interface and lack of tutorials. In addition, World Anvil has multiple subscription plans and the standard version lets a user build only 2 worlds and write a limited amount of articles, while Lore Map allows its users to build unlimited worlds and write unlimited articles. World Anvil also displays ads that the majority find bothersome and you have to pay for the subscription for them to disappear. Lore Map will not display any ads.
3.2 Legend Keeper

Fig. 2: World Anvil dashboard

Legend Keeper [7] is a modern web app that makes it easy to design, build, and share worlds of any kind of story. The app advertises itself as „the fastest and most flexible worldbuilding app you'll ever use" This application has a clean, simple, and pleasant interface all while it doesn't lack in functionalities. It doesn't enforce any limits on creation or storage space. An impressive feature is the auto-linking pages, a button that generates links between pages by detecting the names through the wiki pages. The application also offers the ability to create boards with text, shapes, sticky notes, and even links to pages. A preview of this feature can be seen in Figure 3.
Another great benefit is that besides the ability to create interactive maps, the application also offers a selection of high-quality maps from top cartographers so that users who don't have a map yet can start from one of those. This feature can be observed in Figure 4.

Despite the clean interface and useful features, the application doesn't provide timelines and relationships overview. In addition, the application is only available for 14 days, after which the user has to pay a subscription to continue creating. Lore Map on the other hand offers unlimited access to its features.
3.3 Notebook.ai
Notebook.ai is a smart notebook for worldbuilders [8]. World creation takes the form of blocks, like character blocks, location blocks, and item

Fig. 4: Map feature in Legend Keeper
Fig. 3: Board feature in Legend Keeper

blocks. Each block has a different configuration with a specific list of categories. The template for these blocks can be customized. For a character, the user can complete information about looks, personality, family, inventory, and others. An item has categories about its appearance, its history, and its abilities. If a user wants to link a character to a certain item, they can go into the character's inventory category and select the item that was previously created from a dropdown menu. A unique feature is the Dashboard that asks you questions to help expand the world and displays the recently edited blocks. The dashboard can be seen in Figure 5.
The interface is simple and easy to use and the blocks are color-coded which makes them easier to distinguish. Notebook.ai seems to have a different way of organizing data compared to the other 2 applications. The limitations of this application come from the limited features available unless the user pays for a subscription:
-Only 4 kinds of blocks available: universe, location, character, and item
-Image upload limit of only 50Mb which of course limits the usage of high-quality images
-A user can create up to 5 universes
In addition, besides enumerating the links to other blocks, it lacks any visual representations of the data. It doesn't have any features for interactive maps, timelines, or relationship graphs, unlike Lore Map. There is also no limit to image storage on Lore Map.

4 EXISTING TECHNOLOGIES

The application is built using the MEAN stack (MongoDB, Express.js, Angular, Node.js). This chapter will analyze alternative technologies that the application could have used and will motivate the choice of these technologies. The chapter will be divided into sections dedicated to front-end technologies, back-end technologies, and databases.

4.1 Front-end technologies

Angular is considered one of the top three most popular front-end frameworks today, the other two being React.js and Vue.js [9]. This section will first describe all of the frameworks briefly, mentioning their advantages and disadvantages.
React.js [10] is an open-source Javascript library for building user interfaces. It was developed by Facebook and first released in May 2013. React is component-based, meaning that users can build encapsulated components that manage their own state and then compose them to make complex user interfaces (UIs).
Figures 6 and 7 show two equivalent examples of component definition, the first one using a function and the second one using an ES6 class [11].
Before version 16.8 was released and lifecycle Hooks were introduced [12], class methods were used for components that had their own state, while function components were used for stateless components [13]. Today, state management can also be used in function components

Fig. 6: React function component
Fig. 7: React class component

and users prefer function components because they are written shorter and simpler and are easier to develop, understand and test [14].
Document Object Model (DOM) manipulation is expensive, and React was designed to optimize this through state management and virtual DOM. The virtual DOM is a lightweight copy of the real DOM which is faster to manipulate. When a component is rendered, every object from the virtual DOM is updated, then React compares the updated version of the virtual DOM with its previous version before the update. This way, React knows exactly what DOM objects were updated and will update only the modified objects in the real DOM [15]. The usage of the virtual DOM makes React update faster, but it consumes additional memory because the copy of the virtual DOM tree must be stored [16].
The main advantages of the React.js framework are: [17]–[21]
Easy to learn
Virtual DOM produces good performance
Components can be written as functions that are easier to write
Component reusability
One-way data binding gives developers better control
The disadvantages of this framework are:
Constant updates make for a poor documentation
React is a library, not a framework and it needs companion libraries for problems such as routing and state management
Cascadin Style Sheets (CSS) are globally scoped, meaning that a class will apply to any element in every component that contains it. This could be either an advantage or disadvantage, but I consider it a disadvantage because it forces a lot of different class names.
Vue.js [22] is an open-source progressive JavaScript framework for building user interfaces. It was created by Google employee Evan You and was first released in February 2014. Vue is focused on the ViewModel layer of the MVVM (Model-View-ViewModel) pattern. The View and the Model are connected through two-way data bindings [23]. Similar to React, Vue provides reactivity through a virtual DOM. All elements in Vue are implemented as virtual DOM nodes, named VNodes [16].
Vue is also a component-based framework. Components are created using an HTML-like file format called Single-File Component (*.vue files). It combines HTML elements and JavaScript functionalities and styles in a single file. Figure 9 shows an example of a Single-File Component [23].

