UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Bilawal Hussain

Software Platform for 3D Model Conversion and Management
Master's Thesis (30 ECTS)

Supervisor(s):

Prof. Gholamreza Anbarjafari,
Dr. Cagri Ozcinar
Timo Pastila, Ainak Oy

Tartu 2020

Table of Contents

Abstract **3**

1. Introduction **5**

2. Background **7**

2.1 Augmented Reality 7

2.2 Tools and File Types 7

2.3 Existing Solutions 9

2.4 Scope and Requirements 9

2.5 Technology Stack used 10

3. Solution **10**

3.1 Overall Architecture design 10

3.2 Frontend Application - Web Portal 11

3.2.1 Technology Stack 12

3.2.2 Architecture 12

3.2.3 Application 18

3.2.4 Testing and Planning 23

3.3 Frontend Application - 3D Model Editor 24

3.3.1 Technology Stack 24

3.3.2 Architecture 24

3.3.3 Application 25

3.3.4 Testing and Planning 27

3.4 Backend Application 28

3.4.1 Technology Stack 28

3.4.2 Architecture 29

3.4.3 Database 34

3.4.4 Application 37

3.4.5 Testing and Planning 44

4. Deployment and Testing **45**

4.1 Containerization 45

4.2 Testing 47

5. Conclusion and Future Work **48**

6. References **49**

Appendix A **50**

License **52**

Software Platform for 3D Model Conversion and Management

Abstract

In any industry, if a business promotes its products to the right people in the correct manner, it becomes the leader in that industry. Marketing has evolved over the years as the technology becomes better and better. Over the years, marketing was done in the form of door to door sales to billboards and then on electronic media. Now we are in the golden age of technology, where marketing as well as planning can be done in the form of Augmented Reality. Architects can measure and create layout plans of the space using Augmented Reality or people can select furniture by placing them in their living rooms and looking from all sides to see if it is a perfect match.

Having an Augmented Reality app is not a straightforward task. Every major field of science uses different 3D file formats to create and manage their 3D models. Using those 3D models in an Augmented Reality may not be possible as the underlying Augmented Reality frameworks may not read those formats. Adding runtime 3D models to Augmented reality applications is also a major challenge.

In this thesis, a software as a service (SaaS) solution is developed that addresses some of the issues related to Augmented reality applications. These issues are converting different 3D file formats to a single 3D file format that can be used in Augmented Reality applications and that 3D file format can also be rendered at runtime. This thesis presents the development process of a software platform where three different applications, a user interface, a 3D model editor and viewer, and a backend application, are developed. The details of the inner functionalities of each application with future work and potential improvements of the platform are also discussed.

Keywords: Augmented Reality, GLTF, FBX, OBJ, 3D Model Conversion, 3D Model Editor in a web browser

CERCS: P170

Tarkvaraplatvorm 3D-mudeli teisendamiseks ja haldamiseks

Abstraktne

Igas valdkonnas töötavaks mudeliks võib lugeda seda, et kui ettevõte propageerib oma tooteid õigetele inimestele, siis saab temast oma ala liider. Tehnoloogia aina paremaks muutumisega on ka turundus aastate jooksul arenenud. Läbi aastate toimis turundus uksest ukseni müüjatest kuni reklaamplakatiteni ja siis edasi elektroonse meedia abil. Nüüd me oleme jõudnud kuldsesse aega, kus turundus ja kavandamine on võimalik läbi augmenteeritud reaalsuse.Augmenteeritud reaalsus võimaldab arhitektidel mõõta ja luua ruumile paigutusplaane, samuti saavad inimesed valida mööblit, paigutades selle oma elutuppa ja vaadates seda iga nurga alt, et olla kindel, et see on sobib perfektselt.

Augmenteeritud reaalsuse rakenduse tegemine ei ole lihtne ülesanne. Iga suurem teadusvaldkond kasutab erinevaid 3D failiformaate oma 3D mudelite loomiseks ja juhtimiseks. Nende 3D mudelite kasutamine augmenteeritud reaalsuses ei pruugi võimalik olla kuna augmenteeritud reaalsuse alusraamistik ei loe neid formaate. Kestvusaja/tööaja 3D mudelite lisamine augmenteeritud reaalsuse rakendustesse on samuti suur väljakutse.

Käesolevas lõputöös töötatakse välja teenuse lahendusena tarkvara, mis adresseerib mõningaid augmenteeritud reaalsuse rakendustega kaasnevaid probleeme. Nendeks probleemideks on erinevate 3D failiformaadite üheseks 3D failiformaadiks, mida saaks kasutada augmenteeritud reaalsuses, teisendamine ja et seda 3D failiformaati saaks ka töö ajal muuta. See lõputöö tutvustab tarkvara platvormi, kus töötatakse välja 3 erinevat rakendust: kasutaja interface, 3D mudeli töötleja ja vaatleja ja backend rakendus, arendamisprotsessi. Autor samuti arutab iga rakenduse sisemiste funktsionaalsuste detailide üle koos tulevikus teha vajava tööga ja platvormi potentsiaalste parandustega.

Keywords: Liitreaalsus, GLTF, FBX, OBJ, 3D-mudeli teisendamine, 3D-mudeli redaktor veebibrauseris

CERCS: P170

1. Introduction

Economic growth and human progress depend on technological innovation. As human progress depends on technology, so in the past few decades, technology has grown tremendously and revolutionized the work processes across major industries. Especially Computer Science technology has grown so much that it is now considered as a fundamental building block in almost every major industry including medical, education, telecommunication, construction, automobile manufacturing, renovation, warehouse designing, and planning, etc.

Augmented reality (AR) is a new technology that consists of adding virtual elements coherently in a real scene [1]. It is one of the fields of computer science that has seen growth at a very fast pace in recent years and now almost all of the major industries are rapidly adopting it in their business process.

In every major industry, the marketing of the product is one of the most important aspects of the business. If the product is marketed in a good way to the right audience then the chances increase significantly that the industry will make business. Marketing becomes a real challenge for the industries that sell huge products that either weigh a lot or very big in size. Most of the industrial equipment is sold after careful planning and precise measurements where it will be installed that make the whole process slow and very time-consuming. The same is true for the industries where public safety or city layout is concerned. For example, a telecommunication industry plans to install their wireless signal posts on the street to increase the coverage of their services. They will need to visit the site and do the measurements approximately and report back if it is safe to install and it is not obscuring anything important or it is not in the way of other objects nearby. Usually, after the report, the engineers place one of their poles and evaluate if it satisfies all the requirements if not then they have to remove it back. This process is laborious, time-consuming, and more importantly very costly. The AR solves this issue elegantly and reduces the time for deployment. This example is taken from a real-world use case of a tech company named Ainak Oy [2] based in Finland and this thesis is provided and supervised by it.

Both of the examples mentioned earlier can be solved by using an AR application. For industrial equipment, the AR application can be used to precisely plan the layout with measurements and then install the equipment on the user premises to evaluate if the equipment fits in the designated place. If it fits then the actual equipment is transferred and installed. For the telecommunication example, the engineers can visit the site,

install the pole virtually in the real-world environment, and evaluate if it fits their standards. The AR application can reduce time and cost tremendously and help the industries to digitize their workflow.

To provide an Augmented Reality application to different industries is a challenge as they have their own needs and requirements. One of the basic problems is that every industry uses a different 3D modeling software and has different 3D model file types. The most popular 3D formats are DAE, STP (STEP), FBX, and OBJ [3], and nowadays GLTF as it is marketed as *a jpeg of 3D file format* [4]. 3D file formats are discussed in section 2.2 of this thesis. All of the formats are not supported by the underlying libraries and software for augmented reality so to support different industries 3D models should be converted to a standardized format. The second problem that arises is that the content of the application should be dynamic that means the AR application users can use different 3D models by themselves. The AR application should support the runtime loading of 3D models.

To solve the above-mentioned problems the intuitive solution is to create software where users can upload their 3D models and the software converts them into supported 3D models for the AR application. The software will also provide a basic viewing and editing of 3D models. The user can upload any number of 3D models and choose which ones should be available in the AR application. The software is a SaaS and can be accessed from the web.

This thesis presents the development of the frontend, backend, and 3D editor application of Ainak Oy which is built in three different tech stacks, and they are discussed in detail in sections 3.2, 3.3 and 3.4.

This thesis is divided into 6 chapters. Chapter 1 provides the introduction followed by chapter 2 that discusses the background, tools, and file types, existing solutions and concludes with what technological stack was chosen to develop this software. Chapter 3 goes into detail of the solution describing the architecture, requirement gathering, and implementation details of the software. Chapter 4 discusses how the software was deployed and tested. The conclusion of the thesis is provided in chapter 5 along with future work.

2. Background

This chapter of the thesis includes a brief discussion on AR along with tools and file types used in AR applications, use cases, existing solutions, scope, and requirements of the software.

2.1 Augmented Reality

Virtual Reality (VR) is a computer-generated three-dimensional content that may or may not look like the real world. It is a simulation that can be interacted physically using handheld devices. AR is an extension of VR. By contrast to virtual reality, augmented reality is imposed on the real world.

AR combines both the virtual world with the real world in a seamless manner. Users can interact with the objects, placed in AR, that are interposed on the real scenes around them. This provides a unique blend between the virtual and real-world and opens many use cases where this can be leveraged. AR is widely used in education, marketing, and entertainment industries.
AR applications usually need a camera and a computer to work. The computer can be a small device like smartphones or it can be wearable glasses like Microsoft Hololens [5], or it can be an ordinary computer.

2.2 Tools and File Types

The AR applications mostly depend on the 3D content provided by the user. This 3D content can be a 3D Model or any other three-dimensional representation of real or virtual world objects. In most cases, it is a 3D model that is generated by 3D modeling software. The most commonly used software to generate 3D models is Blender, Solidworks, 3DS Max, Cinema 4D, etc. Most of the tools produce file types that are understood by them and there are some file types that are compatible with one or more software. There are many 3D file formats but four of them are discussed here as they are most often used in AR applications.

FBX is one of the most commonly used 3D model file formats that can be imported in almost every major 3D software. FBX is owned by Autodesk [6]. Usually, the FBX file format consists of only one file that embeds the shape, geometry, textures, materials, and other information in it. Due to its high interoperability between different 3D modeling software, it is considered as a gold standard file format. As the FBX is not actually an

open-source so most of its implementation is hidden and it is very hard to create loaders and readers for this file format.

Another very popular 3D file format is OBJ. This file format is usually accompanied by another file called material file, MTL. The OBJ file usually holds the information about the shape, geometry, vertices, and normals and the material file (MTL) file holds information about the color and other texture information. Optionally this file format is also accompanied by image files that are then overlaid on the shape of the model. A typical file structure of the OBJ model is shown in figure 1. This file format also has high interoperability between 3D modeling software.

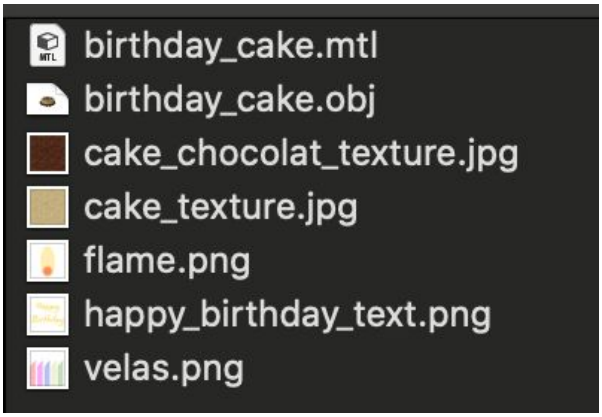


Figure 1: File structure of OBJ model

DAE is another popular file format that is mainly used in CAD software. It is like FBX for CAD software. Its interoperability is high and can be used in almost all major 3D modeling and CAD software. It is a single file format that consists of all the information related to the model like shape, geometry, and colors in it.

GLTF is another file format that is gaining popularity at a very high pace in the 3D development community. It is built by Khronos Group [7] and it is fully open source. All the specifications are written and available for public use. This file format competes with FBX file format as it can contain all the information like shape, geometry, texture, material, and other information in a single file usually called GLB. Its interoperability is also high and almost all of the major 3D modeling software load, understand and export this file format.

