UNIVERSITY OF TARTU
Institute of Computer Science
Informatics Curriculum

Siim Langel

Web-based viewer for Fractal Enterprise models

Bachelor's thesis (9 EAP)

Supervisor(s): Dr. Ilia Bider

Tartu 2022

Web-based viewer for Fractal Enterprise Models

Abstract

This thesis aims to develop a better user experience for viewing diagrams produced by an existing enterprise modeling tool called the FEM toolkit. The FEM toolkit allows exporting models and model groups in a graphical format and in XML format in which the necessary data for rendering all model elements are defined. This solution aims to rebuild the diagrams in a native WEB environment employing the capabilities of https://reactjs.org/ – a high-level library used to build user interfaces for the WEB. The current method requires prior training for the users to understand and navigate models. This thesis describes how to make the process more user-friendly and accessible to a wider variety of end-users.

CERCS: **T120 Systems engineering, computer technology**

Keywords: UX, UI, WEB, React, FEM

Veebipõhine Fraktaal Ettevõtte Mudelite kuvaja

Lühikokkuvõte

Selle teesi eesmärk on arendada parem kasutajakogemus mudelite kuvamisel, mis on olemasoleva ettevõtte modelleerimise rakenduse, FEM *toolkit'i*, poolt loodud. Eelnimetatud rakendus võimaldab eksportida mudeleid ja mudeli gruppe graafilises –ja XML formaadis, mis sisaldab vajalikke andmeid mudelite elementide visualiseerimiseks. Selle lahenduse siht on diagrammid ümberehitada veebipõhises vormingus rakendades https://reactjs.org/ poolt pakutud võimekusi. React on kõrgtaseme teek veebipõhiste kasutajaliideste loomiseks. Olemasolev meetod nõuab mudelite vahel navigeerimiseks ja nende hoomamiseks eelnevat treeningut. See tees kirjeldab meetodit, kuidas muuta see protsess kasutajasõbralikumaks ja kättesaadavamaks suuremale hulgale kasutajatele.

CERCS: **T120 Süsteemitehnoloogia, arvutitehnoloogia**

Võtmesõnad: UX, UI, WEB, React, FEM

Table of contents

Web-based viewer for Fractal Enterprise Models **2**

Abstract 2

Veebipõhine Fraktaal Ettevõtte Mudelite kuvaja **2**

Lühikokkuvõte 2

1 Introduction **6**

1.1 Problem statement 7

2 Background **8**

2.1 Fractal Enterprise Models 8

2.2 FEM toolkit 11

2.3 XML 12

2.3.1 History of XML 13

2.3.3 How XML works 13

3. Requirements **14**

3.1 Functional requirements 14

3.2 Non-functional requirements 15

3.3 Requirements on technologies/tools used for building application 15

4. Technologies/tools chosen for the development **16**

4.1 React.js 16

4.2 Node.js 17

4.3 Express.js 18

4.4 Database 19

4.4.1 MySQL 19

4.4.2 Prisma.js 19

4.5 Authentication 20

4.5.1 Passport.js 20

5 Completed application description **21**

5.1 Client-side 21

5.1.1 Authentication 22
5.1.2 Login page 23
5.1.2 Register page 23
5.1.4 User dashboard 24
5.1.5 Uploading 25
5.1.6 Model viewer page 26
5.2 Server-side 28
5.2.1 Authentication 29
5.2.2 Routing 30
5.2.3 Services 30
5.2.4 Database design 31
5.3 User Interaction 32
5.4 Future development 34
6 Conclusion 35
References 36
Appendix 38
I. Source code and test installation 38
II. Licence 39

1 Introduction

Enterprise Modelling provides a way for humans to reason about the properties and workings of an enterprise by constructing abstracted representations. It can help provide insight into an enterprise's structure, functionality, behavior, management, operations, and maintenance. The need for such a discipline emerged in the 80s. System engineers needed a way to structure and analyze business processes to keep building more complex applications. Needs arose among manufacturing and industrial engineers who had to integrate manufacturing with computer-aided systems. Business analysts and organization experts must describe business entities and revise more efficient solutions. The type of enterprise does not limit the usage of Enterprise Modeling. It can be applied to profit or non-profit organizations, ranging from industrial firms to academic establishments. Thus, Enterprise Modeling has found widespread use in the industry [1].

1.1 Problem statement

Ilia Bider, Erik Perjons, and others have developed a toolkit for modeling enterprises by drawing Fractal Enterprise Models (FEM) [2]. This software was created using the ADOxx meta-modeling platform [3], which enables the creation of modeling toolkits and modeling languages such as FEM. Dimitris Karagiannis, Moonkun Lee, Knut Hinkelmann, Wilfrid Utz, and others emphasize the importance of the ADOxx platform for tools implementations in their book [4], which presents over 20 projects based on the ADOxx platform. Their previous volume [5] contains even more cases of using ADOxx for developing tools. All these tools share with the FEM toolkit a number of issues that are explained in the next paragraph.

Due to the FEM toolkit being built upon the ADOxx platform, users have to go through an excessive installation process to start using the application. This includes installing SQL server and several Microsoft packages. As a result, installation on operating systems other than Windows requires virtualization. The installation is a tedious process that stakeholders other than model developers are not willing or should not have to go through considering that their primary goal is viewing, and not developing models. The problem for viewers is further amplified

because the toolkit, which is aimed for the modelers, is challenging to use for inexperienced business people, often requiring users to attend workshops or receive coaching previously. Finally, no way to share models is provided by the toolkit. Users currently have to export models and distribute them through e-mail or other forms of exchanging files.

This Bachelor's thesis aims to solve these problems by building a WEB-based viewer for FEMs. This approach is platform agnostic as the WEB environment is not dependent on the operating system. The solution will work based on exports from the existing toolkit that can be uploaded and shared between users of the new application - FEM viewer. The FEM viewer will enable model developers to create models and share them with stakeholders without the latter having any installation or training overhead.

2 Background

2.1 Fractal Enterprise Models

Ilia Bider and others describe the need for Fractal Enterprise modes in paper [6]. Many enterprise- and organization-related projects try to identify the most vital processes, if finding all of them proves too tedious. However, this could provide too shallow an overview of the organization, as only highly visible processes will be listed. The significant proportion of less noticeable processes might become outside the scope of consideration. In addition, establishing just a list of existing processes might not be enough for practical purposes. Changing one process may also alter how other processes behave due to their dependencies. Thus, the relationships and interconnections between the processes should be taken into consideration as well.

As a solution to both finding less visible processes and establishing interconnections between found processes, the researchers mentioned above have proposed a so-called Fractal Enterprise Model (FEM). It has two main elements - business processes and assets, as well as depicts relations between them, see Fig. 1 for illustration. Relations are further divided into two categories. If a solid line points from an asset to a process, the process is said to be “using” the asset. A dashed arrow pointing from a process to an asset is used to show a process managing or changing an asset. These elements may have labels to clarify what they represent for processes and assets or show their types -- for relations. The number of ways an asset and a process can be related is not limited. An asset can be used in different ways if it has several labels on an arrow to a process. A process can also modify an asset that it manages in different ways. The concept of archetypes can be used to make the process of building models more structured. They provide a way to create templates by omitting labels from processes and assets creating a repeatable pattern to be used when building a model [6]. A model presented in Fig. 1 was built using the FEM toolkit, see the next subsection. It shows assets as rectangular shapes and processes as ellipsed shapes. It also visualizes the relationships between different elements with labeled arrows. Three elements in this specific model have thicker arrows hovering over them, signaling

that they represent existing somewhere else elements, the latter are associated with them via references. These elements are called ghosts (of the original shapes). The FEM toolkit allows the user to jump from a ghost to the original element, and also find all ghosts of an element that exist in the same or different models.