Vue.js also relies on directives, similar to Angular. Directives are prefixed with „v-". For example, „v-if" is used for conditional rendering, similar to the Angular „*ngIf". An example of using the „v-if" directive can be seen in Figure 8.
The main advantages of Vue.js are: [17]–[21]
Easy to learn
Single-File Components allow for all the data of a component to be stored in a file
Lightweight, around 20KB
Flexible, it can be used for small and big projects
Detailed documentation
Virtual DOM
The drawbacks of Vue are:
Like React, it needs companion libraries

Small community, Vue is still new to the market

Fig. 8: Example of “v-if” directive in Vue.js – the h1 block will only be rendered if the directive’s expression returns a truthy value[47].

Risk over flexibility: Vue might have issues when integrating into big projects

Angular is an open-source TypeScript-based framework. This framework is not to be confused with AngularJS, which is the older, JavaScript-based version, which is no longer under active development. The framework that is discussed in this thesis refers to all versions starting from version 2, which is known as Angular 2+. Angular 2+ was released in 2016 as a complete TypeScript rewrite of AngularJS [16]. Just like the other 2 frameworks, Angular is component-based. However, one different aspect in Angular is the separation of template (HTML), styling (CSS) and component logic(TypeScript) in different files. [german pdf] Components can be created manually, or more commonly by using Angular’s command-line interface ng. Creating a component through the command line generates a folder that contains a template for each of the three files mentioned earlier with the addition of a testing file

Fig. 9: An example of a Single-file Component in Vue.js

(*.spec.ts) [16]. A minimal Angular component can be seen in Figure 10. The component is written in one file, it does not contain any styling or test files.

Angular does not use a Virtual DOM like React and Vue. It interacts with the real DOM and periodically runs a „change detection” mechanism to check whether the application state has changed and if any DOM needs to be updated [24].

Similar to Vue.js, Angular uses directives. An example is NgIf, a built-in structural directive responsible for HTML layout. In the example from Figure 11, the component „app-item-detail” will be shown if the value of „isActrive” is truthy. Directives are normally written between square parentheses, but this is a short-hand notation that Angular interprets and converts into a longer form[25][26].

One notable feature of Angular that other frameworks do not benefit from is the Dependency Injection design pattern. Services can be created as separate Injectable objects and „injected” into other component classes that request them. This design pattern increases flexibility and modularity in applications. There can be a service within another service.

The main advantages of Angular are: [17]–[21]

Useful for large and complex applications

It uses Typescript which makes it easier for developers who are used to strongly typed languages

Unlike Vue, Angular has reliable scalability

Dependency injection and separate files for a component which make Angular a very modular framework

Two-way data binding minimizes possible errors

Detailed documentation

Change detection mechanism that

Big community

The disadvantages are:

Quite difficult to learn

Requires learning Typescript

Fig. 11: Angular NgIf directive example [26]

Before starting the project, I had prior experience with React. However, I wanted to take this opportunity to learn Angular as well. Besides my interest in this framework, the reason for choosing to use Angular was because of its great modularity and detailed documentation in comparison to React and Vue. In addition, I like the use of directives and the ability to create custom directives.

4.2 Back-end technologies

Express.js is one of the most popular back-end frameworks for web development. Alongside Express.js, other two popular frameworks are ASP.NET and Spring Boot [27]. This section will cover these frameworks and highligh the features that made Express.js a suitable choice for this project.

ASP.NET is a framework for building web applications and services with .NET and C#. It was created by Microsoft and released in 2016. To include libraries in ASP.NET projects, Nuget packages can be downloaded.

Advantages of using ASP.NET include [28], [29]:

Reduced coding time

Security using built-in Windows verification

Rich toobox

Supports multi-threading

However, ASP.NET has the following limitations [30]:

Poor documentation

Costly compared to other options, licenses like Visual Studio are needed

Express.js is a framework for web application built on top of Node.js. Node.js is a JavaScript runtime environment used for building server-side event-driven apps and it is often confused with a framework or programming language. Even if Node.js can be used for web application development on its own, it is quite difficult and Express.js offers a robust set of features that make web development easier. To use libraries within Node.js, npm(Node Package Manager), the package manager for Node.js is used.