Which 3D file format to use generally depends on the platform that is being used to create the AR application. In Ainak Oy, the company that provided this thesis, the main platform to create the AR application is Unity3D. The unity3D supports FBX, OBJ, and

DAE natively so their AR application supports only FBX. All of the other 3D file formats are converted to the FBX and then used in the AR application. As the company grew and started to add other services, one of them is Layout Planning, the FBX file format became the bottleneck as it was not usable in the Layout Planning application. So, a case study was done and the author of this thesis proposed that the company should use GLTF as their main 3D file format as it will be usable in all of their services.

The author of this thesis then implemented an automatic file conversion tool that converts FBX, OBJ with MTL, and other texture files to GLTF file format. This conversion tool is discussed in detail in section 3.4.

2.3 Existing Solutions

Market research was done by the author of this thesis in regards to finding software that will provide similar functionality. There was one software found that was studied and discussed with the Ainak Oy. That software was an IKEA AR application [8]. Please keep in mind that the market search that was done was related to AR mobile applications. The IKEA app provided the users to place furniture in their houses and see if it fits there or change the color of the 3D model to check if it looks better. The IKEA app also provided the ability to the user to move and rotate the 3D models. This research also found out that the 3D models in the app were limited and most probably they were embedded in the application meaning the application does not provide run time 3D model download feature.

On the other hand, the software solution that is built by the author of this thesis enables Ainak Oy to support multiple 3D models that can be downloaded and rendered at runtime. Section 3.1 of this thesis discusses the architecture in detail.

2.4 Scope and Requirements

Requirements for the software to be developed in this thesis came from the Ainak Oy. The requirements were based on industry case studies then all of them were documented and the requirements that were related to this thesis were shortlisted and then refined so that the author of this thesis can work on them. The requirement gathering process is discussed in detail in section 3.2.

The scope of the software that is built in this thesis consists of three separate projects consisting of one front-end web application, one ThreeJS (3D rendering) application, and one NodeJS application for the backend.

The front-end web application can be used for user management, adding child accounts, uploading models, making models available in AR applications, and so on. The details are discussed in section 3.3.3

The ThreeJS application is also a web application that is developed and deployed separately and it enables the user to change the rotation of the 3D model, change the pivot point of the 3D model and delete some parts of the 3D models. The details are discussed in section 3.3.

The NodeJS application is built to expose restful APIs that are consumed in both the front-end application and the ThreeJS application. MySQL is used for the database. The details are discussed in section 3.4.

2.5 Technology Stack used

The technology stack that is used to build software follows:

- The Front-end application is built using the VueJS framework. Other UI libraries were also used in conjunction with VueJS, for example, Axios to interact with backend, Buefy for UI components, etc.
- A 3D model rendering and editing application is built using ThreeJS.
- A NodeJS express application was built to handle the backend of this software.
- MySQL was used as the database for this application.
- Docker was used to containerizing the backend application and then used it in deployment.

The next chapter of this application explains all of the above in detail justifying why these choices were made.

3. Solution

In this chapter of the thesis, a detailed explanation is given on the process of developing this software. Discussing all the modules in detail individually and justification is provided on why the specific library or framework was used.

3.1 Overall Architecture design

The software consists of three different applications running in parallel. These applications can communicate with each other to transfer related data to carry out the task. The overall architecture is shown in figure 2.

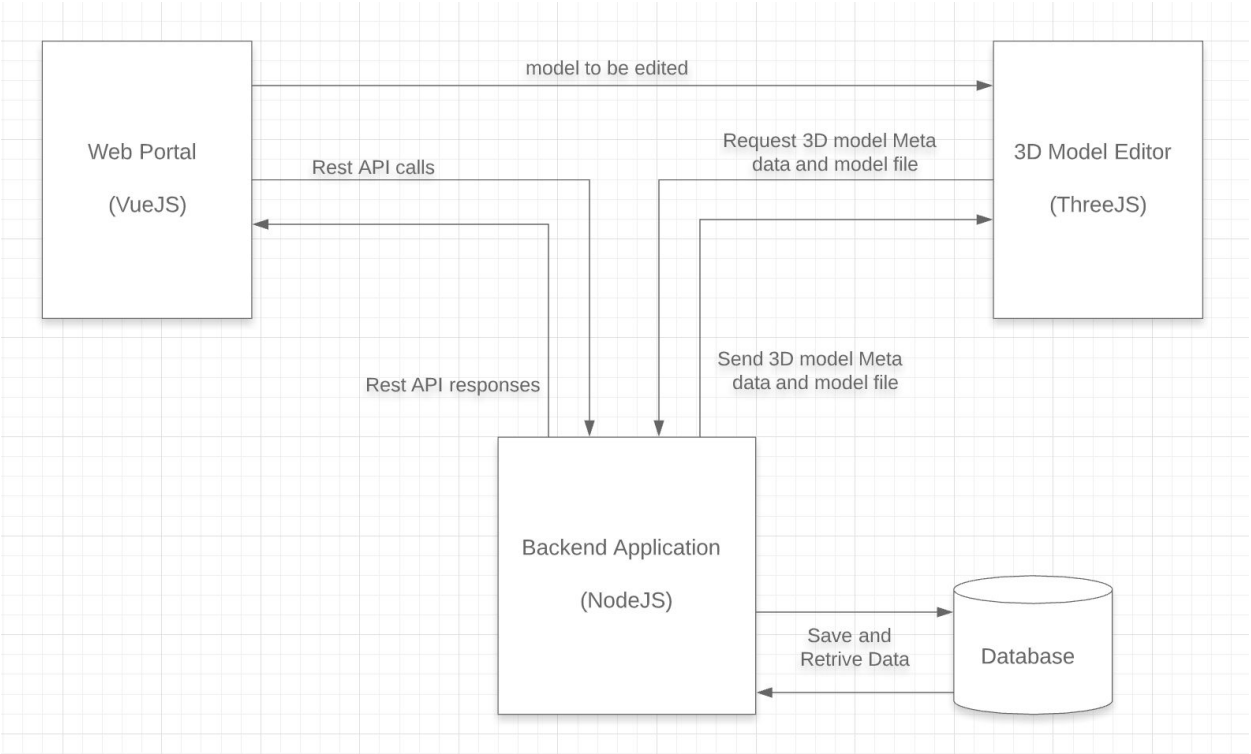


Figure 2: Overall Architecture design

Due to its complexity, the agile software development approach was adopted. The tasks were broken down into smaller tasks and were divided into two weeks sprints. At the end of each sprint, acceptance testing took place where the stakeholder of the task checked if it is working as it is supposed to be and if everything worked then the new task was chosen for the next sprint with acceptance testing requirements.

For deployment, Docker was chosen to containerize the backend application and then deploy it to the live server. The details are discussed in section 4.

For version control, Gitlab was used for all of the applications built in this thesis. A comparison was done between Github and Gitlab and then Gitlab was chosen because it allows unlimited private repositories and hence helped the company in saving money.

3.2 Frontend Application - Web Portal

The front end of this software consists of two applications. This section of the thesis discusses the web portal application in detail.

The web portal application is built using the VueJS framework in conjunction with other libraries like Buefy for UI components and Axios for handling rest API calls. The VueJS application is used to upload, sign in, register, upload 3D models, and make those 3D models in AR application hence it was named *web portal* application.

3.2.1 Technology Stack

There are many frameworks that can be used to create front end applications. The most famous frameworks are ReactJS, Angular, and VueJS. A small case study was done on all of the frameworks mentioned earlier. The main findings are discussed below.

Angular is intended for large applications as it is shipped with features like tempting, routing, state management, etc out of the box which makes it heavyweight compared to the other two frameworks. On the other hand, React and Vue at their core just represents the view layer and thus considered lightweight. They are not shipped with routing, templating, or network calling APIs. Both are good choices for this application.

VueJS was chosen to be the framework for the front end application. The main reasons were as follow:

- **Learning curve**: React has a steep learning curve as compared to the Vue. So to save time and start the development sooner VueJS was the perfect choice
- **Future development**: React and Vue are both future proof frameworks. But VueJS was preferred as it was easier to find new employees after the author of this thesis left the company.
- **Experience**: The author of this thesis had prior experience in the VueJS as it was taught in his Master's curriculum. This also influenced the decision.

3.2.2 Architecture

The web portal application was built as a Single Page Application (SPA). Single page application runs inside the browser and when a user interacts with it the content of the page is dynamically changed and the page is never reloaded. This provides a faster and fluid UI as all of the renderings are done on the user's browser locally. The page requests new information from the server when the data is stored or fetched from the server.

The architecture of this application is component-based. All the components related to each other are grouped together and they communicate with each other to perform the tasks. For example, the upload component interacts with the home component when

the upload is done to update the UI. All of the functionality is done in small component structures and these components are built to be reusable.

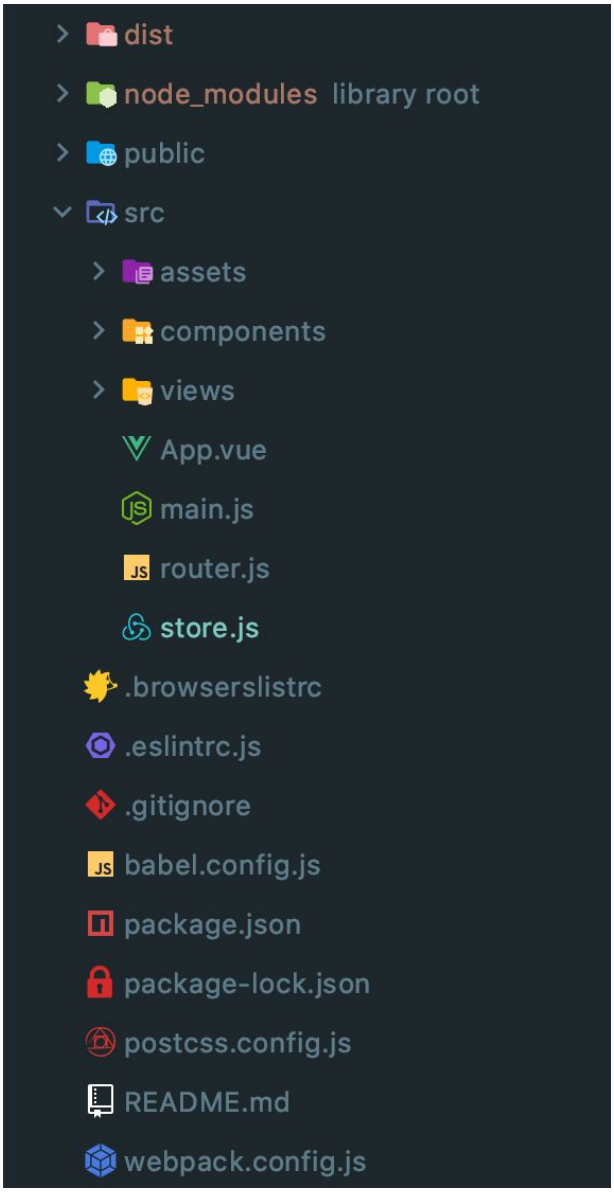


Figure 3: Project folder structure of Web Portal Application

Figure 3 shows the project structure and all the folder and files are described as follow:

- **dist:** This folder is generated when the application is built. This contains the files that are used to host the application on a live server.