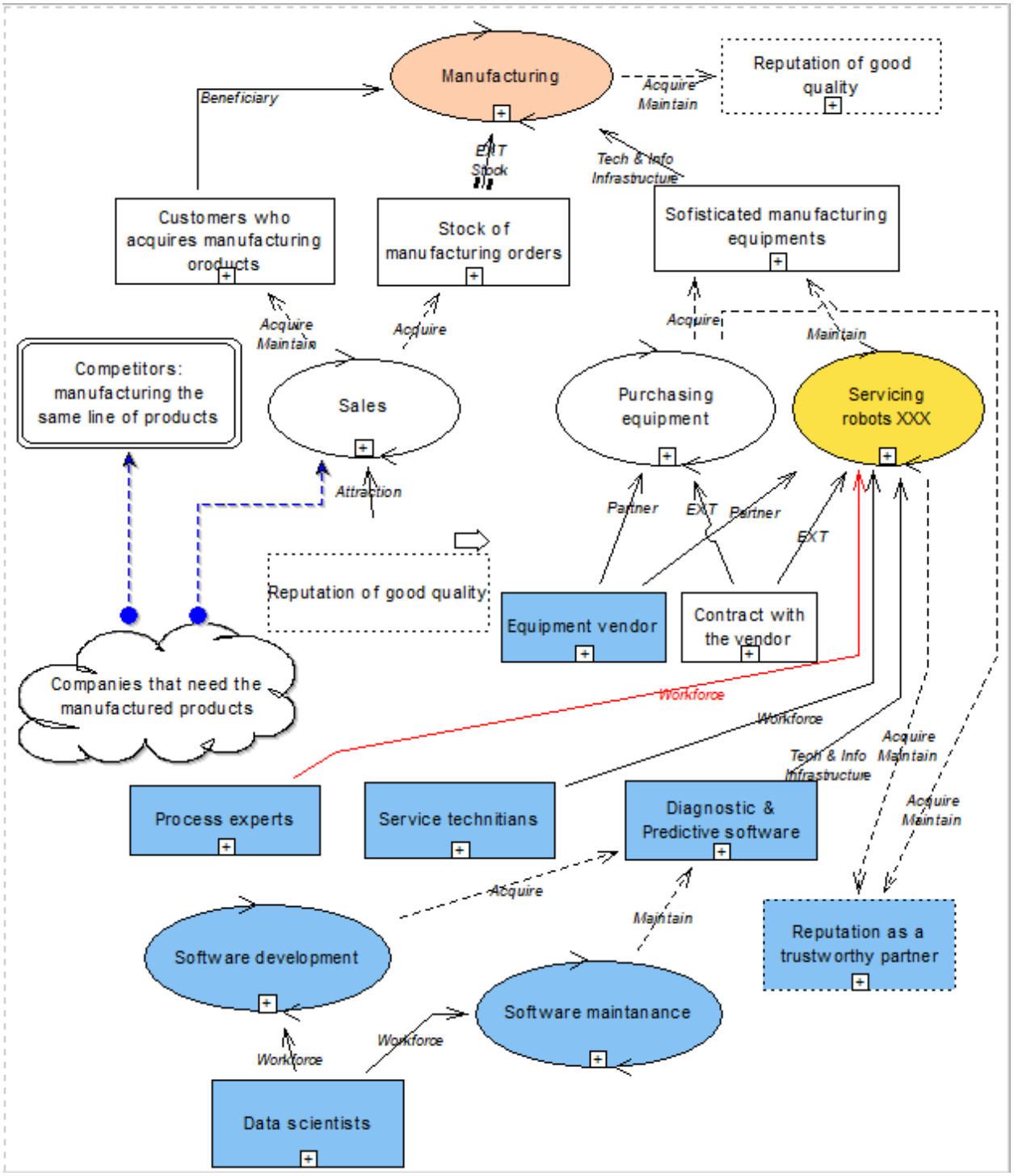

Figure 1. Fractal Enterprise Model Manufacturer.

2.2 FEM toolkit

The FEM toolkit [2] is a modeling toolkit developed using the ADOxx metamodelling platform [3], enabling the creation of Fractal Enterprise Models. FEM model creation is done by adding different business-related elements, e.g. assets and processes and visualizing relations between them by connecting them with labeled arrows. As this is something the most general-purpose diagram tools can do, the FEM toolkit contains some extra improvements.

- The toolkit makes sure that models built using it adhere to the FEM metamodel, meaning that only syntactically correct relations can be drawn between elements.
- Model developers can invoke an archetype function on the elements of the model (processes or assets). When using an archetype on an element, the modeler can choose which relations to add, which leads to new elements being added to the model and connected to the given element by these relations.
- Developers can create multiple copies of the same element in a model through the ghost feature. Ghost elements are visually differentiated from the original shapes and contain a reference to the original for navigation. Thanks to this feature, developers can place a copy of an element into the same or other models [7]. In case of the same model, it helps to avoid too many arrows incoming/outgoing from the same element. In case of different models, the feature allows to interconnect them.

In Fig. 2, the FEM toolkit with its main functionality is shown. On the left, it contains a tree with all models grouped into directories which enables navigation between models. In the middle, there is a bar with all the elements and connector types that can be used to build a model. The right side is devoted to viewing and interacting with a model. All the elements and relations are shown here; a click on any of them provides more detailed information and reference-based navigation. The navigator on the bottom left enables moving around the current model if it has been zoomed in. The header on the top provides access to various functionality of the toolkit. For example, it allows the creation of ghosts from model elements and exporting models into XML files.

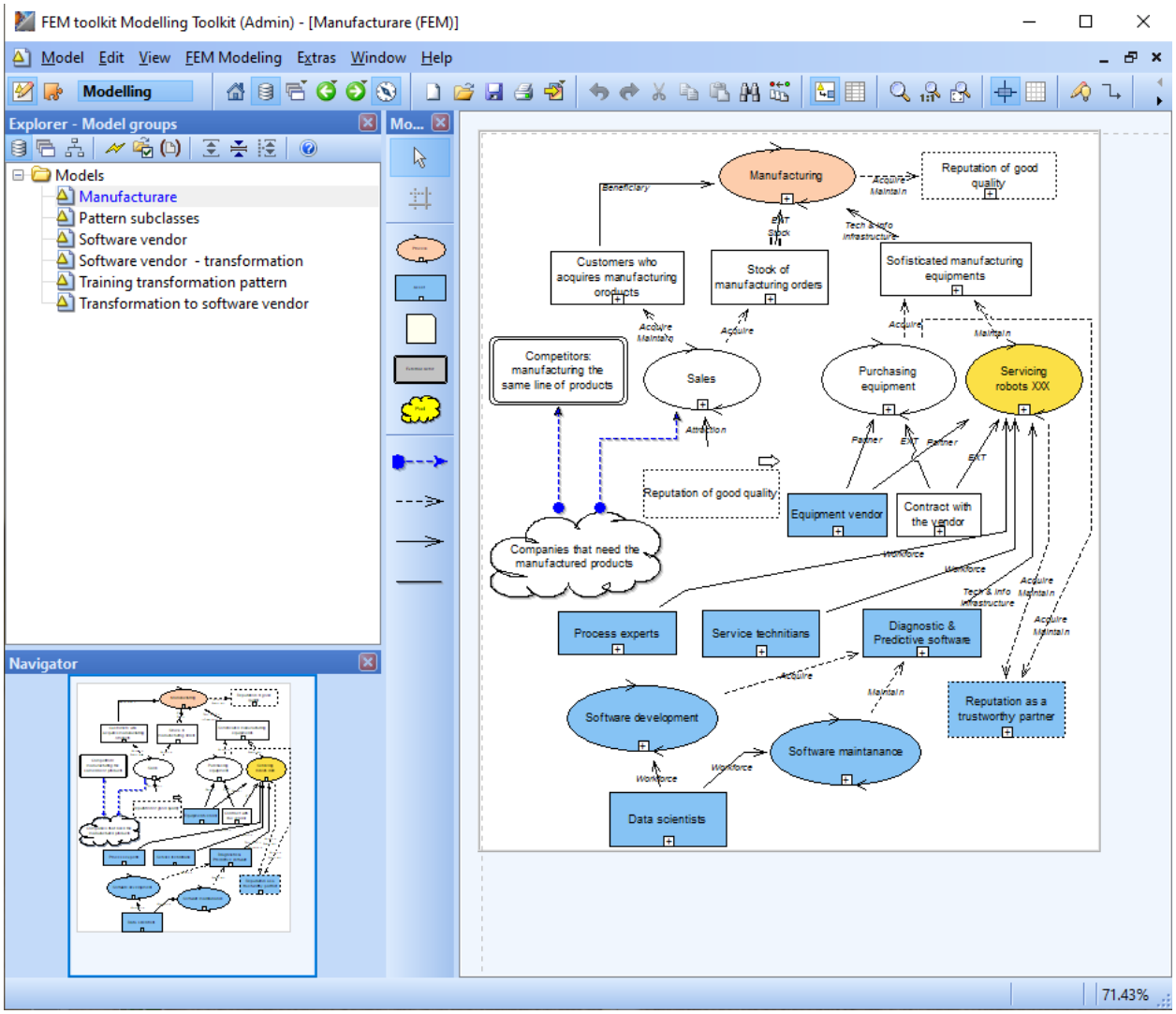

Figure 2. FEM toolkit.