Pros of using Node.js with Express.js [31]:

Thousands of npm packages that can be downloaded and used

It uses middleware functions which has access to the request and response objects of an HTTP requests

Middleware for error handling can be used

Ability to create REST APIs (representational state transfer application programming interface)

Easy to learn

Cons of the framework [32]:

Poor performance for heavy computational tasks

The Asynchronous Programming Model makes code difficult to maintain and it means it doesn't support multi-threading

Java Spring Boot is an extension of the Java Spring Framework which makes development of web applications and microservices with Spring Framework faster and easier [33].

Java Spring Boot offers these advantages [34]:

-The Java community is enormous

-Java is statically typed making it easier to understand more complex code
-Supports multi-threading
-Efficient garbage-collection
This framework also comes with the following disadvantages:
-It uses a lot of memory
-It creates unused dependencies that cause large deployment files

4.3 Database management systems

According to the Stack Overflow Survey in 2021 [27], MongoDB is the fourth most popular database. This section will cover MongoDB and the top two most popular databases, MySQL and PostgreSQL.

MySQL is the most popular relational database management system and was first released in 1995. As a relational database, the data is organized in tables that can be related depending on how the user structures the database [35]. MySQL offers support on all major operating systems, Linux, macOS, and Windows, and supports many programming languages such as C, C++, C#, Python, and JavaScript

The advantages of MySQL are: [36], [37]

It's lightweight
Good performance according to benchmarking tests.
Large community
Easy to use for developers with SQL knowledge
Consistent data through enforcement of the relationship rules

The disadvantages of this database are:
Not suitable for more complex business logic
Not efficient with large databases
Not as flexible as a NoSQL database because of the Entity-Relationship concepts

PostgreSQL is an object-relational database management system that was first released in 1996. Being an object-relational database management system, it is similar to a relational database with the exception that it has an object-oriented database model and it supports objects, classes, and inheritance [38]. Like MySQL, PostgreSQL offers support on Linux, macOS, and Windows and supports many programming languages.

The advantages of PostgreSQL are: [36], [39]
It supports a wide range of SQL syntaxes
Powerful features like Multi-version concurrency control (MVCC) which allows a large number of concurrent users on one system
Supports advanced data types such as arrays and user-defined data types
Suitable for complex queries

The drawbacks of this database are:
Lack of data compression which can affect performance
Performance issues with large databases
High learning curve
Community support is not as available as for MySQL

Unlike the previous two databases mentioned above, MongoDB is a NoSQL database that stores data as JSON-like documents [40] and it was released in 2009. As a NoSQL database, the data doesn't need a predefined schema like SQL databases which makes it more flexible. It also supports the 3 main platforms and many programming languages.

Advantages of MongoDB are: [41]

Flexibility: it is easy to change fields compared to SQL databases
Faster than most relational databases
Simple query syntax
Scalability: MongoDB uses sharding for dividing data from a large set and distributing it to multiple servers
The limitations of MongoDB are:
It doesn't support JOIN operations as relational databases do
With wrong indexing, it can perform slow
Limited data size and nesting
It uses more memory because the lack of JOIN functionality produces duplicates
The main reason that attracted me to use MongoDB was the number of resources regarding web applications built with MongoDB which made learning fast and easy. Because this application executes many CRUD operations at a time, I also took into account the faster CRUD operations compared to SQL databases [42].

5 SUGGESTED SOLUTION

5.1 The MEAN Stack architecture

Before describing the architecture of the application itself, it is important to first discuss the architecture of the stack. As mentioned in the previous chapter, Angular is the client-side framework, Node.js is a JavaScript Runtime Environment that runs the server-side application via Express.js which is the web-application framework built over Node.js and MongoDB is the document database. The interaction between these technologies can be observed in Figure 12.
Initially, when the client makes a request, it is picked up by Angular which parses this request and sends it to Node.js. As a result, an HTTP request is made to an endpoint on the server. Next, the server makes a request through the Mongoose ODM (Object Data Modelling) library [43] to interact with the Database Server that has MongoDB. MongoDB will process the request and if it is accepted, it will relay the data to Express.js. Express will relay the response to Node which sends it to Angular. [44], [45]
As it can be seen from the diagram, the client and the server run locally on the machine, which is the reason why the communication between these two is made with the TCP/IP (Transmission Constrol Protocol / Internet Protocol) protocol. Instead of running the database locally, I chose to use the cloud version offered by the MongoDB Atlas service [46] and connect to the database using the Uniform Resource Identifier (URI) to the cluster, which is why the communication is made through the Secure Hyper Text Transfer Protocol (HTTPS).