- **node_modules:** This folder contains all the node js libraries used in the project. This folder is generated when the libraries are installed in the project.
- **public:** This folder contains the public files and assets of the application. This includes but not limited to *index.html*, the entry point for the application, logo, and icon of the application.
- **src:** This folder contains all the logic and components of the application.
    - **assets:** This folder includes the static assets of the application that can be a company logo, images, fonts, etc
    - **components:** This folder contains all the components of the applications. This folder can be subdivided into smaller folders. Most of the UI work is done here.
    - **views:** This folder contains files that are composed of components. A file can have many components in it. For example, a header component, a body component, and a footer component.
    - **App.vue:** This file is the root component where all the other components are nested. This file is responsible for handling all the components. Generally, a router component is passed to it. This file is also used to override Styles and can be used to provide a reset stylesheet.
    - **main.js:** This file is the entry point for the application. Here all the libraries are initialized and passed to the Vue. It adds and renders the App.vue to DOM (Document object model).
    - **router.js:** This file holds the URLs of the application.
    - **store.js:** This file stores all the persistent data in the application for the given session. This file is used by *Vuex* library that is discussed in the section 3.2.3
- **.browserlistrc:** This file is used to define which browsers are supported for this application
- **.eslintrc.js:** This file is used to control the code quality. This provides rules that should be followed in order to get a consistent code. For example, use single-quotes everywhere.
- **.gitignore:** used to tell the version control system which files should be ignored.
- **.babel.config.js:** this configuration file is used to compile the ES6 code.
- **package.json and package-lock.json:** These files are used to manage the dependencies used in the project.
- **postcss.config.js:** This configuration file is used to compile SCSS to CSS.
- **webpack.config.js:** This configuration file is used to map folders in the project to import them easily. For example, if a component is at the location of *src/folderA/folderB/file.vue* and it is used everywhere then an alias can be defined to access it easier like *@/file.vue*

In order to build the application, other libraries are used in conjunction with the core VueJS library. The libraries are defined in the package.json file. Figure 4 shows the package.json file.

```
{
  "name": "ui",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "serve": "vue-cli-service serve",
    "build": "vue-cli-service build",
    "lint": "vue-cli-service lint"
  },
  "dependencies": {
    "axios": "^0.18.0",
    "buefy": "^0.7.2",
    "epic-spinners": "^1.0.4",
    "jwt-decode": "^2.2.0",
    "vue": "^2.5.21",
    "vue-agile": "^0.3.7",
    "vue-plyr": "^5.1.0",
    "vue-router": "^3.0.1",
    "vue-tour": "^1.1.0",
    "vuex": "^3.0.1",
    "vuex-persistedstate": "^2.5.4"
  },
  "devDependencies": {
    "@vue/cli-plugin-babel": "^3.3.0",
    "@vue/cli-plugin-eslint": "^3.3.0",
    "@vue/cli-service": "^3.3.0",
    "babel-eslint": "^10.0.1",
    "eslint": "^5.8.0",
    "eslint-plugin-vue": "^5.0.0",
    "node-sass": "^4.9.0",
    "sass-loader": "^7.0.1",
    "vue-template-compiler": "^2.5.21"
  }
}
```

Figure 4: package.json of Web Portal Application

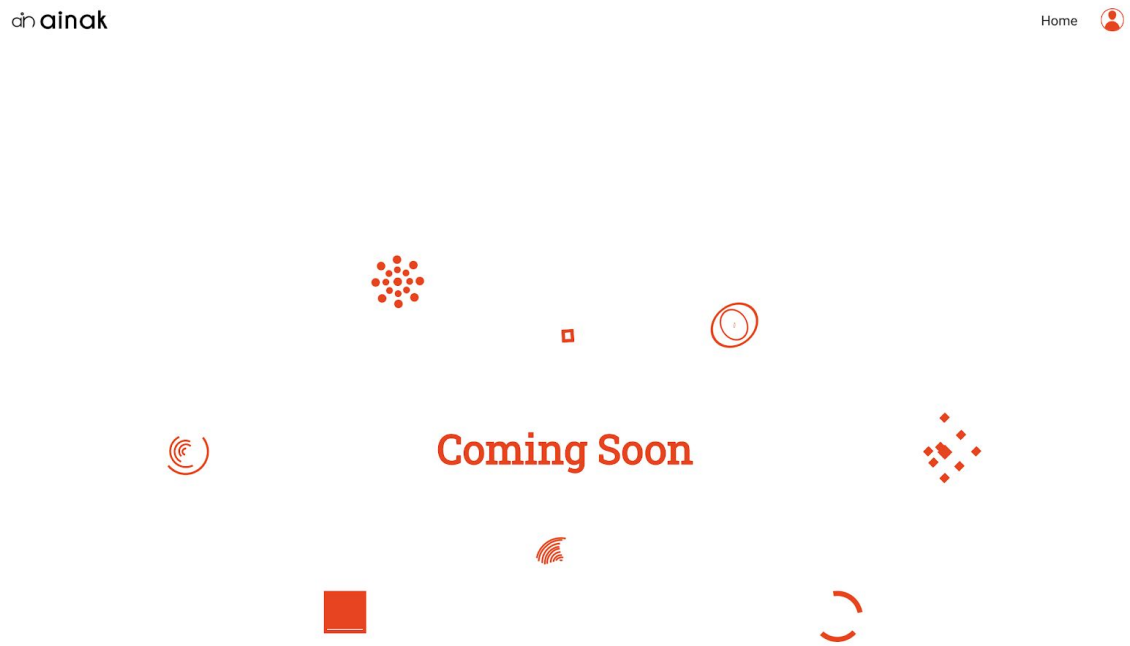
There are two sections for dependencies. One is *dependencies* and the other is *devDependencies.* devDependencies are used to control the development process, for example, ensure the code quality using eslint, etc. These libraries are not discussed in this thesis as they do not impact the outcome of the application.

The main dependencies are discussed in detail below:

Axios:
This library is used to call the Rest APIs. It is a promised based HTTP client. It is used to asynchronously handle API calls.

Buefy:
This is a lightweight UI library that provides pre-built UI components. There are many other UI libraries like this one but this was preferred because the author of this thesis had prior experience in it as it was used in his Master's curriculum

Epic Spinners: This is a UI component library that was used to show feedback to the user. For example, when the user goes to a page that is not available or it is coming soon. Figure 5 shows an example. The live example can be found here: https://ainak-2019.firebaseapp.com/soon



.

Figure 5: Showing Spinner using Epic Spinners library

JWT Decode:
This library is used to decode the JWT tokens that are sent by the backend to check if they are still valid or not.

Vue Agile:
This library is used to show the featured images/Text to the customer on the Landing page and/or other places in the application

Vue Plyr:
This library is used to play a youtube video whenever a customer logs in for the first time. Figure 6 shows the use of this library.

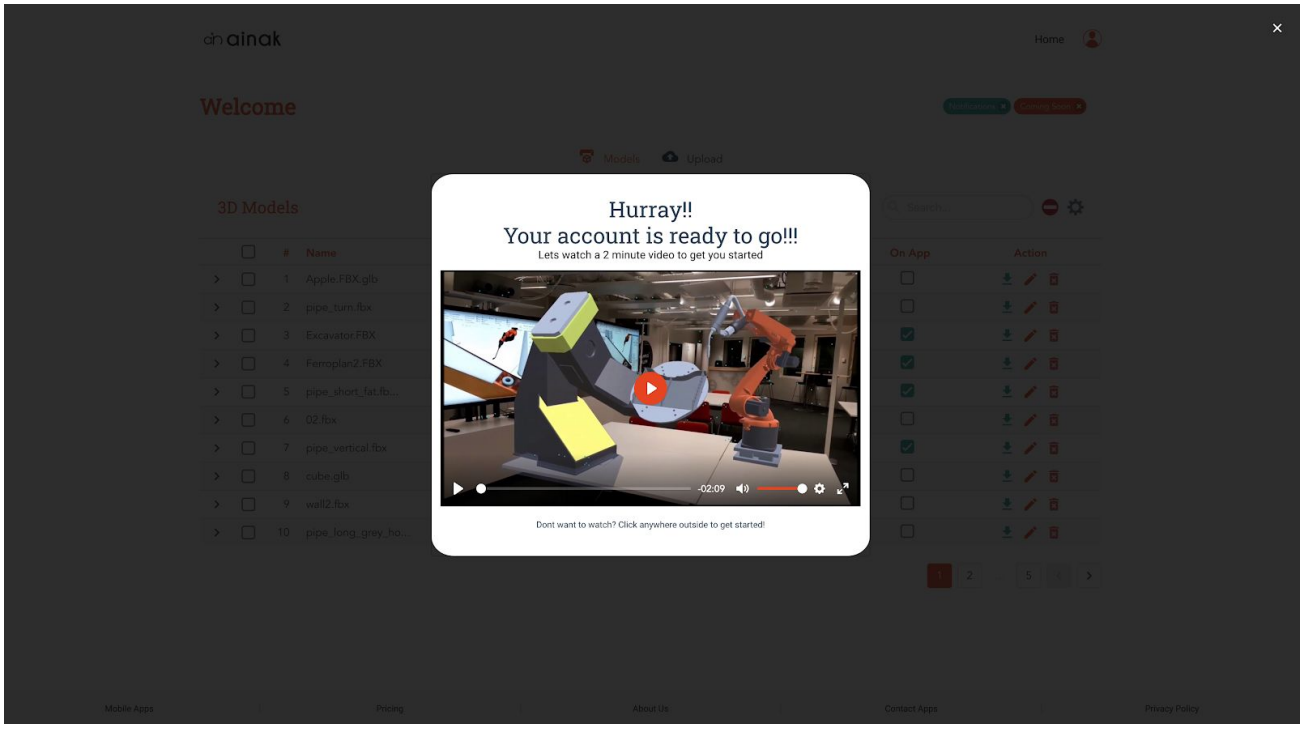


Figure 6: Showing video using Vue Plyr library

Vue Router:
This library is the official router for the VueJS applications. A router is responsible for syncing the address from the URL bar to the view displayed in the browser window.

Vue Tour:

This library is used to onboard new users to the platform by showing them how to navigate and use the application. It is used to show small messages describing what is the purpose of the component. Users can skip it at any time. Figure 7 shows the use of this library.

Figure 7: User onboarding using Vue Tour library

Vuex and Vuex persistence state:
This is the official state management library for the VueJS applications. This library is used to store the user's API JWT token and cache data for other components like the list of uploaded models.

3.2.3 Application

This section of the thesis discusses in detail the inner functions of the application.

Application Entry point:
As discussed in section 3.2.2 *main.js*, as shown in figure 8, is used as the entry point of the application. Here an instance of Vue is created and it appends itself to the DOM (document object model). All of the other libraries are also initialized here and passed to the Vue instance. Now *main.js* is responsible for rendering the *app.js* and appending it to the DOM. All other components and views will be nested to the *app.js* and then rendered in the web browser.

```
import Vue from "vue";
import App from "./App.vue";
import router from "./router";
import store from "./store";
import Buefy from 'buefy';
import VuePlyr from 'vue-plyr';
import VueTour from 'vue-tour'
import VueAgile from 'vue-agile'

Vue.use(VueTour);
Vue.use(VuePlyr);
Vue.use(Buefy);
Vue.use(VueAgile);

Vue.config.productionTip = false;

new Vue({
  router,
  store,
  render: h => h(App)
}).$mount( elementOrSelector: "#app");
```

Figure 8: Structure of main.js

User Interface:

The UI part of the application is divided into two parts. These parts are *views* and *components*. The views contain one or more components. A component may contain other components but usually, the components are made so that they can be reused. Both views and components share the same structure that is shown in figure 9.

```
<template ... >

<script ... >

<style lang="scss" scoped ... >
```

Figure 9: Structure of a typical component in VueJS

The *template* part hosts all the UI elements, the HTML part of a typical website. The *script* part contains the behavior of the application that contains button click events, API

call to fetch, or post data to the server. The *style* part contains all the styling of the component, the CSS part of a typical website.