2.3 XML

One way to export FEMs while preserving all the necessary data to recreate them is to use XML. This section will look into why XML has found such widespread use and how it works.

2.3.1 History of XML

"XML, the Extensible Markup Language is a W3C-endorsed standard for document markup" [8]. XML is derived from a metalanguage called SGML, the Standard Generalised Markup Language published in ISO 8879:1986 [9,10]. SGML attempted to solve much of the same problems as XML has now, and it even found success. However, it was much too complicated ever to be considered to be implemented fully. Its 150-page long technical specification serves as a testament to it. Jon Bosak, Tim Bray, C. M. Sperberg-McQueen, James Clark, and several colleagues decided they could change this. In 1998 they managed to publish XML 1.0, which appealed to developers at once. It encompassed various fields requiring data structuring, from agriculture to law [9][10].

2.3.3 How XML works

In XML files, data is separated into text strings surrounded by tags. The format is not aimed at specifics but instead provides a way to structure data for any field of study. Tags can be nested to form trees or created as siblings to make lists. These elements can have attributes attached to provide semantic meanings. The essence of XML is that it shouldn't be used as a presentation language but rather as a semantic structural language [8–10]. How it should be used is described in the XML 1.0 specification [10]. For a text document to be considered XML, it must be well-formed. To Abide to this standard, before any other element, all XML documents should contain a tag stating the version of XML being used. This is so that an XML processor would know the type of features it is parsing. Each element must have a start tag and an end tag containing at least the name. This name must be the same for both tags. This does not hold for self-closing tags as a single tag forms them. A valid name can contain any character except whitespace but not start with a number, punctuation, or the string "XML" in varying cases. Attributes can only be added to the starting tags of an element. They are denoted by key-value pairs separated by an equals sign, with the values being surrounded by single or double quotes. There is much more than this to XML. However, for the scope of this thesis, knowing this will be enough to parse FEM XML exports.

3. Requirements

This chapter outlines the requirements for the FEM viewer agreed upon by the author and their supervisor. The requirements were ascertained through numerous e-mail exchanges as well as Zoom calls. As the core functionality relied on exports from the FEM toolkit, which has no full documentation, most of the information needed for the development were drawn from the communication with the thesis supervisor who was the project leader of the development of the FEM toolkit.

3.1 Functional requirements

1. The system has to allow user creation.
2. Users should be authenticated through the use of cookies.
3. Users can upload models using XML and SVG exports.
4. Users can view models uploaded by them.
5. Users can share models they have uploaded with other users.
6. Users can view models shared with them.
7. There has to be a Login page.
8. The Login page should enable users to log in with their username and password.
9. There has to be a user-specific Dashboard
10. The Dashboard has to show a list of models uploaded by the user
11. The Dashboard has to display a list of users a model has been shared with
12. The Dashboard has to display a list of models shared with the user.
13. The Dashboard has to show which user shared a model with a user.
14. The Dashboard has to enable users to delete models they have uploaded.
15. The application has to have a Header.
16. The Header has to have navigation.
17. The Header has to enable logging out authenticated users.
18. The application has to have a Viewer page to display models.

19. The Viewer has to display a Model Tree on the right side.
20. The Model Tree has to show a tree of model names.
21. The Model Tree has to highlight the current model.
22. The Model Tree has to enable model switching.
23. The Viewer has to use the remaining space to display a model.
24. Model Elements have to be highlighted on click.
25. A Details Popup should show up when clicking a Model Element
26. The Details Popup has to be draggable.
27. The Details Popup has to show information regarding the current Model Element.
28. The Details Popup should show the current Model Element usages and references.
29. The Model has to be zoomable.

3.2 Non-functional requirements

1. Compatibility requirement. The application should be compatible with the most spread WEB browsers:
    a. The application has to support Google Chrome.
    b. The application has to support Mozilla Firefox.
    c. The application has to support Safari.
2. Security requirement. The application must use HTTPS protocol

3.3 Requirements on technologies/tools used for building application

By the technology/tools here, we understand any library, package or tool used in developing the system.

1. The technology/tool should have enough functionality to realise the above listed requirements.
2. The technology/tool should be high-level, to exclude where possible, low-code programming, and make the maintenance and further development easier.
3. The application should be possible to maintain and further develop even after the thesis project has been finished. Therefore, the technology/tool should have some spreading on

the market to ensure that it will exist and be supported for some time. Also, it should be easy to find developers having appropriate skills, or willing to acquire them.

4. Technologies/tools chosen for the development

This chapter lists technology/tools chosen for the development based on the requirements from the previous section.

4.1 React.js

React is a declarative Javascript library that helps developers build user interfaces [11]. Engineers created it at Facebook to solve problems that arose when developing user interfaces that got more complex at an alarming rate. When React was created in 2013, it changed how web applications were made [12]. The following points elaborate on why React.js is a good choice for building web applications.

- One of the reasons that React is so powerful and has found such widespread use is the use of virtual DOM. This concept enables React to make changes to the actual DOM only when needed, making applications run in a browser much more efficiently [12]. With this in place, developers can tell React what state they want their application to be in, and React updates the actual DOM accordingly by making updates only when needed [13].
- React encourages a component-based design where components can be built, encapsulated, isolated, and later composed into complex UIs. This provides reusability as components can be plugged in anywhere when needed, even between different projects. As components are split independently, they can be thought of and tested independently, enabling the creation of highly scalable applications. A component is just a JavaScript function that can be easily defined. With such minimal overhead, getting started with React is simple, provided the developer has prior HTML and JavaScript knowledge [14].
- The feature that sets React apart from several other UI libraries and frameworks is the use of the JSX transform layer. It changes the XML syntax in which a component is written to React's actual code for rendering [12]. Unlike traditional methods that separate logic and markup into different files, React components contain both. It is similar to templating languages but differs because it utilizes the full power of JavaScript [15].

Models adhering to FEM standards may have an arbitrary amount of elements that need to be rendered, requiring high performance. This becomes especially apparent when elements need to be interactive as well. FEM viewer has potential for further development, so making it scalable is necessary. The author chose to use React because it helps solve the above problems. Having prior experience with React helped with preplanning all of the requirements.

4.2 Node.js

Node.js is an asynchronous event-driven platform built on Chrome's V8 JavaScript runtime to design scalable network applications [16]-[17]. Node.js deviates from the more common concurrency model of using operating system threads by using a single-threaded event loop. This enables it to be non-blocking unless I/O operations need to be performed. In such cases, requests are taken from the queue and assigned threads from a limited internal thread pool to process the request. Once a blocking task is done, it is placed back into the queue without other tasks having been blocked [18].

Running JavaScript in the browser comes with the disadvantage of running it on different browsers that may use a different subset of JavaScript features. Node.js doesn't have this problem as it allows targeting specific Node.js and JavaScript versions that are known in advance. Furthermore, Node.js comes coupled with the latest version of Chrome's V8 engine. As a result, developers can use all the latest features of ECMAScript without any extra steps, including transpilation and polyfilling [19].

Although Node.js uses JavaScript, it does not run in the browser and thus access the operating system. Most importantly to this project, it has access to the filesystem and the ability to create HTTP(S) servers [19]. Thanks to this, FEM models can be uploaded and persisted on the server for processing and sent to the client for presentation.

4.3 Express.js

Express.js is a minimalist web framework built on top of Node.js to create flexible and fast APIs quickly. It has a vast amount of HTTP utilities that satisfy the needs of most server-side applications [20]. Express.js provides the developer with a minimal set of functionality to get started with web development. Even though this is true, thanks to the open-source nature of

Node.js, a myriad of open-source middleware have been created to handle more complex development problems. This project uses authentication, session, and file upload middleware more specifically. This flexibility provided to the user does come with its downsides. Choosing the suitable packages from so many options can be challenging. When more opinionated frameworks enforce a rigid structure to applications, Express.js leaves the choice up to the user. This choice means that planning the architecture of an application becomes another complex problem to solve [20,21]. For the FEM viewer, the author decided to use Express.js for the following reasons.

- The application's server-side has three main requirements: uploading files, user authentication, and persistence through a relational database. Express.js can provide all this without minimal overhead, instead of more opinionated frameworks that come with these and much more unnecessary features included.
- Express.js and React.js both use JavaScript. Having the same syntax on both the client and server-side makes for a smoother development process. This also helps separate the concern of structuring and typing the parsed FEM models. It is principal to keep the structure of the parsed models in sync between the client and server-side as the models are parsed on the server-side and then sent to the client.

4.4 Database

4.4.1 MySQL

MySQL is the most popular open-source SQL database management system that supports relational databases and is developed by Oracle Corporation. It separates data into tables that are stored physically as files. It enables developers to set up relationships and rules between data, which it enforces to keep data consistent [22].