Fig. 12: Diagram of the full stack application

5.2 Application architecture

The architecture is made up of three main components: the front-end component, the back-end component, and the database component. The front-end and back-end applications run on different processes, and when the back-end runs, the server automatically connects to the database.

The front-end application is responsible for providing a user-friendly interface for the user and the back-end application handles user authentication, HTTP requests, and communication with the database. All data is stored in the database. These components will be discussed in more detail in the following paragraphs.

5.2.1 Front-end application

Figure 13 presents the component diagram of the front-end application. The root component of the application is the „App-component" which is first rendered by the index.html file on the DOM. This component uses the „Navigation-component" which represents the navigation menu of the application. The „Navigation-component" is divided into 2 visual parts: the navigation menu and the content that is rendered on the page depending on the route. To render content that depends on the page route, the navigation component uses a router-outlet that manages which component to render depending on the route. The router-outlet paths are defined in the App-routing-module.

If the user is not logged in, the „Login-page-component" is displayed and the user cannot access other pages before logging in. After authenticating, the user can see the „Lore-collection-component" where they can see and manage the list of worlds that they have created. Adding, deleting, and modifying worlds call for the service represented as „Lore-collection-service" which makes HTTP requests to the server endpoints to update the data in the database. When accessing one of the worlds, the user is shown the „Main-text-page-component" which is the first editable document that is automatically created with the world. Every change in the editor of the document triggers an update request to the „Lore-collection-service" so that progress is automatically saved.

If a user wants to add a map, they can access the „Map-collection-component" through the menu. There, they can see all the maps that have been created and can be managed in the same way as the world list. When creating a new map, the user is shown the „Map-upload-component" where they can upload an image that serves as a map. When a map is created, the user is shown the „Map-page-component" which shows the interactive map and where the user can place marks. This component can be subsequently accessed from the „Map-collection-component". Adding, deleting, and modifying maps call for the service represented as „Maps-service".

A user can also manage multiple documents, which can be seen in the navigation menu through the „Document-expansion-panel-component". This component uses the „Document-tree-component" to display the list of documents in a hierarchy. To obtain the documents and their hierarchy, the „Document-expansion-panel-component" needs to call the „Documents-service" to retrieve the documents. When accessing one document, the user is shown the „Document-page-component" which uses the „Documents-service" to retrieve the data contained in the document.

All services make HTTP requests to the back-end application.

5.2.2 Back-end application

The architecture for the back-end application can be seen in Annex 1. The files are grouped in folders depending on the type of functionality they provide. The arrows are pointed from the file that requires a module to the file from which the module was exported.

The „config” folder contains a file that stores all environment variables and a function for establishing the connection to the database. The „middleware” folder contains custom middleware functions that are used by other files in the backend. These functions are:
-advancedResults: This is a function that allows advanced querying features like filtering, selecting certain fields, or pagination. It also allows populating fields with data, which means that fields that normally contain just an id referencing another Mongodb document from the database will instead contain all of the data of that document.
-async: The application uses many async middleware functions and each of them needs to use try/catch blocks. This repetition of try/catch blocks would break the DRY (Don't Repeat Yourself) principle, therefore a handler function is needed.
-error: a function that handles different types of errors such as duplicate keys or a request for a resource that doesn't exist
-auth: implements middleware for protecting routes. It creates an error response if there isn't a token provided in the header of the request or if the token doesn't have the correct format
The „models” folder contains data models which define the schema of each collection from our database. The „utils” folder contains the „errorResponse.js” file which defines a custom class that extends the Error class, the „sendEmail.js” utility which sends an email to a certain recipient with a custom message and the „sendTokenResponse.js” which creates a JSON token and saves it in the browser cookies.
The „controllers” folder contains controller methods which are associated with certain routes defined in the „routes” folder. One example controller method is the getDocuments method which makes a GET request to receive all Documents from the database.
The server is outside the folders mentioned above. It requires all the routes defined in the „routes” folder in order to mount them. To make environment variables accessible everywhere in the application, the environment variables in the „config.env” file are required. To connect to the database, the server uses the „connectDB” method inside the „db.js” file and to run the error handler it requires the error handling middleware.

5.2.3 Database

MongoDB uses the document model instead of the relational model. While MongoDB doesn't impose schemas, they can be created with the help of the Mongoose library to define the shape of documents from a collection. In the diagram from Figure 14, User, Lore, Document, and Map represent the collections, similar to tables in relational databases. A relation between collections is created when a field from one collection has an object field that contains the type „mongoose.Schema.ObjectId” and the reference to a collection. In this diagram for example, a lore can be associated with the user that created it by completing the user's id in the „owner” field and it will be a reference to the User collection. Similarly, the Document and Map collections should reference the Lore collection because it should be known what maps and documents belong to a certain lore. The id must respect the format of a „mongoose.Schema.ObjectId” because when a document is added to a collection, this kind of id is generated automatically.