The *views* have the exact same structure but they can contain different components in their template. For example, a view can have a header component, a body component, and a footer component. Figure 10 shows a view like this.

```
<template>
    <div class="main">
        <!-- header -->
        <div class="main-header container">
            <ainak-header/>
        </div>

        <!-- body -->
        <div class="main-content container">
            <home-content/>
        </div>

        <!-- footer -->
        <div class="main-footer">
            <ainak-footer/>
        </div>

        <!-- Absolute elements to show in the center of screen -->
        <b-modal :active.sync="isNewUser" class="new-user-modal" @close="handleOutsideClick" ... >
        <home-tour></home-tour>

    </div>
</template>

<script ... >

<style lang="scss" scoped ... >
```

Figure 10: Structure of View component

Navigation:

The navigation is handled by the *vue-router* library. To define all the routers available in the application a *router.js* file is added and then it is supplied to the Vue instance that was created when the application was loaded. The sample route from the *router.js* file is shown in figure 11.

```
{
  path: "/login",
  name: "Login",
  // route level code-splitting
  // this generates a separate chunk (about.[hash].js) for this route
  // which is lazy-loaded when the route is visited.
  component: () ⇒
    import(/* webpackChunkName: "Login" */ "./views/Login.vue")
},
```

Figure 11: Structure of a typical route

All of the routes are loaded using a lazy loading technique. Lazy loading means that the resources are loading beforehand and are loaded on demand. This dramatically reduces the application loading time as the whole application is not loaded on the initial render.

State management:

Vuex is used for state management of the application. It is the official state management tool for VueJS applications. State management can be defined as storing the user information for the current session to reduce network traffic and store user-specific information in the browser for later use. To define this storage, a *store.js* file is created in the application directory. This file includes all the information that is needed for the application to work. Depending on the application complexity and size there can be many stores but in this case, the application was small enough to just have one store. Figure 12 shows the structure of a store.

```
export default new Vuex.Store( options: {
    // State = the data
    state: {_apiURL: 'https://ainakcloud.herokuapp.com' ... },
    // Mutations = Editing the data (state)
    mutations: { ... },
    // Actions = Functions that call the mutations that edit the data (state)
    // (API calls to the server goes here)
    actions: {...},
    //Getter = Get the state data (computed/filtered/processed data from the state)
    getters: { ... }
});
```

Figure 12: Structure of a Vuex Store

The Vuex state management is unidirectional that means the data can flow in only one direction.

The *state* is where all the information is stored. It is also called a *single source of truth*. *Mutations* are the functions that can change the state. There is no other way to change the state. It ensures that when the state is updated it is intentional so it prevents any component to update the state. *Actions* are the functions that are used to interact with the backend using API calls and they are called from the UI. *Getters* are the functions that have access to the state but they can only read it and cannot change it. They are mostly used from the UI to access the state. Figure 13 describes how the Vuex works in a typical VueJS application.

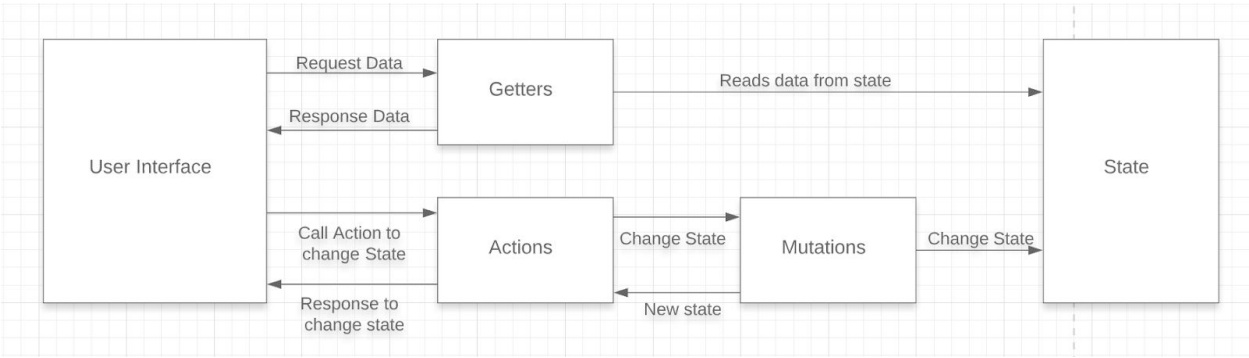


Figure 13: Vuex structure in VueJS applications

UI and UX:

Buefy, a UI library, was used to create the user interface of the application. Buefy contains a set of UI components that are made using best practices and provides similar look and feel for a smooth UI experience. Figure 14 shows the home page of the application.

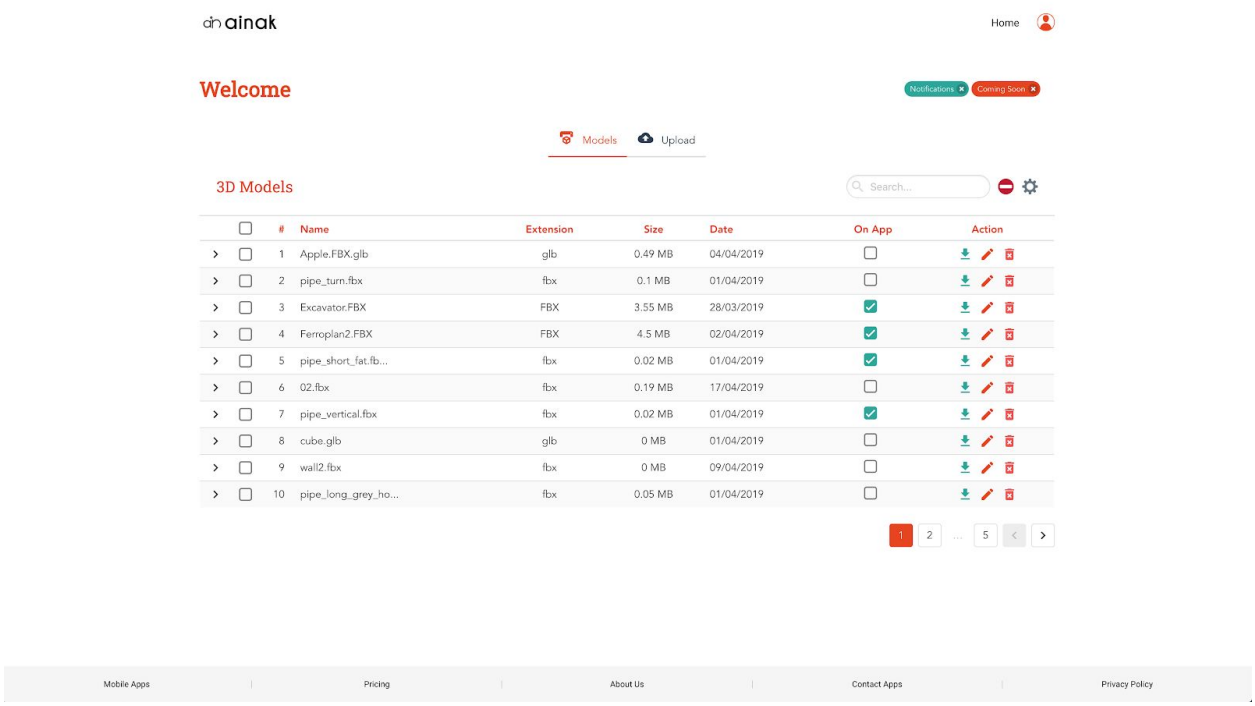


Figure 14: Homepage of Web Portal Application

A lot of attention was given when designing the user experience for this application. The UX (user experience) was made as linear as possible so confusion can be avoided. The UI remains the same on all of the pages and provides a consistent UX to the users.

3.2.4 Testing and Planning

UI testing was not done automatically. A browser was used to test and build the required features. Then the developer, the author of this thesis, and the product owner worked collectively to test the features. The testing took place on every sprint review. If the feature fully passed the acceptance testing requirements then the feature was marked as complete otherwise a list of bugs or unexpected behavior was given to the developer to fix.

After every successful acceptance testing, a new feature was planned for the next sprint and acceptance testing requirements were given to the developer. If the developer had any issues or questions with the feature or acceptance testing requirements they were communicated with the product owner and a resolution was made.

3.3 Frontend Application - 3D Model Editor

This section of the thesis discusses the 3D model editor application in detail. The ThreeJS application is used to do basic 3D model viewing and editing like deleting parts of the 3D model, changing its rotation or pivot point hence it was named *3D model editor* application.

3.3.1 Technology Stack

There are many 3D libraries available to create a 3D web application. They are based on WebGL. WebGL, Web Graphics Library, is a javascript API that is used to render interactive 2D and 3D graphics in compatible web browsers. Implementing a plain WebGL application has a steep learning curve. 3D libraries that are based on WebGL mostly do the heavy lifting and let the developer bootstrap applications in a relatively short time.

The two main 3D libraries are ThreeJS and BabylonJS. Both were studied by the author of this thesis and then ThreeJS was selected. The main reason BabylonJS was not selected was that it does not support 3D models exported in FBX file format. On the other hand, ThreeJS supported all the formats. It has a big development community so getting help was easy. The main reason ThreeJS was selected was that it has an open-source 3D editor that provides all the functionalities that were needed to build this application. The open-source 3D editor was bootstrapped and then the application was further developed on it.

3.3.2 Architecture

The 3D model editor application was bootstrapped from the ThreeJS editor so it follows its architecture and file structure. It uses vanilla javascript. The UI components were created using javascript and then they were appended to the DOM of the application. There was a learning curve while adapting to this approach because HTML was not used to create UI components.

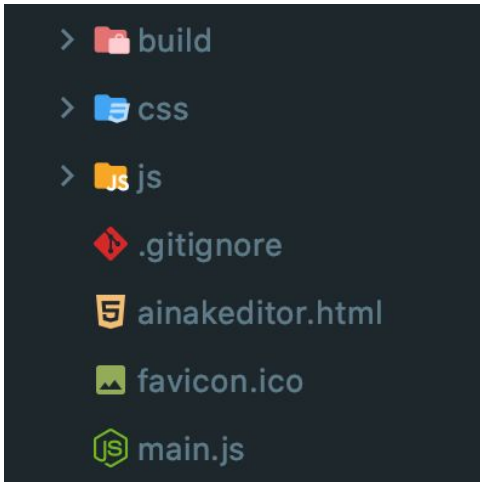


Figure 14: File structure of 3D Model Editor application

Figure 14 shows how the files were structured in this application. A brief description of all the files and folders is listed below:

- **build:** This folder contains the pre-build ThreeJS library both in production-ready (.min) and regular development-ready format.
- **css:** This folder consists of all the stylesheets.
- **js:** This folder contains all the 3D file loaders, lights, and all other components that are needed to build a 3D application for the web.
- **ainakeditor.html:** This file initializes the ThreeJS library and as well as all of the other components used from the JS folder.
- **main.js:** This file is responsible for creating and managing the 3D window in the editor. When the *ainakeditor.html* is loaded this file creates a 3D window and when the HTML file is closed it closes the 3D window and manages the garbage collection.

3.3.3 Application

This section of the thesis discusses the inner workings of the application in detail.

Application Entry point:

The entry point for the application is *ainakeditor.html.* This file loads the ThreeJS library and also loads all of the other components needed for the application, for example, 3D model loaders like GLTF, FBX, OBJ loaders.

UI and UX:

The UI of the applications strongly resembles the Web portal application UI. This way users can still have the same look and feel in both of the applications. The color scheme and font styles are also the same. Figure 15 shows the main page of the application after a 3D model is loaded.