- One of the main driving factors behind choosing MySQL is its being open source, meaning the author can download it and use it for free.

- Secondly, it can run on different operating systems such as Microsoft Windows and different Linux distributions. As this also holds for Node.js, development, and production are both operating system independent.
- Finally, MySQL integrates with the Object-relational mapping(ORM) library, described in the next paragraph.

4.4.2 Prisma.js

Systems based on an object-oriented domain model often need a way to persist data of the entities present in the FEM viewer. Object-relational mapping provides a way to abstract the difference between the object-oriented paradigm and data access [23]. Using an ORM, developers can interact directly with objects rather than writing SQL to manipulate data.

Prisma.js is a Node.js ORM library that integrates with open source databases, most notably for this project, with MySQL. Prisma.js has many features that help developers make fewer errors when working with databases [24].

- It provides a schema file that can be used to specify the shape of the database in a more human-readable and declarative way than SQL can [25].
- It can automatically generate SQL for database migrations to keep the database and the schema in sync [26].

The author chose to use this library because it abstracts data access, which makes development and maintenance more manageable and lets the author focus on business logic.

4.5 Authentication

The FEM viewer requires some data to be specific to a select number of users. This means that users have to be authenticated while maintaining a pleasant user experience as per the problem statement. The author chose to use session-based authentication for this. Session-based authentication is a stateful authentication technique that creates a session on the server-side to hold necessary information to keep a connected user authenticated. The browser is also sent

information about the session to be stored in the form of a cookie. This way, the server can decide whether a user is authenticated or not [27].

4.5.1 Passport.js

Passport.js is one of many authentication middlewares for Node.js. It can be used with any Express.js based application, integrating well with the development stack of this thesis's application. It comes with built-in support for session-based authentication [28]. These points fill all the requirements for an authentication library that the FEM viewer needs. It comes with the downside of having very minimal documentation. However, this can be overlooked due to the popularity of the library. This widespread use means that extensive blog posts and articles covering the library's features cover this issue. Furthermore, high demand for the library resulted in regular maintenance and continued support.

5 Completed application description

The FEM viewer enables users to upload, share and view Fractal Enterprise Models. The present chapter describes its architecture and functionality. It also gives insight into how the functional requirements described in chapter 3 have been implemented. Bear in mind that this chapter describes the application's state at the time of writing, as the application is subject to further development.

5.1 Client-side

The author took a minimalist approach while designing the application's user interface. Design decisions were made so that users would only be shown the minimal amount of information needed. Users have to intuitively be able to navigate the application and make use of its features. Fig. 3 illustrates the architecture of the client-side solution as a diagram. It shows a tree structure of how the React.js components hierarchy is built.

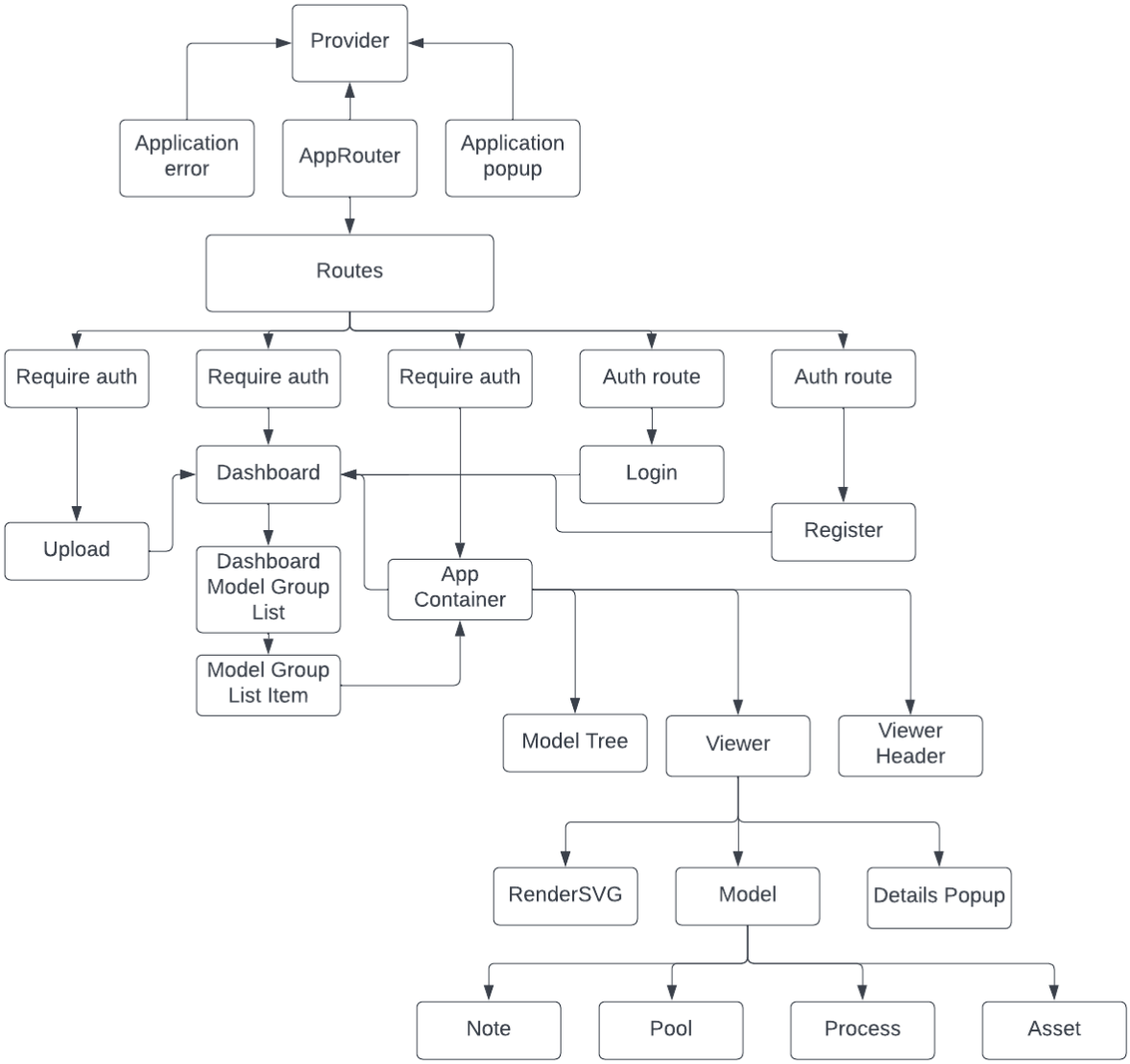

Figure 3. Client-side solution diagram.

5.1.1 Authentication

The client-side routing was set up so that pages are set to be accessed by either authenticated or unauthenticated users. Unauthenticated users trying to access protected routes would be denied access and redirected to pages that enable them to authenticate. In the same fashion,

authenticated users trying to access routes that allow authentication would first have to log out. Otherwise, they are redirected to keep them from reauthenticating.

5.1.2 Login page

Users first visiting the FEM viewer or unauthenticated users will be redirected to the Login page. The page contains a header letting users know they have reached the FEM viewer application. It includes a single form prompting the user for their login details. As the requirements did not describe any need for account verification, an alphanumeric username and password sufficed. Users are also presented with a link to the Register page if they have not yet obtained an account.

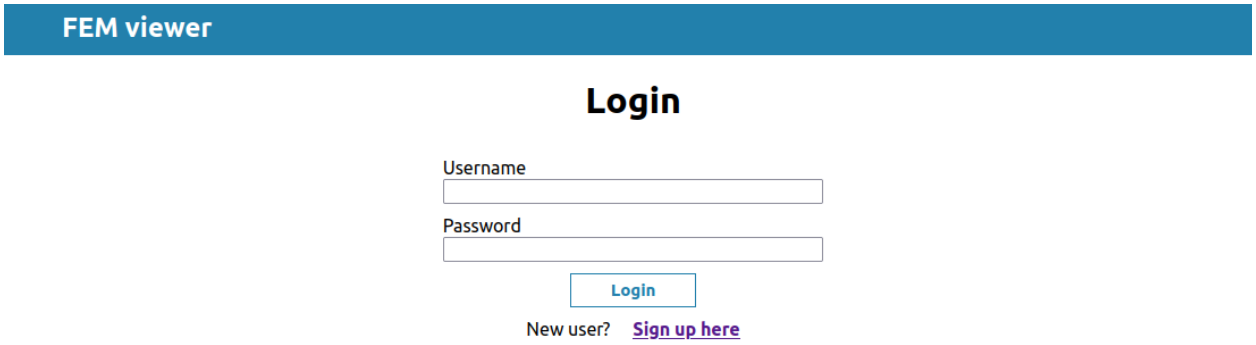

Figure 4. Login page.