6 IMPLEMENTATION DETAILS

This chapter will discuss the main features of the application in more detail focusing on how they were implemented. The first important problem that had to be solved was the creation of a secure account

system and the protection of important routes that require a user to be authenticated.
When making the User model for the database, simply storing the password is not enough because whoever might access the data in the database would have access to it. To prevent this, the bcryptjs library was

Fig.14: Database diagram

used to hash the password before saving it in the database. In Figure 15, after checking that the password field was modified, a „salt" value is generated and used to hash the password. The function is asynchronous because generating „salt" and hashing the password return a Promise which needs to be awaited. Bcrypt can also compare the crypted password with the plaintext and return wheter they match or not. Therefore, the password remains crypted and cannot be intercepted by any attackers.
To
check that a user is authenticated, a Bearer token is needed. Whenever a user registers, logs in, or resets their password, a bearer token is created and saved in the browser cookie, signaling that the user is authenticated. The generation of the Bearer token itself is implemented in the User model just like the method above. Storing the token in the browser cookies, setting an expiration limit, and sending back an HTTP response with an OK status code is handled by a utility file „sendTokenResponse.js". The code snippet in Figure 16 will show the token generation method.
In this method, the „jsonwebtoken" library (aliased as jwt)
generates a JSON web token using the id of the User as the payload, a secret value defined in the config.env file, and an option defined by the expiration time for a token which is also defined in the config.env file.
To
protect important routes, a middleware function was implemented. It checks if the header of a request contains a valid bearer token. If there is one, it will decode the token using the secret stored in the environment

Fig. 15: Password encryption
Fig. 16: Token generation

variables and find the user. The authenticated user's id can be easily extracted from the decoded token and the user is searched in the database using the id. If the user is not found, an error message is sent back with an HTTP error code of 400. To use the middleware
It is important that a user is able to securely change the password if they have forgotten it. This functionality is split in two middleware functions, one for generating a reset token and sending a reset email to the user's address and another function which sets the new password making sure the token is valid. These two functions are shown in Figure x and Figure y respectively. The first function takes the user email provided in the request body and checks if there is a user registered with that email in the database returning an error if the email is not found. If the user is found, a special token and an expiration date are generated. The function which generates the token is implemented in the User model similarly to the getSignedJwtToken and is shown in Figure z. The generated token and expiration date are saved in the databse. Next, an email is sent to the user's email address with the URL which contains the reset token. After

the user has entered the new password and confirms it, the second function gets the token from the URL. The token is then encrypted because the reset token in the database is also encrypted and the function looks for a user who has that token and makes sure the token didn't expire by also checking the resetPasswordExpire field. If no such user is found, an error response is returned, else the new password is set the the token and its expiration are cleared from the database. Finally, a token response is sent and the user is authenticated in the system.
The „sendEmail" function is a utility middleware. The Nodemailer module was used to send emails to the user. In the development stage, an email sandbox service was used to catch the emails instead of sending them to the real addresses.
There are other security vulnerabilities besides stolen passwords. The biggest vulnerability is represented by MongoDB's vulnerability to NoSQL Injections. A NoSQL Injection is when an attacker manipulates MongoDB queries by sending unsanitized JSON objects in the request body. For example, given the following MongoDB query:
db.users.find({ username: req.body.username,
password: req.body.password });
An attacker can send the following request body:
{
'username': {'$gt': ''},

'password': {'$gt': ''}
}
The „$gt" field is the „greater than" operator and MongoDB will compare the empty string with the username and password from the database, returning a true statement all the time. Therefore, the attacker will bypass authentication.
To prevent such attacks, the module „express-mongo-sanitize" was integrated in the API. The module offers middleware that searches for „$" operators in object keys and either removes them or replaces them with other characters.
Another vulnerability is cross-site-scripting (XSS). XSS attacks occur when an attacker sends malicious scripts which can be executed by the browser, for example through an input containing code wrapped in <script> tags. To prevent this attack, another module, „xss-clean", is enabled on the server.
Additional security can be achieved by setting various HTTP headers. All that is needed is the „helmet" middleware to be enabled on the server, just like the other security middleware mentioned earlier.