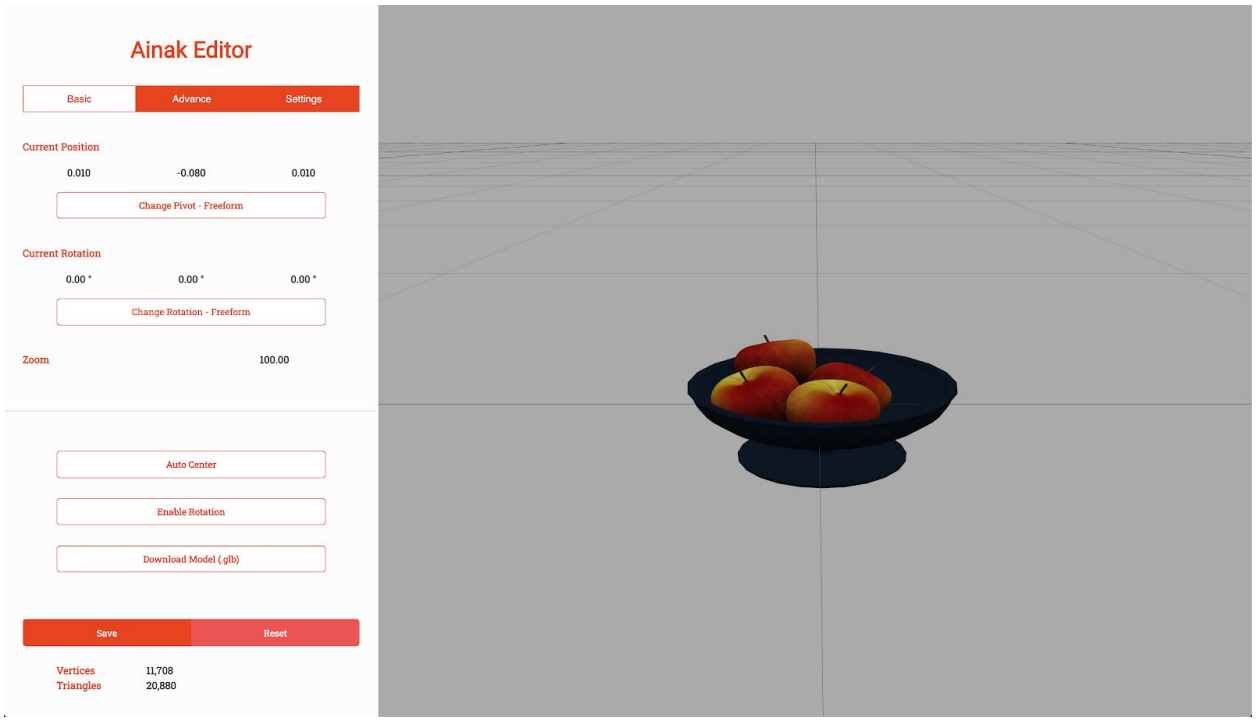


Figure 15: 3D Model Editor application home page

The UI consists of two panels where one on the left hosts all of the controls and the right one hosts the 3D model.

From the controls panel, the user can change the position of the pivot and rotation of the 3D model. This panel also provides the option to the user to download the 3D model in GLB, binary GLTF, format. Users can also enable rotation where the 3D model rotates around its pivot point on Y-axis (up axis). From the advanced tab, the user can see the hierarchy of the model's different layers and from the settings tab, the user can see different shortcuts available to use in the application. The user can press the save button when the editing is done or click the reset button to start over. When the user saves the 3D model the new metadata, which includes new rotation, pivot point, and the 3D model file, is sent to the backend application and stored. Next time the user opens the same model, the user can see the edits they had made earlier.

Proper user feedback is given to the user when the user interacts with the application, for example, when the model loads it shows the loading bar on the top of the page and if the model does not load it shows an error message to the user. Figure 16 shows how an error message is shown while loading a model fails.

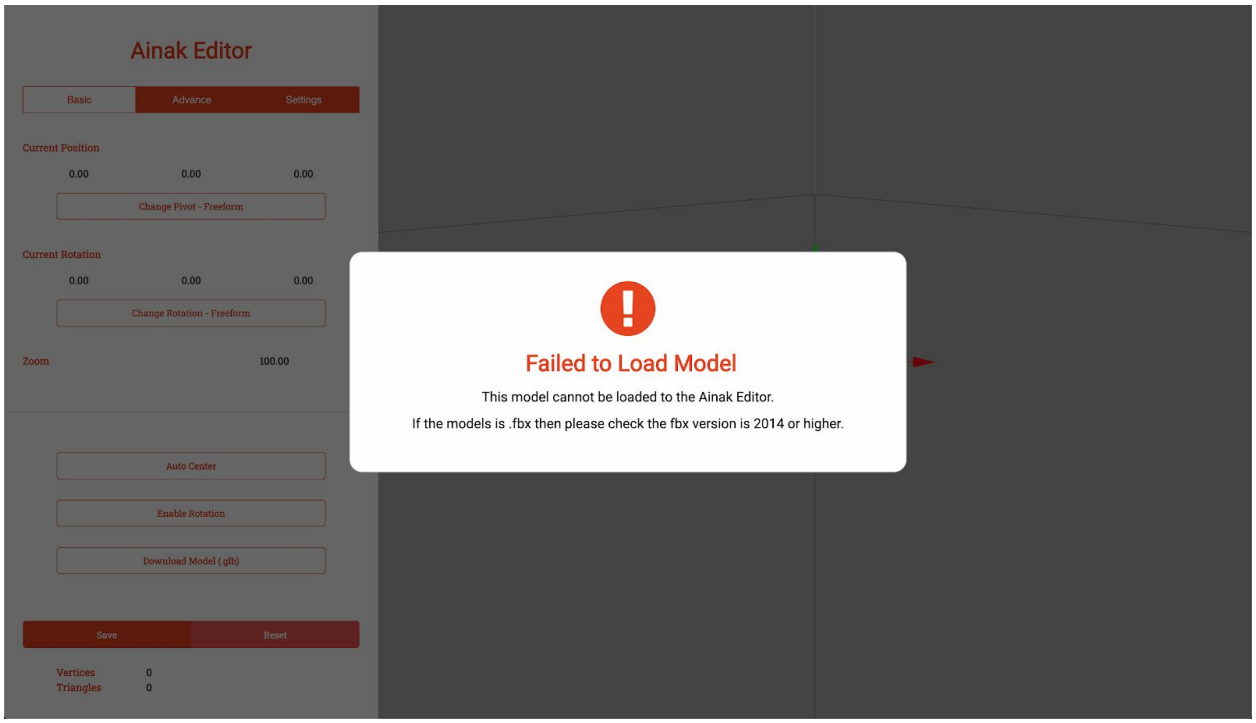


Figure 16: Model Failed to Load

3.3.4 Testing and Planning

Testing for this application was done manually where the developer, the author of this thesis, and product owner sit together and test the application on different browsers. If there are any changes to be made were documented and then worked upon. Mostly, testing was done on the basis of acceptance testing requirements set while planning the sprints.

The planning was done on sprint bases where a feature was selected to be developed in the sprint. All the requirements were defined and then if there were any questions they were asked to the product owner. Finally, during the planning phase, a list of acceptance testing requirements is made which was used to evaluate if the sprint was successful or not.

3.4 Backend Application

This section of the thesis discusses the backend application in detail. The backend application is a Restful NodeJS application. This application exposes the APIs that are used in frontend applications that are discussed in sections 3.2 and 3.3. Also, these exposed APIs are used in the Mobile application as well. MySQL was used as a database.

3.4.1 Technology Stack

There are many frameworks to choose from when creating a restful backend application. Some of the most popular are NodeJS, PHP (laravel), and Django. A study on all of these frameworks was done and in the end, NodeJS was chosen.

All of these three frameworks provide the basic features that are needed to build this application so all of them were compared on the basis of security, scalability, and team skills.

- **Security:** Django is a web application framework that provides inbuilt authentication which makes it secure. It helps the developers to avoid common mistakes and make the security easy to implement and avoid common pitfalls. Laravel is also a web framework that provides security out of the box. Developers however can configure or change it according to their needs. NodeJS does not provide any security out of the box. It lets the developers create their own authentication systems. So from a security point of view both, Django and laravel are good choices.
- **Scalability:** All of these frameworks are built to be scalable and for this application, they provide enough features to scale in the future. So from a scalability point of view, any of them is a good choice.
- **Team Skills:** The main factor in making the decision to choose the framework for this application was team skills and prior experiences. As the team is more efficient and comfortable in javascript so NodeJS was heavily supported. For security concerns, there are many open source libraries that can provide state of the art authentication and security. Also for this application Express JS library was used to expose the APIs. Express app provides many functionalities including middlewares. Using these middlewares all the APIs can be secured. So, for future development and less learning curve, NodeJS was chosen to be the framework for this application.

Apart from the core web framework, 3D model conversion libraries were used to automate the conversion of different 3D file formats to GLTF file format. There are a limited number of such libraries available that can work with the NodeJS application out of the box. Most of the conversion tools are command-line based for example Assimp [9] and FreeCAD [10], that is they are needed to be executed in the operating system terminal. NodeJS application provides the ability to execute commands in the terminal but it also introduces security issues if those commands are compromised. So those command-line tools were avoided. To convert the FBX file format to GLTF an open-source library *fbx2gltf* [11] was used and similarity to convert OBJ file format to GLTF *obj2gltf* [12] library was used. After the successful conversion of the model, it is validated by using *gltf-validator* [13] library that is built by the authors of GLTF file format.

A brief discussion is made about the command line tools like Assimp and FreeCAD in chapter 5 of this thesis.

For the database, MySQL was used and it provided all the functionality that was needed for this application and the team was most familiar with this database.

3.4.2 Architecture

For this application Express JS web framework was used. Express JS is a minimal and flexible NodeJS web application framework that provides robust and unopinionated features for the web and mobile applications [14]. Using this framework bootstrapping the application was easy and hassle-free. As this framework is unopinionated, so the structure of the application mostly depends on the development.

To develop the database for this application an ORM technique was used. ORM, object-relational mapping, is a technique to convert data to and from incompatible type systems. The ORM used for this application is Sequelize [15]. The main advantage to use an ORM is that it can abstract the database layer. ORM can be used to define the database structure that includes creating the database, tables, and all the relations between the tables using the NodeJS application. Also, ORM can be used if the database engine needs to be changed for example from MySQL to Postgres without creating tables and relations. Section 3.4.3 discusses the use of Sequelize in detail.

For this application, the structure looks like as shown in figure 17.