5.1.2 Register page

The Register page enables users to create accounts with a username and a password. These fields have constraints placed on them described in chapter 5.2.1. The design is identical to the Login page, with a few differences. It prompts users to enter their details to create a new account rather than log in with an existing one. It also requires entering the chosen password twice to reduce typos. It has a navigation link back to the login page as well. Upon successful form submission, the user is automatically logged in and has a new account created. The login action is the same as in the login page, redirecting the user to their specific dashboard.

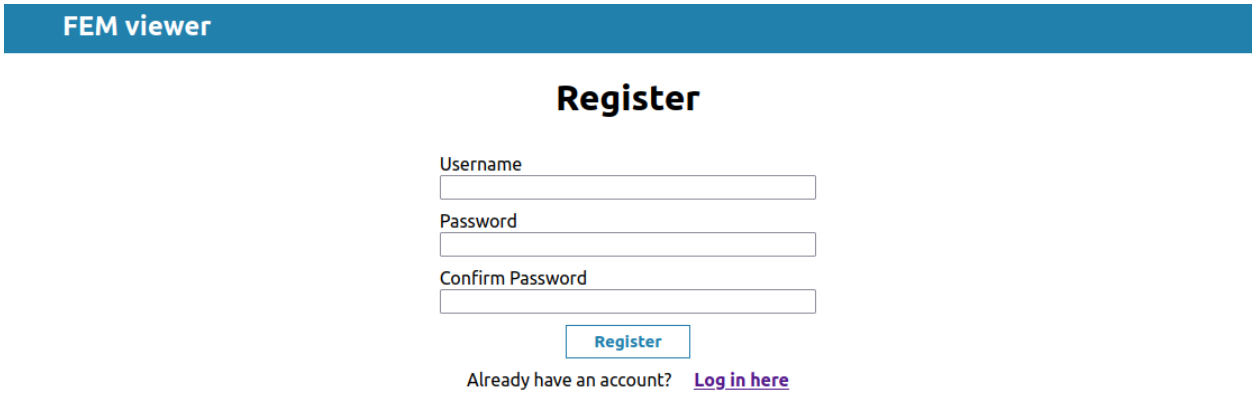

Figure 5. Register page.

5.1.4 User dashboard

The user dashboard is the part of the FEM viewer where all user-specific data for navigating and managing models is. It is separated into a header and two lists for different model groups. The header welcomes the user, letting them know they have logged in with the correct account. It also has navigation to the upload page and a button to allow logging out.

The first list contains models that the current user has uploaded themselves. The right side of a model item in this list allows that model to be shared with other users by entering the desired user's username. Validation is enforced so that users cannot share models with themselves or with nonexistent users. The left side shows the name of the model group and which users it has been shared with. It also contains a button to take the user to the model viewer page and another to delete the model group. Model group deletes cascade to shares so that when a model group is deleted, the users that have had the model group shared with them will no longer have access to that model either.

The second list is similar to the first list with the difference of having fewer privileges. It contains the model groups that other users have shared with the current user. The left side of a model group item in this list only shows the model group’s name and the button for viewing this model. The right side displays which user shared this model group with the current user.

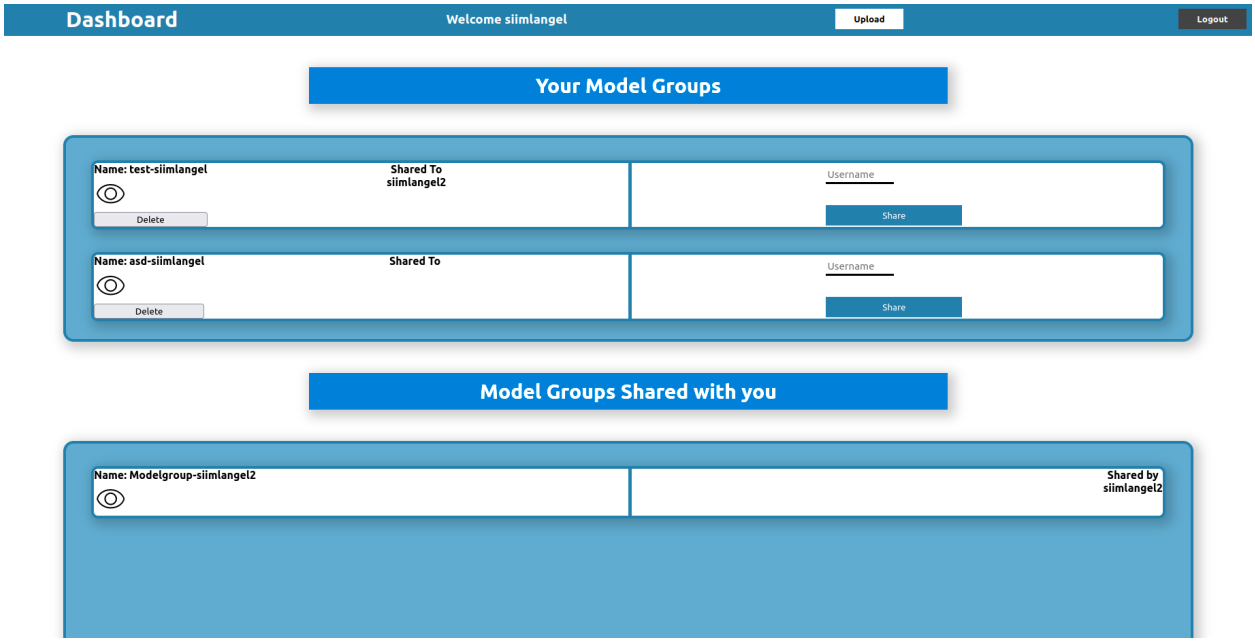

Figure 6. Dashboard.

5.1.5 Uploading

The upload page is another page requiring the user to be authenticated. It contains a header to show users that they have reached the upload page and a button to navigate back to their dashboard. The body of the page is a form for providing the necessary data for uploading a model group. It requires the user to specify a unique model group name that they have not yet used for a previous upload. It prompts the user to choose an XML file in the form of a FEM toolkit XML export. To enforce this format, the XML file is validated after upload and an error is shown to the user in case of faulty input. To validate, the application tries to parse the file into the format needed to display the model(s) it contains. The second file input requires the user to insert one or many SVG files, also exported from the FEM toolkit, corresponding to each model in the XML export.

The XML export contains the required transform-related data and data about the properties of models and their elements. However, this could be enough to rebuild models in the application.

The author found it too difficult to display the models correctly with this data alone. Thus, SVG exports are used to display the models' visuals, and XML exports handle making the models interactive.

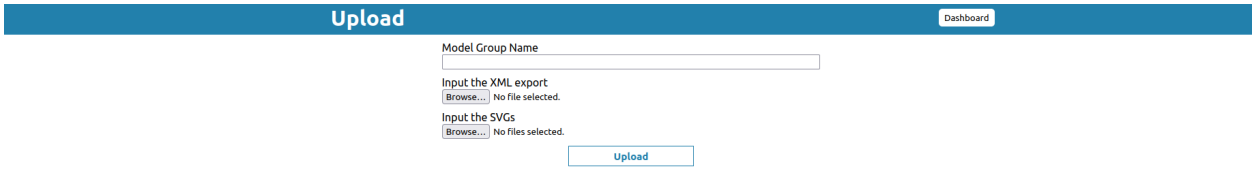

Figure 7. Upload page.

5.1.6 Model viewer page

The model viewer page contains the core functionality of the FEM viewer. It is split into three parts, a header, a model tree, and the viewer. The viewer and the model tree are divided by a draggable bar with a handle. The divider enables users to resize the viewer and the model tree. Apart from titles, the header has a scrollbar for zooming models in and out and a navigation button to take the user back to their dashboard. The model tree contains a list of all the models present in the currently viewed model group. Users can click each item in the list to select which model to display in the viewer on the right. The selected list item is highlighted to let the user know the current model. The viewer on the right is responsible for displaying a model.