With the server implemented and security insured, the front-end application could be implemented. An important problem to solve was to make sure that an unauthenticated user cannot acces the pages unless they log in. For this, route guards, a feature provided by the @angular/router are needed. Figure 17 shows the router guard implementation. A guard is a service that can implement some interfaces provided by Angular which force the service class to add certain methods that the @angular/router can execute before it loads a route to check whether to continue or do something else. The interface that is needed for this project is the CanActivate interface which is imported from @angular/router. The class has to implement the canActivate method

which receives two arguments: the route that the user is trying to lead and the state which respresent a snapshot of the entire router state. The method returns either a true value and the user can access the route or it returns false the the user is denied access and redirected to the login page. To check if the user is logged in and can access the route, a separate service that deals with user authentication needs to be injected. Specifically, the getIsAuth method which returns whether a user is logged in or not. More about the authentication service will be discussed in the following paragraph.

Fig. 17: Router guard service

The authentication service communicates with the back-end application and executes all authentication operations such as logging in, logging out and getting the currently logged in user. Requests to the backend are sent using the HttpClient built-in feature. When logging in, the service takes the token provided by the back-end and stores it in the browser cookies and when logging out, the cookies are cleared and the user is sent to the login page. The front-end uses the token and sends it back to the back-end to receive information about the currently logged in user.
When a user is logs in, they are taken to the page that displays the lores that they have created. The lores are displayed as a list of cards. The HTML template of the component responsible for Lore management can be seen in Figure 18. Here, an *ngFor directive is used to repeat the mat-card component for each item in the loreData list. The list contains the lores that are retrived from the database. Each of these cards contain 2 buttons, one for deleting the lore from the list and another one for accessing the contents of the lore.

Fig. 18: HTML template for listing lores created by the user

To fetch the data from the database and also add, delete and update lores, a custom service named “LoreCollectionService” was created that communicates with the backend by making requests to the API. The service is used by injecting it in the component that needs it, in this case the “LoreCollection” component. The component needs the list of lores as soon as it’s loaded, therefore the lifecycle hook “ngOnInit” executes the “getLores” method which calls the authentication service’s “getLores” method which fetched the data. The method can be seen in Figure 19.
Figure 20 shows the “getLores” method inside the “LoreCollectionService”. Fetching the lores for a certain user requires the authentication service in order to get the id of the user that is currently logged in. Then the method sends a GET request to the API route “api/v1/lore-collection” with the query parameter “?owner=${res._id}” which selects only the lores whose owner is the currently logged in user.
When a user adds a new lore, a modal dialog prompt is opened asking for the title of the new lore. The model dialog component can be seen in Figure 21. There are multiple modal dialogs throughout the application, therefore a separate component is created which can be reused. To render the modal dialog, the openDialog method, which is shown in Figure 22, is triggered when the user clicks the „+” button. This method first opens the dialog by calling the open function which takes the

component that needs to appear and a list of options which include the width of the window and the text that should be displayed in different fields. A reference to that dialog is stored in dialogRef. When the dialog is closed, the value of the input field is returned and can be used in a subscribe method. After getting the input data, the LoreCollectionService will add a new entry with that data.

Fig. 19: Component method for fetching data
Fig. 20: Method for fetching data in the “LoreCollectionService”

A user can begin writing for a lore that they have created by clicking the „Write” button. They are redirected to the main page of the lore. By default, each lore comes with a main document in which the user can start writing. In order to write and format data in a document, a rich text editor was needed. The Quill Rich Text Editor is a component that offers all the functionalities of a rich text editor and it can be downloaded using the npm command. This component can be used by importing the Quill library in the component that uses it, in this case the MainTextPageComponent. The rich text editor component that is inserted in the MainTextPageComponent’s template contains multiple properties:
-#editor associates the id „editor” to the component
-[modules] is a component directive which specifies the list of tools that the editor should provide. The list was defined in the MainTextPageComponent and property binding is used to set this list to the modules directive
-(onContentChanged) is an event triggered by changing the text in the editor. This function is used to update the contents of a lore in the database
-Format specifies what kind of format should the editor’s content should have
-[(ngModel)] sets the content of the editor and is bound to the content variable in the MainTextPageComponent. This content is fetched from the database every time the MainTextPageComponent is rendered
-(contextmenu) is an event which triggers when the user right-clicks inside the editor. This event records the cursor position and opens a menu at that position. This menu offers the possibility of linking a document to the text selected in the editor.

Fig. 21: The modal dialog component for adding a new lore

A user can add new documents by clicking the „Add a new document” button in the Documents section of the navigation menu. When clicked, the button opens another dialog asking for the name of the new document and for the folder in which the document should be placed. The folder is selected by selecting a value from the list of folders that already exist. A document has the same structure as the main text page

and has the same functionalities.

Fig. 22: Method that handles opening and closing of the modal dialog
Fig. 23: The rich text editor component used in MainTextPageComponent template

The documents that are created are displayed as a tree. Figure 24 shows the document tree in the application.