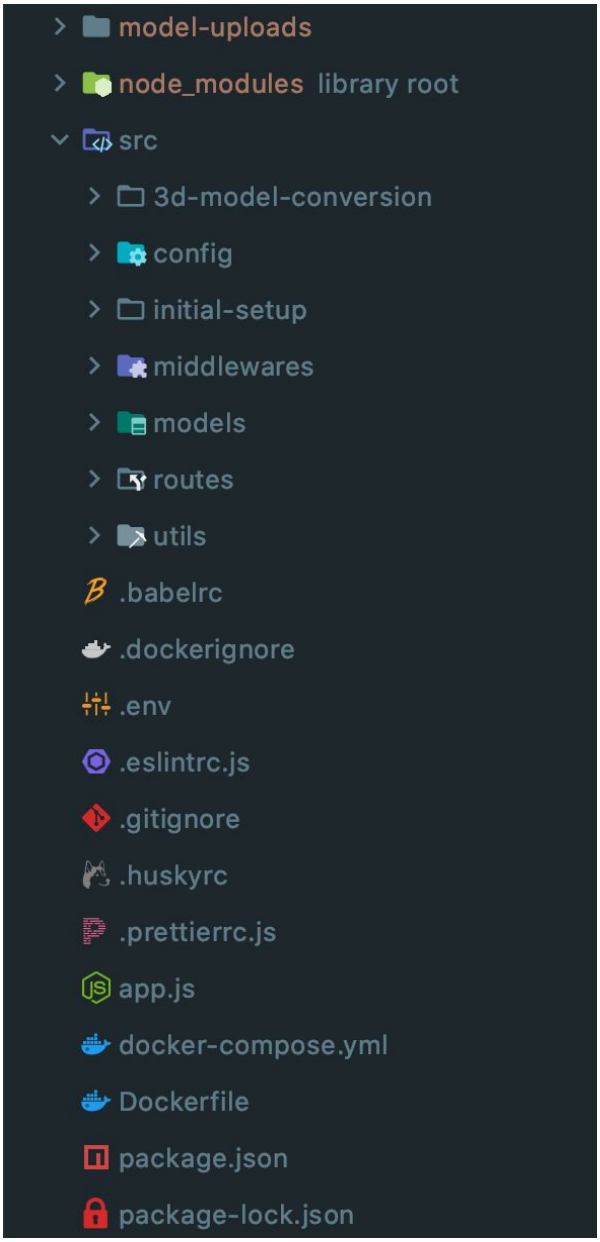


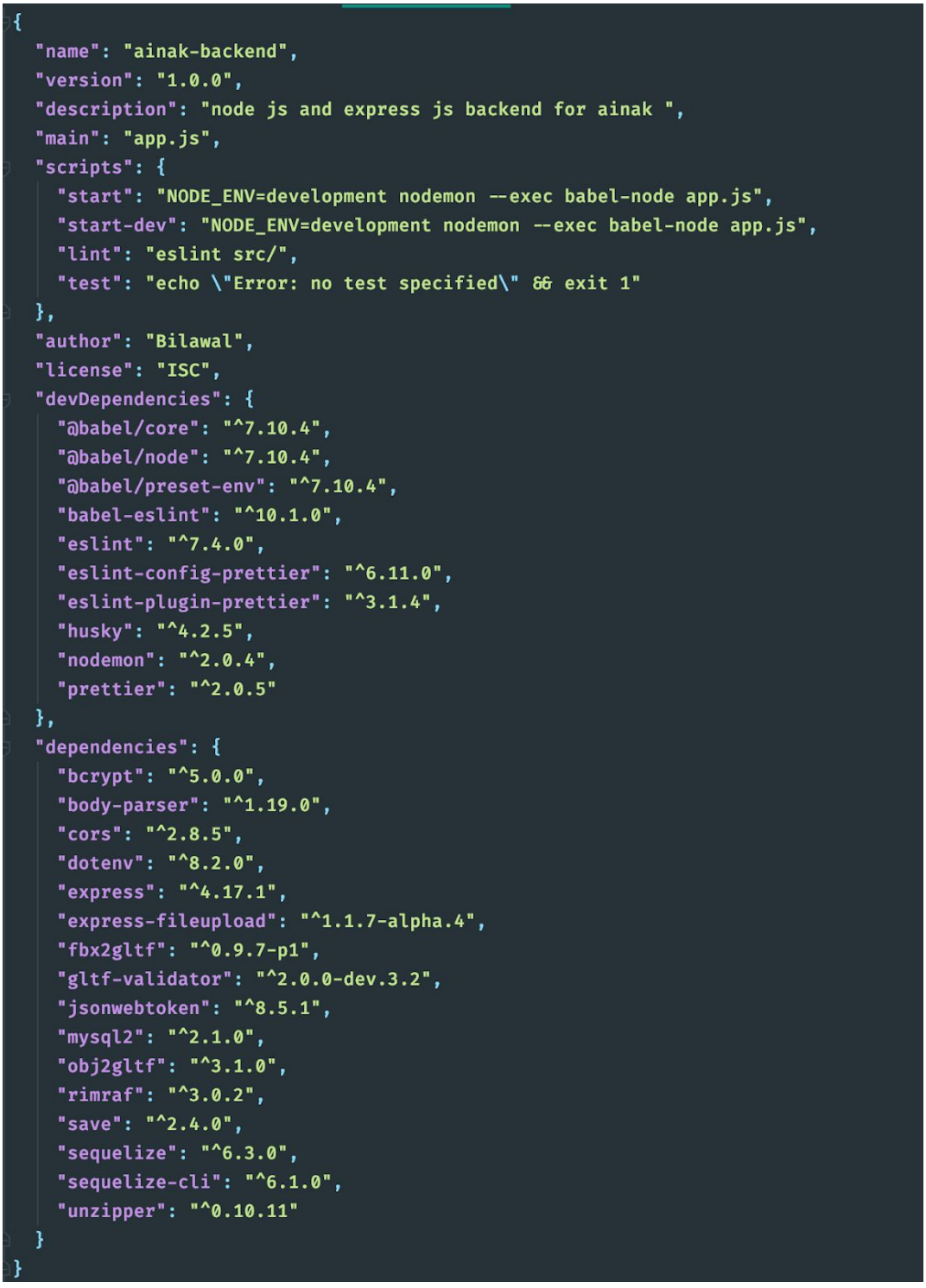
Figure 17: File structure of the backend application

The files and folders from figure 17 are described below:

- **model-uploads:** This folder contains all the models that are uploaded. The models will be saved to this folder after they are registered in the database.
- **node_module:** This folder contains all the node js libraries used in the project. This folder is generated when the libraries are installed in the project.

- **src:** This folder contains the main logic of the application. The subfolder structure is defined as below:
    - **3d-model-conversion**: This folder contains the logic of converting and validating 3D models.
    - **config**: This folder contains the configuration file of the Sequelize ORM.
    - **initial-setup**: This folder contains the logic to set up the initial values in the database. For example, setting up initial user roles.
    - **middlewares**: This folder contains all the middlewares. Middlewares includes the tokenization and authorization of routes.
    - **models**: This folder contains all the definitions of database tables.
    - **routes**: This folder contains all the routes that are provided in the application.
    - **utils**: This folder contains all the utility functions like error handling and validations.
- **.babelrc**: This file helps in compiling ES6 code.
- **.dockerignore**: This file tells docker which files and folders to ignore.
- **.env:** This file stores the environment variables like API keys and secrets.
- **.eslint.js:** This file is used to control the code quality. This provides rules that should be followed in order to get a consistent code. For example, use single-quotes everywhere.
- **.gitignore:** used to tell the version control system which files should be ignored.
- **.huskeyrc:** This file is used to control the quality of the code before the code is pushed to the version control system. In this case, eslint is run before the code is committed to the GitLab.
- **.pritterrc.js:** This configuration file is used to auto-format the code to achieve a consistent coding style.
- **app.js:** This file is the entry point of the application. This file is responsible to create and launch the application.
- **docker-compose.yml and DockerFile**: These files are used to containerize the application for development and deployment purposes.
- **package.json and package-lock.jason:** These files are used to manage the dependencies used in the project.

In order to build this application, many other libraries were used. Figure 18 shows the package.json file of this project showing all the dependencies used in this application.



```
{
  "name": "ainak-backend",
  "version": "1.0.0",
  "description": "node js and express js backend for ainak ",
  "main": "app.js",
  "scripts": {
    "start": "NODE_ENV=development nodemon --exec babel-node app.js",
    "start-dev": "NODE_ENV=development nodemon --exec babel-node app.js",
    "lint": "eslint src/",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Bilawal",
  "license": "ISC",
  "devDependencies": {
    "@babel/core": "^7.10.4",
    "@babel/node": "^7.10.4",
    "@babel/preset-env": "^7.10.4",
    "babel-eslint": "^10.1.0",
    "eslint": "^7.4.0",
    "eslint-config-prettier": "^6.11.0",
    "eslint-plugin-prettier": "^3.1.4",
    "husky": "^4.2.5",
    "nodemon": "^2.0.4",
    "prettier": "^2.0.5"
  },
  "dependencies": {
    "bcrypt": "^5.0.0",
    "body-parser": "^1.19.0",
    "cors": "^2.8.5",
    "dotenv": "^8.2.0",
    "express": "^4.17.1",
    "express-fileupload": "^1.1.7-alpha.4",
    "fbx2gltf": "^0.9.7-p1",
    "gltf-validator": "^2.0.0-dev.3.2",
    "jsonwebtoken": "^8.5.1",
    "mysql2": "^2.1.0",
    "obj2gltf": "^3.1.0",
    "rimraf": "^3.0.2",
    "save": "^2.4.0",
    "sequelize": "^6.3.0",
    "sequelize-cli": "^6.1.0",
    "unzipper": "^0.10.11"
  }
}
```

Figure 18: package.json of the backend application

As we can see there are two sections for dependencies. The devDependencies are not discussed here as they are used for code quality and streamline the development

process and they do not affect the outcome of the application. The dependencies are discussed in detail as below:

Bcrypt:
This library is used to hash the passwords and then stored in the database. This library is also used to compare hash passwords when the user logs in with a password.

Body-parser:
This library is responsible to parse the data that is coming as the part of the request body. This library cannot parse multipart data.

Cors:
This library is used to allow requests to APIs from different origins. As APIs will be hosted on a different server than the UI.

Dotenv:
This library is used to enable *environment variables* in the project. Environment variables are used to store API secrets or token private keys.

Express:
This is the main framework used to build this application. This is a web framework that is used to expose routes for APIs.

Express-fileupload:
This library is used to parse multipart requests. It will provide the file from the request body and then be used to move that file to a new location on the server.

Fbx2gltf:
This library is used to convert FBX file format to GLTF file Format.

Gltf-validator:
This library is used to validate the converted GLTF models.

Jsonwebtoken:
This library is used to generate and validate JSON web tokens.

Mysql2:
This is a MySQL client for node applications. This is used to establish a connection to the MySQL server installed on the hosting server.

Obj2gltf:
This library is used to convert the OBJ file format to GLTF file format.

Rimraf:
This library provides the functionality of removing files and folders from the hosting server.

Save:
This library is used by Sequelize ORM to save objects to the datastore.

Sequelize and Sequelize-cli:
These libraries provide ORM for MySQL. These are used to create, connect, and Query MySQL databases.

UnZipper:
This library is used to unzip files on the hosting server.

A detailed discussion of how the above libraries are used in this application is discussed in section 3.4.4.

3.4.3 Database

For this application, MySQL is used as a database engine. The database and its tables are not created manually. For this purpose, an ORM, Sequelize, is used. The benefit of using an ORM is that it allows the flexibility to change the database engine in the future without developing the whole schema for it. For example, if MySQL database is to be replaced by Postgress then only the configuration file of the ORM needs to be changed and the rest will be handled by the ORM. The downside of using an ORM is that it introduces a learning curve if the developer has never used an ORM before.

For this project, Sequelize ORM is used and its configuration is stored in *src/config/config.json* file. Figure 19 shows a sample configuration file.

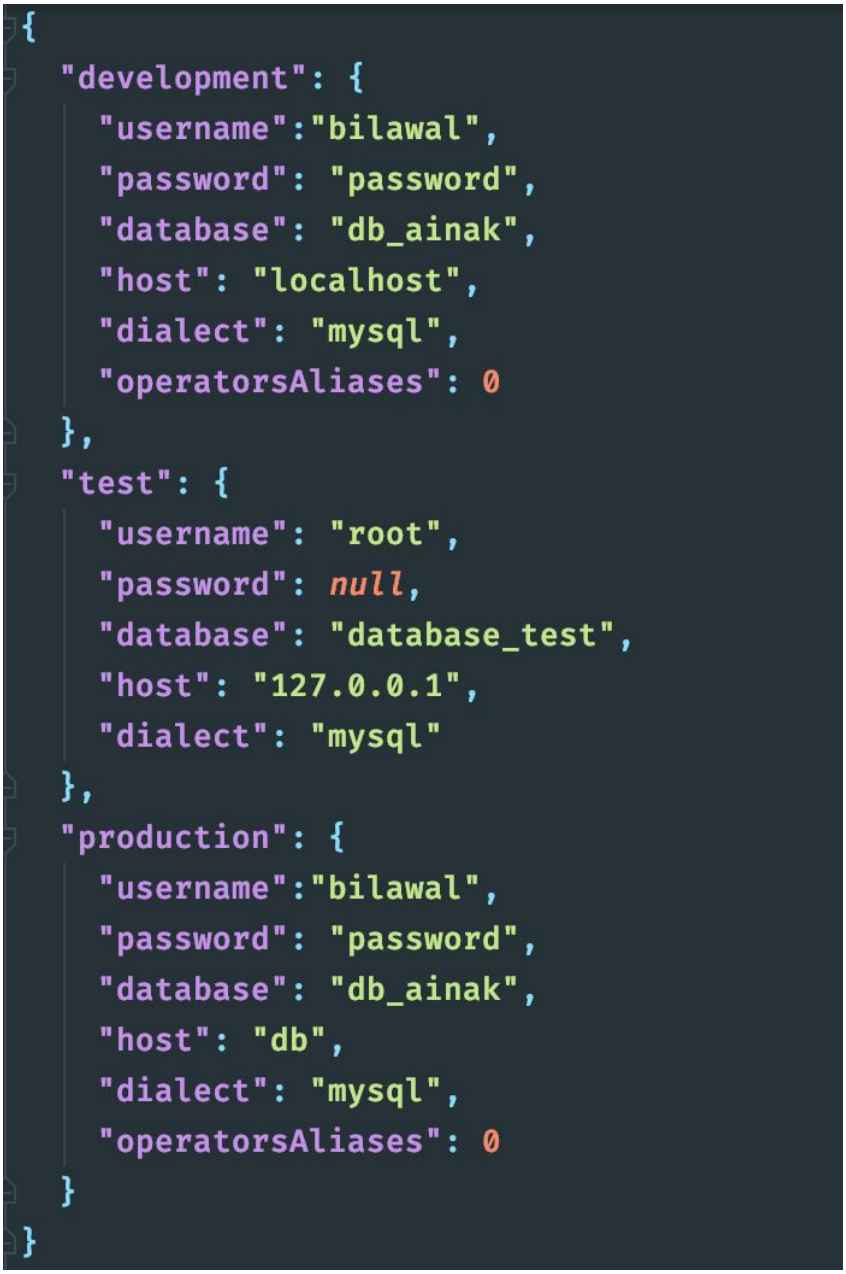


Figure 19: Sequelize Configuration file

A configuration file usually contains the username and password to access the database from the database engine, a database name, URL of the host, and dialect of the database. A dialect can be MySQL, SQLite, etc depending on which database engine is used.