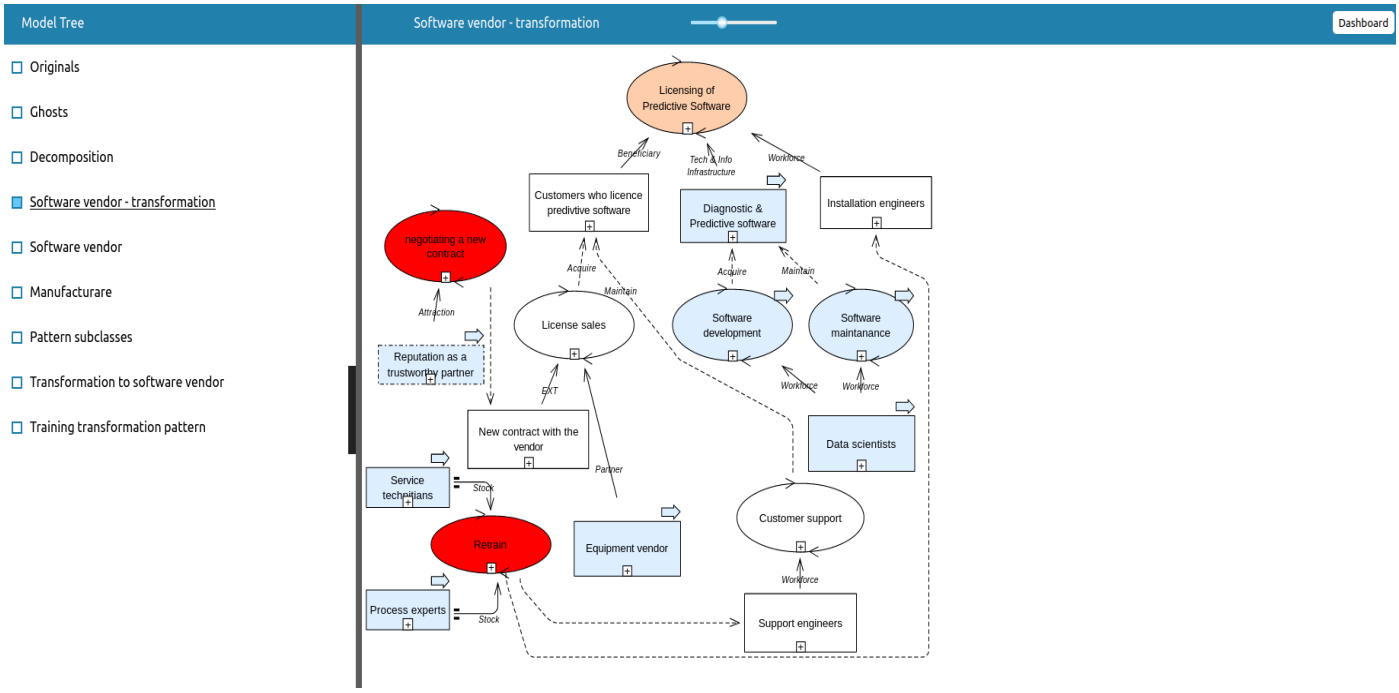

Figure 8. Model viewer page.

Users can click elements in models to select them, highlighting the currently chosen element and displaying a popup window with details about that element. The popup can be dragged around the viewer and closed by clicking away from it or on the close icon on the top right part of the window. It contains general information about properties and references for referenced elements. For sub-processes for example, a list of all occurrences is shown where the process has been referenced among all models in the given model group. For elements that reference other elements, navigation is provided to the original element.

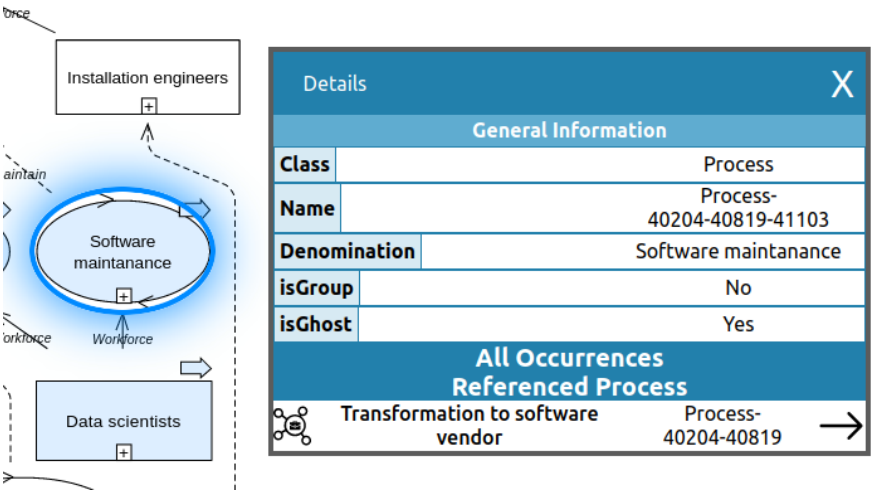

Figure 9. Model element details.

5.2 Server-side

The author and their supervisor agreed on the need for a server-side solution. It was necessary to create an authentication system and to persist data. A REST API approach was taken to exchange data between the client and the server. Fig. 10 illustrates the architecture of the server-side solution as a diagram.

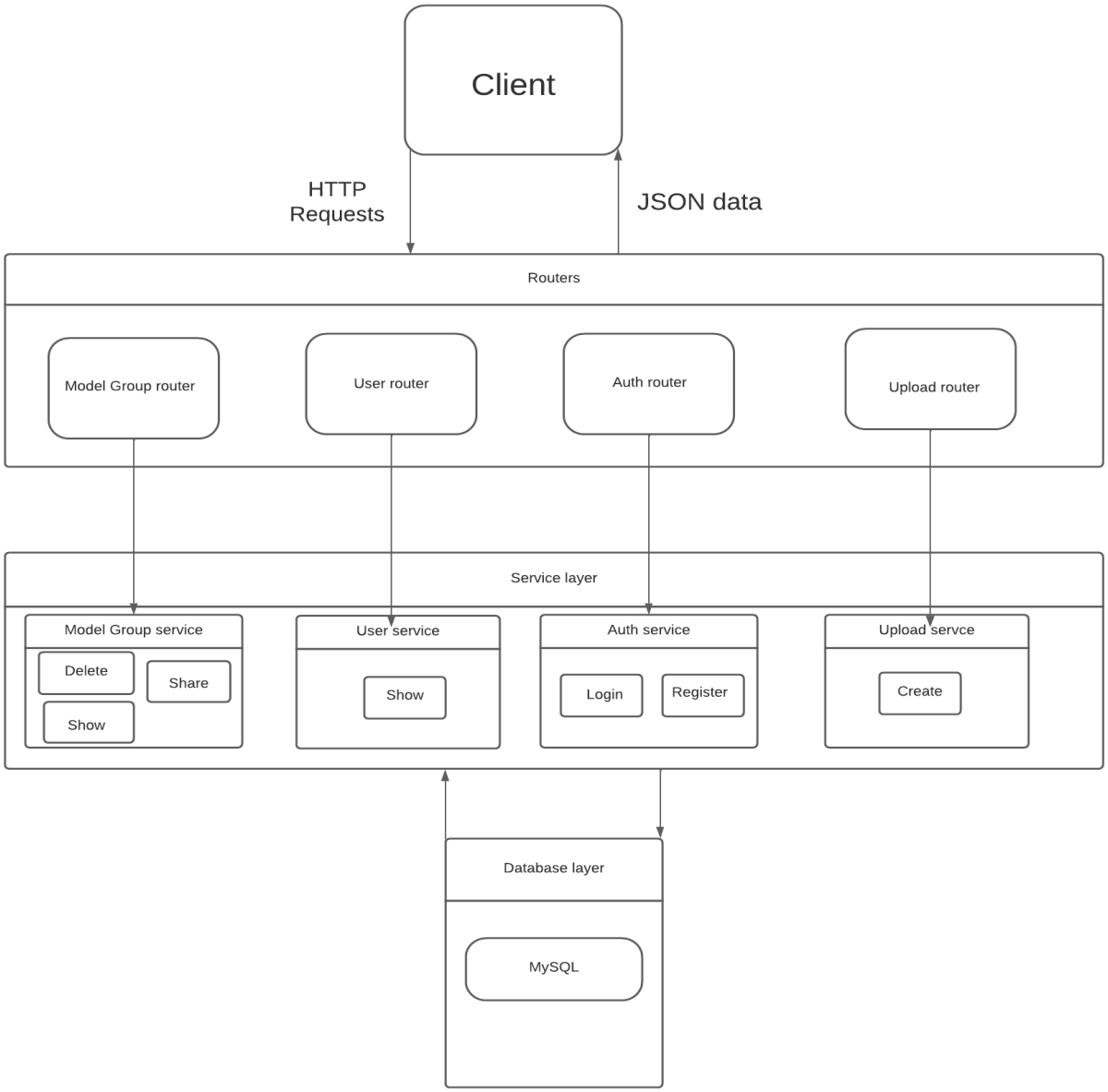

Figure 10. Server-side solution.