Fig. 24: The document tree in the application menu

7 EVUALUATION

The back-end application was tested throughout its development using Postman. Postman is platform for testing APIs by serving as a client that makes requests to endpoints. All routes function properly and have the expected behaviour. Figure 25 shows an example of a GET request which returns the user that is currently logged in. Another example of an unauthorized DELETE request is shown in Figure 26.

Fig. 26: Postman DELETE request without Bearer token authorization
Fig. 25: Postman GET request with the response in the bottom

[Deploying the application with heroku]
[User's feedback from forms]

8 CONCLUSIONS AND SUBSEQUENT DEVELOPMENTS

This project developed a solution for writing organizing a fantasy world in an interactive way using modern and popular technologies. Users can now create multiple worlds and within each of these worlds they can create documents structured in folders, interactive maps, historical timelines and character relationship diagrams. All API routes from the application server were tested using Postman, a platform for testing APIs. The application was also deployed and made available using the Heroku service and users who tested the application expressed their feedback in a form which they had been offered.
The development of this application greatly improved my knowledge of web development in general and of the technologies that I have used. With this knowledge come more possibilities of development. A few features that could further be added to this application are:
-The possibility of adding users to a project for collaboration.
-Sharing documents with other users for read-only purposes.
-The possibility of automatically setting a hyperlink to a document to all instances of a word.
-Having multiple types of waypoints on the interactive map and being able to filter them.

-Diplomacy webs which display the relationships between nations.
To make the application successful, it should be tested for a higher number of concurrent users and make sure that it can be easily used by millions of users. In addition, Search Engine Optimization (SEO) is necessary for making this web page more findable by potential users.

9 BIBLIOGRAPHY

[1] J. Clute and J. Grant, The encyclopedia of fantasy. Macmillan, 1999.
[2] L. Carter, Kingdoms of Sorcery. Doubleday, 1976.
[3] S. J. Press, St. James Guide to Fantasy Writers. Saint James Press, 1996.
[4] G. Gygax and D. Arneson, dungeons & dragons, vol. 19. Tactical Studies Rules Lake Geneva, WI, 1974.
[5] M. J. P. Wolf, Building imaginary worlds: The theory and history of subcreation. Routledge, 2014.
[6] "World Anvil Worldbuilding tools & RPG Campaign Manager | World Anvil." https://www.worldanvil.com/ (accessed Jun. 19, 2022).
[7] "LegendKeeper - Worldbuilding app with maps, wiki, and whiteboards." https://www.legendkeeper.com/ (accessed Jun. 23, 2022).
[8] "The smart notebook for worldbuilders - Notebook.ai." https://www.notebook.ai/ (accessed Jun. 19, 2022).
[9] "@angular/cli vs ember-cli vs polymer-cli vs react vs vue | npm trends." https://www.npmtrends.com/react-vs-vue-vs-@angular/cli-vs-ember-cli-vs-polymer-cli (accessed Jun. 18, 2022).
[10] "React – A JavaScript library for building user interfaces." https://reactjs.org/ (accessed Jun. 18, 2022).
[11] "Components and Props – React." https://reactjs.org/docs/components-and-props.html (accessed Jun. 18, 2022).
[12] "Hooks at a Glance – React." https://reactjs.org/docs/hooks-overview.html (accessed Jun. 18, 2022).
[13] "Tutorial: Intro to React – React." https://reactjs.org/tutorial/tutorial.html#function-components (accessed Jun. 18, 2022).
[14] "Understanding Functional Components vs. Class Components in React." https://www.twilio.com/blog/react-choose-functional-components (accessed Jun. 18, 2022).
[15] "React: The Virtual DOM | Codecademy." https://www.codecademy.com/article/react-virtual-dom (accessed Jun. 18, 2022).

[16] M. Levlin, "DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte," 2020.
[17] "React vs Angular vs Vue.js — What to choose in 2021? (updated in 2021) | by TechMagic | TechMagic | Medium." https://medium.com/techmagic/reactjs-vs-angular5-vs-vue-js-what-to-choose-in-2018-b91e028fa91d (accessed Jun. 19, 2022).
[18] "Angular vs. React vs. Vue.js: Comparing performance | LogRocket Blog." https://blog.logrocket.com/angular-vs-react-vs-vue-js-comparing-performance/ (accessed Jun. 19, 2022).
[19] "Angular vs React vs Vue - My Thoughts." https://academind.com/tutorials/angular-vs-react-vs-vue-my-thoughts (accessed Jun. 19, 2022).
[20] "Angular vs React vs Vue: The Main Differences and Use Cases." https://incora.software/insights/react-vs-angular-vs-vue-the-main-differences-and-use-cases/55 (accessed Jun. 19, 2022).