All the tables and their relations are usually stored in the models folder. A typical table definition is shown in figure 20.

```
module.exports = (sequelize, DataTypes) => {
    const User = sequelize.define(
        'User',
        {
            id: {
                type: DataTypes.INTEGER,
                allowNull: false,
                primaryKey: true,
                autoIncrement: true
            },
            role_id: {
                type: DataTypes.INTEGER,
                allowNull: false
            },
            name: {
                type: DataTypes.STRING,
                allowNull: false
            },
            email: {type: DataTypes.STRING ... },
            password: {type: DataTypes.STRING ... },
            active: {type: DataTypes.BOOLEAN ... },
            created_at: {type: DataTypes.BIGINT ... },
            updated_at: {type: DataTypes.BIGINT ... }
        },
        { ... }
    )
    User.associate = (models : Models ) => {
        User.belongsTo(models.Role, options: {foreignKey: 'role_id'})
        User.hasOne(models.Profile, options: {foreignKey: 'user_id'})
    }
    return User
}
```

Figure 20: Table definition in Sequelize ORM

First, all of the columns of the table are defined and then the relations or associations are defined. Associations can be OneToOne, OneToMany, or ManyToMany.

3.4.4 Application

This section of the thesis discusses in detail the inner working of the backend applications.

Application Entry Point:

As stated in section 3.4.2 the entry point of the application is *app.js.* This file is shown in figure 21.

```
import 'dotenv/config'
import express from 'express'
import cors from 'cors'
import routes from './src/routes'
import bodyParser from 'body-parser'
import db from './src/models'
import middleware from './src/middlewares'
import {init} from './src/initial-setup'
import fileUpload from 'express-fileupload'

const app = express()

//middleware
app.use(cors())
app.use(bodyParser.urlencoded({extended: false}))
app.use(bodyParser.json())
app.use(
    fileUpload( options: {
        createParentPath: true
    })
)
//routes with middleware
const api = '/v1/api'
app.use( fn: api + '/login', routes.login)
app.use( fn: api + '/register', routes.registration)
app.use( fn: api + '/reset-password', routes.restPassword)
app.use( fn: api + '/forgot-password', routes.forgotPassword)
app.use( fn: api + '/user-role', middleware.authenticateToken, routes.userRole)
app.use( fn: api + '/child-account', middleware.authenticateToken, routes.childAccount)
app.use( fn: api + '/profile', middleware.authenticateToken, routes.userProfile)
app.use( fn: api + '/model', middleware.authenticateToken, routes.userModel)
app.use( fn: api + '/home', middleware.authenticateToken, routes.home)
app.use( fn: api + '/meta-data', middleware.authenticateToken, routes.metadata)

app.listen(process.env.PORT, () => {
    db.sequelize
        .sync({force: false})
        .then(() => init().then(() => console.log(`🚀 Ainak •• api listening to  port ${process.env.PORT}`)))
})
```

Figure 21: Structure of app.js

App.js file is responsible for loading and initializing all the libraries used in the application. So all of the required libraries are imported. Then middlewares are defined. After that, all the available routes are defined. If routes are looked closely there are two types of routes there, one that is not authenticated by a middleware these are open routes that mean users of this application do not need an active account to access them, the other, for example,*/profile* uses middleware for authentication that means the user of this application needs an active account with a valid token to access those APIs. The authenticate Token middleware is a custom middleware that is shown in figure 22. This middleware takes the token from the request body and then tries to validate it. If the validation is successful then it will append the user's *id, active status,* and *role* to the request body that will be used later on in the process.

```
import jwt from 'jsonwebtoken'
import handleError from '../utils/error-handler'
import db from '../models'

export const checkToken = (req, res, next) => {
    //Get the token from the header
    let token = req.headers['x-access-token'] || req.headers['authorization'] || ''

    if (token) {
        // Remove Bearer from the the token if its there
        if (token.startsWith('Bearer ')) {
            // Remove Bearer from string
            token = token.slice(7, token.length)
        }

        //Verify token
        jwt.verify(token, process.env.SECERT_KEY,  options: async (err, decoded) => {
            if (err) {
                return res.status(403).send(handleError( errorCode: 100,  errorFields: [],  errorMessageForUser: 'Invalid Token'))
            } else {
                const user = await db.User.findOne({
                    attributes: ['role_id', 'active'],
                    where: {
                        id: decoded.id
                    }
                })

                if (!user) {
                    return res.status(403).send(handleError( errorCode: 100,  errorFields: [],  errorMessageForUser: 'Invalid Token'))
                }

                req.authenticated_user_id = decoded.id
                req.authenticated_user_role_id = user.get( key: 'role_id')
                req.authenticated_user_active = user.get( key: 'active')
                next()
            }
        })
    } else {
        return res.status(400).send(handleError( errorCode: 100,  errorFields: [],  errorMessageForUser: 'Invalid Token'))
    }
}

export default {checkToken}
```

Figure 21: Structure of Authenticate Token middleware

3D file format Conversion:

When the application is up and running on the server then a registered user can upload a 3D model file that will be stored in the database and the actual file will be moved to a folder on the hosting server. The file names are randomized before storing the 3D models on the server but when a user requests a model then it is given to them with the correct file name.

The conversion process is asynchronous which means it will not block the user until the conversion is done. When the user uploads the model a notification is given to them that model is uploaded and then the conversion process is started. The model upload and conversion process work as described below.

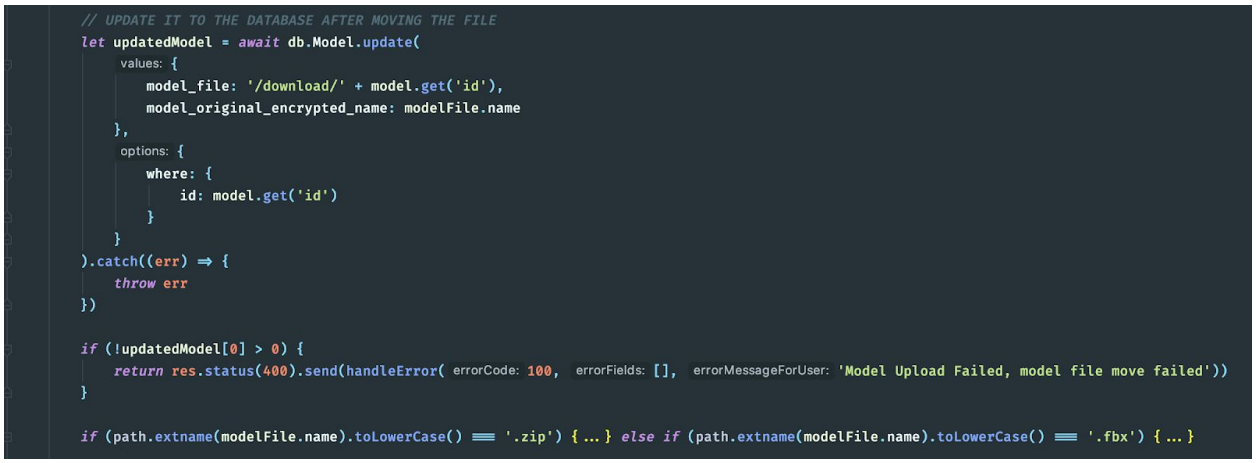
First, the model file is validated if there is a file in the request body. If there is a file then the data related to the file is entered in the database. Figure 22 shows this process.

```
//Use the name of the input field to retrieve the uploaded file
let modelFile = req.files.model_file

if (!name) {
    name = modelFile.name
}

//Save Model to the database
let model = await db.Model.create( values: {
    user_id: userID,
    model_file: '',
    model_extension: path.extname(modelFile.name),
    name: name,
    file_name: modelFile.name,
    description: description,
    app_access: false,
    model_original_encrypted_name: '',
    deleted: false,
    created_at: Math.floor( x: Date.now() / 1000),
    updated_at: Math.floor( x: Date.now() / 1000)
}).catch((err) => {
    throw err
})

if (!model) {
    return res.status(400).send(handleError( errorCode: 100,  errorFields: [],  errorMessageForUser: 'Model Upload Failed'))
}
```

Figure 22: Saving file data in the database

After the file data is successfully saved in the database then the actual file name is randomized and then it is moved to the model-uploads folder. After the successful move of the file, the database entry is updated to add the model file location and its new randomized name. After that, the extension of the file is examined, and depending on

the file extension a proper 3D model conversion library is called. This process is shown in figure 23.



```
// UPDATE IT TO THE DATABASE AFTER MOVING THE FILE
let updatedModel = await db.Model.update(
    values: {
        model_file: '/download/' + model.get('id'),
        model_original_encrypted_name: modelFile.name
    },
    options: {
        where: {
            id: model.get('id')
        }
    }
).catch((err) => {
    throw err
})

if (!updatedModel[0] > 0) {
    return res.status(400).send(handleError( errorCode: 100,  errorFields: [],  errorMessageForUser: 'Model Upload Failed, model file move failed'))
}

if (path.extname(modelFile.name).toLowerCase() === '.zip') { ... } else if (path.extname(modelFile.name).toLowerCase() === '.fbx') { ... }
```

Figure 23: Updating file data and choosing proper conversion library

In this application to support the conversion of the OBJ file format with texture and materials, the file should be uploaded in the *.zip* format. When a .zip file is detected then it is first unzipped on the server then the 3D conversion library is used to convert the model to GLTF. This is shown in figure 24. If the file is a .FBX then the fbx conversion library is called. If the file is none of the above extension it is still stored on the server but no conversion is done on it. And finally, a response is sent to the user.

```
if (path.extname(modelFile.name).toLowerCase() === '.zip') {
    fs.createReadStream( path: process.env.MODLE_UPLOAD_LOCAION + modelFile.name).pipe(
        unzipper
            .Extract( opts: {
                path:
                    process.env.MODLE_UPLOAD_LOCAION +
                    path.basename(modelFile.name, path.extname(modelFile.name)) +
                    '/'
            })
            .on('close', function () {
                objToGLTF(
                    src: process.env.MODLE_UPLOAD_LOCAION +
                        path.basename(modelFile.name, path.extname(modelFile.name)),
                    process.env.MODLE_UPLOAD_LOCAION,
                    path.basename(modelFile.name, path.extname(modelFile.name)),
                    ext: '.glb',
                    model.get('id')
                )
            })
    )
} else if (path.extname(modelFile.name).toLowerCase() === '.fbx') { ... }
```

Figure 24: Unzipping an OBJ model and then converting it to GLTF

The model after conversion looks like shown in figure 25 on the file structure of the hosting server.