5.2.1 Authentication

For users to be able to use the FEM viewer, they have to first register an account with an username and a password. These two fields are then validated to ensure no duplicate user creation occurs nor that users try to register with weak passwords. To differentiate between users, their usernames have a unique constraint. After successful validation, passwords are hashed

using the bcrypt password-hashing function with 2^10 salting rounds. Hashing and salting are done so that when attackers get access to the database, they wouldn't be able to access users' accounts. Moreover, it mitigates the risk of system administrators with access to the database abusing their privilege.

When a login request is sent from the client to the server, the request includes the credentials provided by the user in plain text format. After that, the server verifies the credentials by hashing the plain text password and comparing that to the corresponding user's hashed password in the database. On success, a session is created containing the user's uuid, and a response is sent to the client with a cookie containing the session id. This cookie is sent with every further request from the client, letting the server know the user's authentication status. The cookie persists in the browser so that users don't need to log in every time they reopen their browser.

5.2.2 Routing

To manipulate and exchange data between the server and the client, the application uses routers from Express.js. They enable the server to accept HTTP requests at specified URLs using different HTTP verbs. POST requests are used to create data, GET to fetch data, PATCH to update data, and DELETE for deletions in this application's context. Routers direct these requests with data to the service layer, which handles business logic. Services then manipulate or fetch data and return the routers some data to be sent to respond to the client. As both the client and the server run on JavaScript, data is sent in JSON format. This makes it convenient to extract data from requests and responses.

5.2.3 Services

The Service Layer is an abstraction over domain logic. As mentioned in previous chapters, services are responsible for handling the application's business logic. For this application, they are implemented as JavaScript classes that inherit from a base service superclass. This superclass requires that classes extending it provide an implementation for an execute method and that elements are given access to a given request and a database connection. The "execute" method

should only be concerned with one task. For example, there are three services for manipulating model group-related data - one for showing the model group, one for sharing a model group, and one for deleting a model group. This single-responsibility approach makes business logic easier to implement and prevents unexpected errors during future changes.

5.2.4 Database design

The database model with tables and relations is shown in Fig. 11. All in all, it contains five tables to represent all the data that the FEM viewer needs.

The User table contains an id field as the primary key as a UUID. For authentication purposes, it includes a username and a password field. In the future development, a field for the role of an user could also be added for implementing role-based authorization.

The File table is used to save XML and SVG file names needed to reconstruct FEM models. The table also has an id field as the primary key as a UUID. It holds a field for the name of the file, representing the name under which the file is saved on the server's file system. It also has a foreign key field to connect it with a given model group.

The ModelGroup table holds an id in the same way as the previous tables. It also has a name field that is inputted by the user and is interpolated with the uploading user’s username to create a unique name. This field and the filename are used to save files in the file system. To do this, a directory is created with the name of the model group and its files are saved under it.

The ModelGroupsOnUsers table is used to connect model groups with users. A join table is used to create a many-to-many relationship as any number of model groups could be created by one user and then shared with other users for viewing. It also contains a field called “owner” to distinguish whether the user uploaded the model or has had it shared with them.

Finally, the Share table stores how model groups are shared between users. It has a foreign key to the ModelGroup table and two to the User table - One user who initiated the share and the other to whom the model group is being shared.

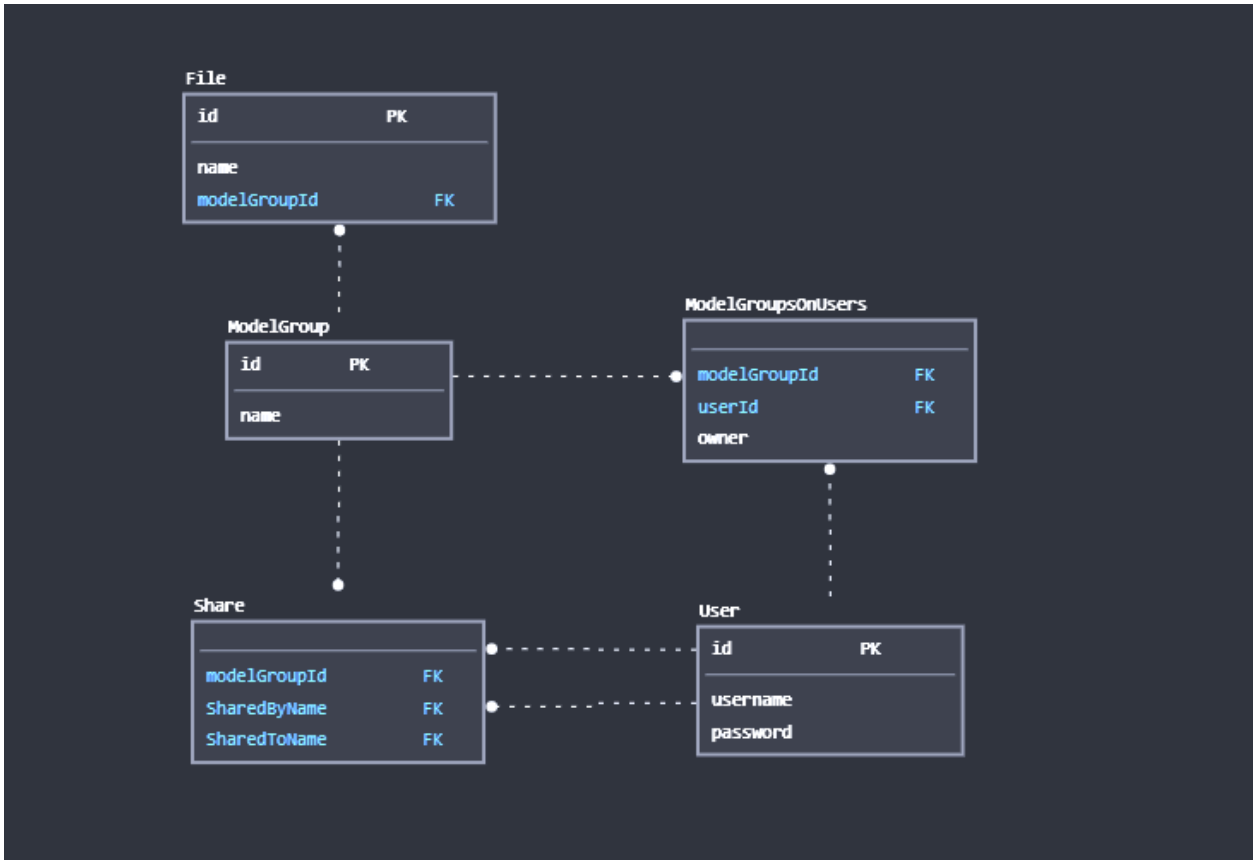

Figure 11. Database model

5.3 User Interaction

The following diagram gives a general overview of how the FEM viewer functions. It is presented in the form of an interaction diagram of UML. It outlines how a user would first create an account to start using the application and then explains the process of uploading a model and viewing and sharing it.