[21] “Best Frontend Frameworks for Web Development in 2022.” https://www.simform.com/blog/best-frontend-frameworks/ (accessed Jun. 19, 2022).
[22] “Vue.js - The Progressive JavaScript Framework | Vue.js.” https://vuejs.org/ (accessed Jun. 18, 2022).
[23] “Getting Started - vue.js.” https://012.vuejs.org/guide/ (accessed Jun. 18, 2022).
[24] “Angular - Angular change detection and runtime optimization.” https://angular.io/guide/change-detection (accessed Jun. 18, 2022).
[25] “Angular - Structural directives.” https://angular.io/guide/structural-directives (accessed Jun. 18, 2022).
[26] “Angular - Built-in directives.” https://angular.io/guide/built-in-directives (accessed Jun. 18, 2022).
[27] “Stack Overflow Developer Survey 2021.” https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologies (accessed Jun. 21, 2022).
[28] “Difference Between Node.js and Asp.net - GeeksforGeeks.” https://www.geeksforgeeks.org/difference-between-node-js-and-asp-net/ (accessed Jun. 21, 2022).
[29] “ASP.NET – Pros and Cons You Ought to Know.” https://www.popwebdesign.net/popart_blog/en/2020/03/asp-net-pros-and-cons-you-ought-to-know/ (accessed Jun. 21, 2022).

[30] “What are the advantages & disadvantages of using ASP.NET? | Compspice.” https://www.compspice.com/what-are-the-advantages-disadvantages-of-using-asp-net/ (accessed Jun. 21, 2022).
[31] “Express.js Mobile App Development: Pros and Cons for Developers.” https://binariks.com/blog/express-js-mobile-app-development-pros-cons-developers/ (accessed Jun. 21, 2022).
[32] “Advantages & Disadvantages of Node.js : Why to Use Node.js?” https://www.simform.com/blog/nodejs-advantages-disadvantages/ (accessed Jun. 21, 2022).
[33] “What is Java Spring Boot? | IBM.” https://www.ibm.com/cloud/learn/java-spring-boot (accessed Jun. 21, 2022).
[34] “Node.js vs Java spring boot for microservice | Software Development Company in USA with Top Software Services |.” https://www.sayonetech.com/blog/nodejs-vs-java-spring-boot-microservice/ (accessed Jun. 21, 2022).
[35] “MySQL :: MySQL 8.0 Reference Manual :: 1.2.1 What is MySQL?” https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html (accessed Jun. 20, 2022).
[36] “MySQL vs PostgreSQL vs SQLite: A comparison of 3 popular RDBMS.” https://devathon.com/blog/mysql-vs-postgresql-vs-sqlite/ (accessed Jun. 20, 2022).
[37] “SQL vs NoSQL comparison: MySQL, PostgreSQL, MongoDB & Cassandra.” https://devathon.com/blog/sql-vs-nosql-mysql-vs-postgresql-vs-mongodb-vs-cassandra/ (accessed Jun. 20, 2022).
[38] “What is Object-Relational Database Management System (ORDBMS)? - Definition from Techopedia.” https://www.techopedia.com/definition/8715/object-relational-database-management-system-ordbms (accessed Jun. 20, 2022).
[39] “PostgreSQL vs MySQL: Difference You Need To Know - InterviewBit.” https://www.interviewbit.com/blog/postgresql-vs-mysql/ (accessed Jun. 20, 2022).
[40] “What Is MongoDB? | MongoDB.” https://www.mongodb.com/what-is-mongodb (accessed Jun. 20, 2022).
[41] “Understanding the Pros and Cons of MongoDB.” https://www.knowledgenile.com/blogs/pros-and-cons-of-mongodb/#Advan

tagesofMongoDB (accessed Jun. 20, 2022).

[42] R. Deari, X. Zenuni, J. Ajdari, F. Ismaili, and B. Raufi, “Analysis and comparison of document-based databases with sql relational databases: Mongodb vs mysql,” in Proceedings of the International Conference on Information Technologies, 2018, pp. 1–10.
[43] “Mongoose ODM v6.4.0.” https://mongoosejs.com/ (accessed Jun. 19, 2022).
[44] “Want to understand the MEAN Stack quickly? Here’s documentation with useful diagrams.” https://www.freecodecamp.org/news/cjn-understanding-mean-stack-through-diagrams/ (accessed Jun. 19, 2022).
[45] “How to Build a Node.js Ecommerce App?” https://www.simform.com/blog/build-ecommerce-web-application-node-js/ (accessed Jun. 19, 2022).
[46] “What is MongoDB Atlas? — MongoDB Atlas.” https://www.mongodb.com/docs/atlas/ (accessed Jun. 19, 2022).
[47] “Conditional Rendering | Vue.js.” https://vuejs.org/guide/essentials/conditional.html (accessed Jun. 18, 2022).
T

10 APPENDIX

Appendix 1: Back-end architecture diagram