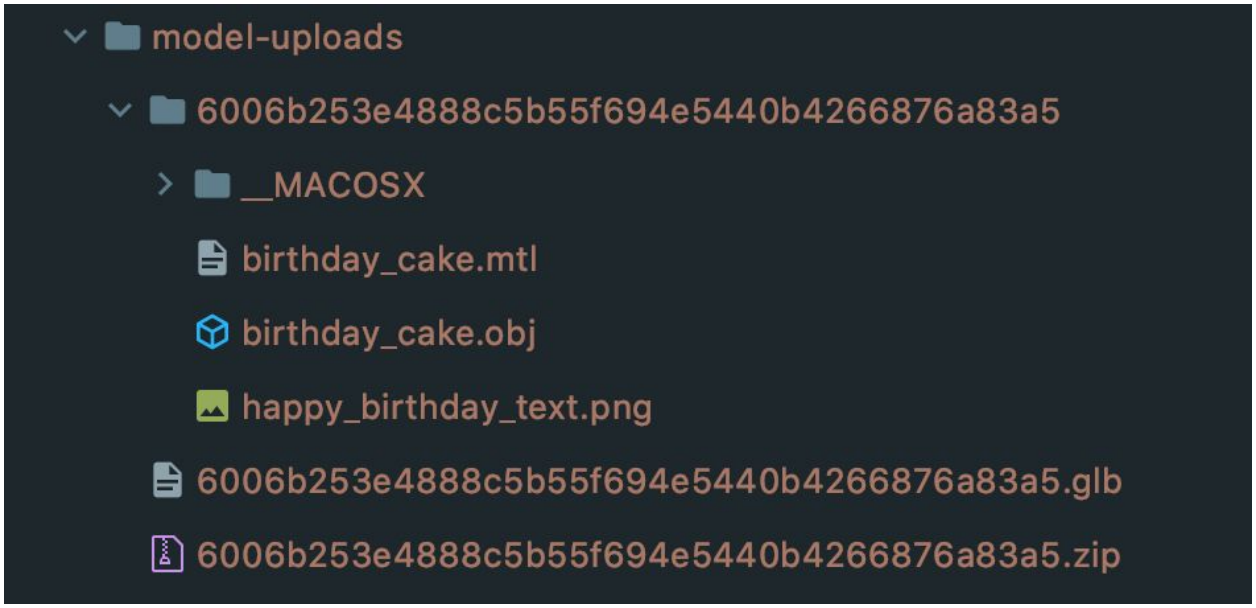


Figure 25: Model uploads folder structure

Here both the .zip file and the newly converted .glb file are stored. Please note that the first folder that includes the .obj model is deleted whether the conversion is successful or not. It is shown here just to clarify the process of unzipping and then converting the model.

In figure 23 there is a function *objToGltf* is used to do the actual conversion. The process of the conversion is described below.

First, the name *.obj* file is determined by looping through all the files in the given folder then the path of that *.obj* file is passed to the *obj2gltf* library. As this library is a promise based so it returns if the conversion was successful or not. If the conversion was successful then the model is validated by using the gltf-validator library and if the newly converted model is valid then the database entry of that particular model is updated and the path to the newly converted model is added. The whole process is shown in figure 26. The same process is followed for the conversion of .fbx file format.

Figure 26: objToGltf conversion process

After a successful conversion, the user can call the API to get the newly converted model or the original model anytime. A sample response is shown in figure 27. Both of the download URLs are available in the response of the API call.


```
{
    "status": true,
    "message": "User 3D model fetched successfully",
    "model": {
        "id": 2,
        "user_id": 1,
        "model_file": "/download/2",
        "model_extension": ".zip",
        "model_file_modified": "/download/modified/2",
        "model_extension_modified": ".glb",
        "name": "Excavator",
        "file_name": "Archive.zip",
        "description": "A good model :) ",
        "app_access": false,
        "deleted": false,
        "created_at": 1595433286,
        "updated_at": 1595433286
    }
}
```


Figure 27: Sample response to get uploaded model

Appendix A contains all the APIs available in the backend application.

3.4.5 Testing and Planning

The testing of the backend application was done manually. Postman was used for testing all the endpoints. The developer, author of this thesis, and the product owner collectively tested the APIs and these tests were done on each sprint review. If the APIs pass the acceptance testing requirements then the new APIs were chosen for the next sprint else the list of bugs or unexpected behavior was reported to the developer to fix.

After every successful acceptance testing, the new APIs were selected to be developed and the requirements and acceptance testing requirements were given to the developer. If the developer had any questions they were communicated to the product owner and a resolution was made.

4. Deployment and Testing

To deploy and test the UI before making it live to the public a staging server was used. The staging server was a firebase server that hosted the UI and 3D model editor projects. To test the backend server with the UI and 3D editor a Linux based server was used. The backend application was containerized using Docker. The containerization process is discussed in section 4.1.

4.1 Containerization

A containerization is a form of operating system virtualization that lets the applications run in a totally isolated environment in user spaces. These spaces are called containers. Containers are similar to virtual machines but they do not have their own operating system. They use the host machine operating system that makes them much smaller in size as compared to the virtual machines.

Docker is one of the most famous containerization tools that let the developers create their own container or use open source containers. Docker solves the most basic problem of development that is if the application is running on the developer's computer it will run everywhere. Docker is not just used in the deployment process but also in the development process. It lets the developers have the same environment regardless of the operating system or file structure.

For this project, only the backend application was containerized using docker. The DockerFile is shown in figure 28.

```
FROM node:12

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

# install the dependencies
RUN npm install


# If you are building your code for production
RUN npm ci --only=production

# Bundle app source
COPY . .

EXPOSE 3000
CMD [ "npm", "start" ]
```

Figure 28: DockerFile of the backend application

As the backend application depends on the MySQL server so a docker compose file was created. Docker compose is a tool to define and run multiple container docker applications. The docker compose file is shown in figure 29. For the MySQL image, a volume was specified as the data should be persisted even if the docker reloads the container or it crashes. Same for the API service, there are commented volumes as they were needed while developing and provided live reload while developing and testing the APIs.

```yaml
version: '3.3'
services:
  db:
    image: mysql
    container_name: ainak-db
    restart: always
    environment:
      MYSQL_DATABASE: 'db_ainak'
      MYSQL_USER: 'bilawal'
      MYSQL_PASSWORD: 'password'
      MYSQL_ROOT_PASSWORD: 'password'
    ports:
      - '4000:3306'
    expose:
      - '4000'
      # Where our data will be persisted
    volumes:
      - ./db:/var/lib/mysql
  api:
    build: .
    container_name: ainak-api
    restart: on-failure
    ports:
      - '3000:3000'
    expose:
      - '3000'
#    volumes:
#      - .:/usr/src/app/
#      - /usr/src/app/node_modules/
    depends_on:
      - db
    links:
      - db
```

Figure 29: docker-compose.yml of the backend application

After installing the docker on the Linux hosting server *docker-compose up -d* command was run from the root folder of the project to make the system live.

The other two applications, UI and 3D model editor were hosted without containerization on normal Linux based servers.

4.2 Testing

When all three applications were live then a whole system-wide test was done by the full team. A new user was created, then different models were uploaded, and then those models were tested in the Ainak AR application, the company's own AR application. After a few weeks of testing then the system was made live for the customers of the company.

5. Conclusion and Future Work

At the end of the thesis, a complete working software system was delivered. That can lead the user from creating their own account to upload 3D models and then edit them in the browser. After that making them available in the AR application. However, in its current state, the system can only handle two different 3D file formats, OBJ and FBX, conversion to GLTF that is a huge step toward the right direction. Now with the results of this thesis, the converted GLTF files can be used in Ainak Oy's other projects that only support GLTF files. Before that the whole process was manual, that is the FBX models were converted to GLTF manually.

Even though the result of this thesis was a successful project, a lot of improvements can be made. For example, other 3D file formats can be supported, among them STEP, DAE, and IFC are important ones. To support these formats Softwares like Assimp and FreeCAD can be studied and implemented as they both provide command-line tools to automate the 3D file format conversion. Also, 3D mesh compression can also be implemented to increase the performance of the AR applications as the devices running these applications usually do not have enough power to render a high poly 3D model.

Other than that the testing was done manually in the whole project that can be automated meaning creating a workflow where the user has to write tests for every functionality by introducing TDD (Test Driven Development) and BDD (Behaviour Driven Development).

In the end, it will be necessary to conduct load testing when multiple users upload different 3D models with big file sizes. The application should be responsive and have a delay of fewer than two seconds even hundreds of user upload models at the same time.

6. References

[1] Azuma, R. T. et al. (1997). A survey of augmented reality. Presence, 6(4):355–385.

[2] “Ainak Oy” [Online]. Available: http://ainak.io/ [Accessed: 15-Jul-2020].

[3] “Most used 3D model formats” [Online]. Available: https://all3dp.com/3d-file-format-3d-files-3d-printer-3d-cad-vrml-stl-obj/#Top [Accessed: 28-Jul-2020].

[4] “JPEG of 3D” [Online]. Available: https://www.khronos.org/gltf/ [Accessed: 28-Jul-2020].

[5] “MS Hololens” [Online]. Available: https://www.microsoft.com/en-us/hololens [Accessed: 28-Jul-2020].

[6] “Fbx owned by Autodesk” [Online]. Available: https://en.wikipedia.org/wiki/FBX [Accessed: 28-Jul-2020].

[7] “GLTF built by Khronos Group ” [Online]. Available:https://en.wikipedia.org/wiki/GlTF [Accessed: 28-Jul-2020].

[8] “IKEA AR application” [Online]. Available:https://apps.apple.com/us/app/ikea-place/id1279244498 and https://play.google.com/store/apps/details?id=com.inter_ikea.place&hl=en [Accessed: 28-Jul-2020].

[9] “Assimp” [Online]. Available: https://www.assimp.org/ [Accessed: 28-Jul-2020].

[10] “FreeCAD” [Online]. Available: https://wiki.freecadweb.org/Getting_started [Accessed: 28-Jul-2020].

[11] “fbx2gltf” [Online]. Available: https://www.npmjs.com/package/fbx2gltf [Accessed: 28-Jul-2020].

[12] “obj2gltf” [Online]. Available: https://www.npmjs.com/package/obj2gltf [Accessed: 28-Jul-2020].

[13] “gltf-validator” [Online]. Available: https://www.npmjs.com/package/gltf-validator [Accessed: 28-Jul-2020]

[14] “Express JS” [Online]. Available: https://expressjs.com/ [Accessed:

28-Jul-2020]

[15] “Sequelize” [Online]. Available: https://sequelize.org/ [Accessed: 28-Jul-2020]

[16] “Firebase” [Online]. Available: https://firebase.google.com/ [Accessed: 28-Jul-2020]

Appendix A

APIs list:

Method	URI Template	Relation	Current State	New State	Comments
POST	/login	login			Login the user
POST	/register	register			Registers a new user on the platform
POST	/forgot-password	forgot password			Request for change password
PUT	/reset-password	reset password			Resets the user password
POST	/profile	profile			Creates user profile
PUT	/profile	update profile			Updates user profile
PUT	/user-role	update user role			Updates user role
POST	/child-account	child account			Create a new child account
GET	/child-account	get all			Get all child accounts of the user
GET	/child-account/:id	get			Get child account

					by ID
PUT	/max-child	max child			Set max child account limit for the user
POST	/model	3d model			Upload a 3D model
GET	/model	get all			Get all uploaded 3D models of the user
GET	/model/:id	get			Get a 3D model by ID
DELETE	/model/:id	delete	active	deleted	archive 3D model by id
GET	/download/:id	get			Download model by ID
GET	/download/modified/:id	get			Download a converted 3D model by id
POST	/meta-data/:id	meta data			Create metadata of the 3D model
PUT	/meta-data/:id	meta data			Create metadata of the 3D model
DELETE	/meta-data/:id	meta data	active	deleted	Archive metadata of the 3D model
GET	/meta-data/:id	get			Get metadata of the 3D model
PUT	/available-on-app	available on app			Toggle the model accessibility in the AR Mobile application

License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Bilawal Hussain**

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,

Software Platform for 3D Model Conversion and Management

supervised by **Prof. Gholamreza Anbarjafari, Dr. Cagri Ozcinar, Timo Pastila**

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.

Bilawal Hussain
04/08/2020