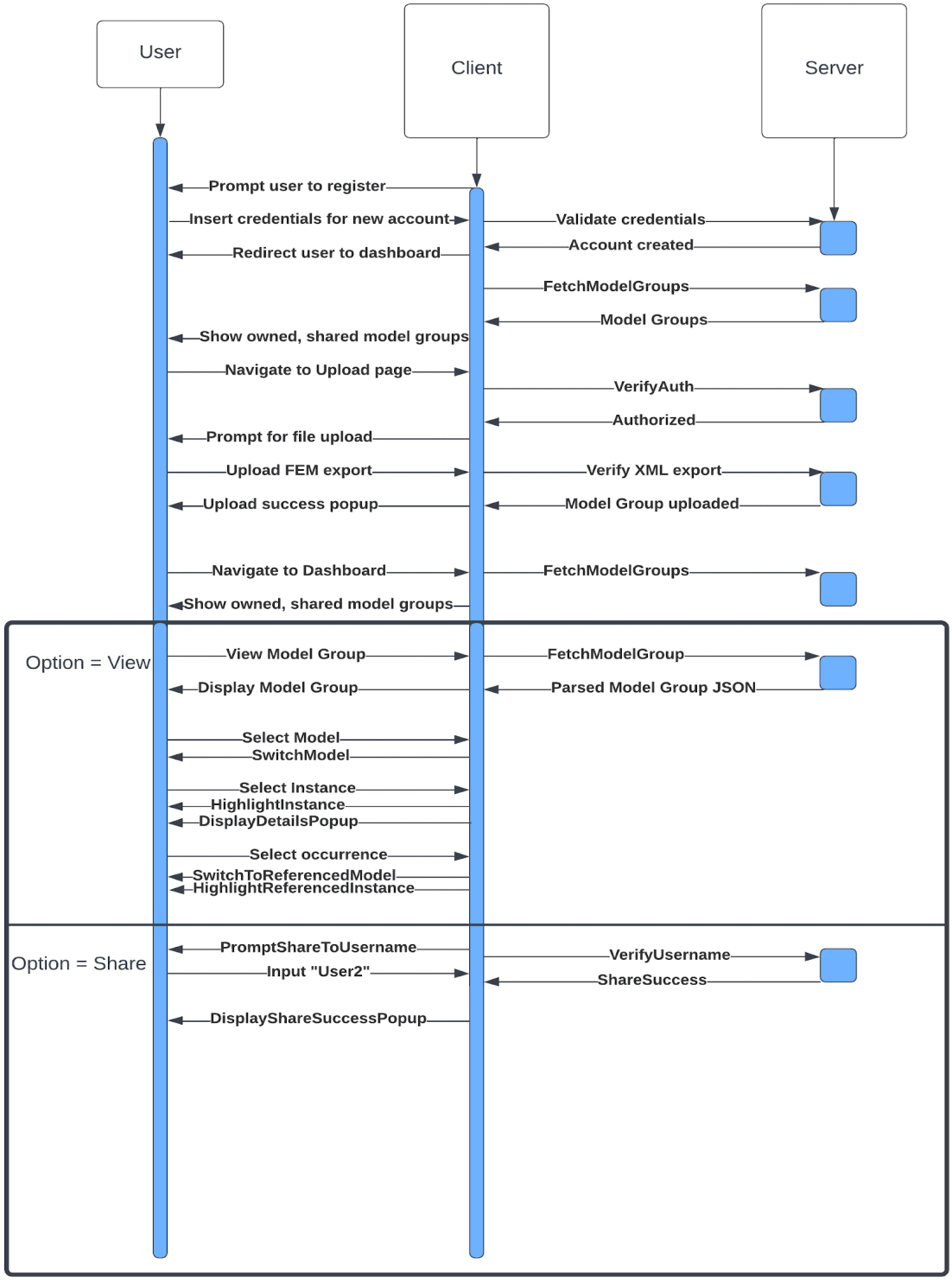

Figure 12. Interaction diagram.

5.4 Future development

The FEM viewer has prospects to be developed further in the future. The author plans to implement some new functionality and leave everything else to other interested parties.

The author will keep working on changes to user administration. Regarding this, account creation should be steered by already existing users. Furthermore, users should be split into Administrator, Developer, and Viewer roles. Administrators would be able to create any type of account and have complete control over uploading and sharing model groups. Developers should only be able to create Viewer accounts and share models with the accounts they have created. The Viewer role would be the most deprived of privileges, being only able to view models shared with them.

In the future, the application could have the functionality of creating and editing FEM models in addition to viewing them. This falls out of scope for this thesis as it is a much more time-consuming task than developing the viewer. Thus the author has decided to leave this development for anyone who wishes to pursue it.

6 Conclusion

This Bachelor's thesis aimed to create a web application that would provide a better user experience when viewing and sharing Fractal Enterprise Models than when using the FEM toolkit. It was designed to be platform-agnostic and usable without installations requiring technical knowledge.

The thesis starts by stating the problem the author is attempting to solve and some history concerning the core concepts needed to interpret the said problem. It continues by describing how and which requirements were agreed upon by the author and their supervisor, which technologies were employed and what the completed application consists of. Finally, it gives some insight into what further developments this application is subject to.

The completed application used React.js to build a rich user interface. For data manipulation and persistence, a server working on Node.js was added.

References

1. Enterprise modelling: Research review and outlook. Comput Ind. 2020, 122. http://dx.doi.org/10.1016/j.compind. (11.04.2022)

2. Bider I, Perjons E, Klyukina V. Tool Support for Fractal Enterprise Modeling. Domain-Specific Conceptual Modeling. 2022, p. 205–29. http://dx.doi.org/10.1007/978-3-030-93547-4_10 (11.04.2022)

3. The ADOxx Metamodelling Platform - Welcome to ADOxx.org - ADOxx.org. https://www.adoxx.org/live/home (11.04.2022)

4. Dimitris Karagiannis, Heinrich C. Mayr, John Mylopoulos. Domain-Specific Conceptual Modeling. 2016, 241-67. https://doi.org/10.1007/978-3-319-39417-6 (02.05.2022)

5. Dimitris Karagiannis, Moonkun Lee, Knut Hinkelmann, Wilfrid Utz. Domain-Specific Conceptual Modeling. 2022, 23-40 .https://doi.org/10.1007/978-3-030-93547-4 (02.05.2022)

6. Bider, I., Perjons, E., Elias, M. *et al.* A fractal enterprise model and its application for business development. *Softw Syst Model* 16, 2017, 663–689 . https://doi.org/10.1007/s10270-016-0554-9 (11.04.2022)

7. Bider I, Perjons E, Klyukina V. Tool Support for Fractal Enterprise Modeling. Domain-Specific Conceptual Modeling. 2022 205–29. https://link.springer.com/chapter/10.1007/978-3-030-93547-4_10 (27.04.2022)

8. Harold E, Means S. XML in a Nutshell. "O'Reilly Media, Inc.", 2004, 689 p. https://books.google.com/books/about/XML_in_a_Nutshell.html?hl=&id=_QAhsNWUSFoC (27.04.2022)

9. International organization for standardization. International Standard ISO 8879:1986: Amendment 1 : Information Processing. Text and Office Systems. Standard Generalized Markup Language (SGML). 1988. 15 p. https://books.google.com/books/about/International_Standard_ISO_8879_1986.html?hl=&id=5w-xoQEACAAJ (28.04.2022)

10. SGML. https://www.britannica.com/technology/computer-programming-language (07.12.2021)

11. React. Available from: https://reactjs.org/ (12.04.2022)

12. Gackenheimer C. Introduction to React. 2015. http://dx.doi.org/10.1007/978-1-4842-1245-5 (12.04.2022)

13. Virtual DOM and Internals. https://reactjs.org/docs/faq-internals.html (12.04.2022)

14. Components and Props. https://reactjs.org/docs/components-and-props.html (12.04.2022)

15. Introducing JSX. Available from: https://reactjs.org/docs/introducing-jsx.html (12.04.2022)

16. Node.js. https://nodejs.org/en/ (12.04.2022)

17. About Node.js. Available from: https://nodejs.org/en/about/ (12.04.2022)

18. What Is Node.js and Why You Should Use It. Kinsta®, 2021. https://kinsta.com/knowledgebase/what-is-node-js/ (12.04.2022)

19. Casciaro M, Mammino L. Node.js Design Patterns: Design and implement production-grade Node.js applications using proven patterns and techniques, 3rd Edition. Packt Publishing Ltd, 2020. 660 p. https://play.google.com/store/books/details?id=xBr0DwAAQBAJ

20. Express - Node.js web application framework. https://expressjs.com/ (13.04.2022)

21. Express/Node introduction. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction (13.04.2022)

22. 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 (13.04.2022)

23. Lorenz M, Hesse G, Rudolph J-P. Object-relational Mapping Revised - A Guideline Review and Consolidation. Proceedings of the 11th International Joint Conference on Software Technologies. 2016, 157-168. http://dx.doi.org/10.5220/0005974201570168

24. Next-generation Node.js and TypeScript ORM. https://www.prisma.io (13.04.2022)

25. Prisma schema. https://www.prisma.io/docs/concepts/components/prisma-schema (13.04.2022)

26. Prisma Migrate. https://www.prisma.io/migrate (13.04.2022)

27. Hsu S. Session vs Token Based Authentication. Medium. 2018. https://sherryhsu.medium.com/session-vs-token-based-authentication-11a6c5ac45e4 (13.04.2022)

28. Passport.js. https://www.passportjs.org/ (13.04.2022)

Appendix

I. Source code and test installation

The application's source code is hosted on GitHub and can be found at the following URL: https://github.com/siimlangel/FEM. The repository contains installation instructions and hardware requirements in the README.md on the front page. It also contains a user manual and the open source licence for using the software.

A test installation for the software can be found at http://femviewerserver.cloud.ut.ee It will be maintained for some time before and after the thesis defence. This installation is being used by third parties for testing the software.

II. Licence

I, Siim Langel,

1. Herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, Fractal Enterprise Model Viewer web application, supervised by Ilia Bider.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Siim Langel

10/05/2022