МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА КОМПОНЕНТА VUEJS ДЛЯ ВІЗУАЛІЗАЦІЇ ГЕОМЕТРИЧНИХ МОДЕЛЕЙ»**

Виконав: студент 2 курсу, групи 8.1218
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
Р.О. Онопрієнко
(ініціали та прізвище)

Керівник зав. кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної математики, доцент, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент
_____ Лісняк А.О.
(підпис)

« 29 » травня 2019 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Онопрієнку Руслану Олександровичу
(прізвище, ім'я та по-батькові)

1. Тема роботи (проекту) Розробка компонента VueJs для візуалізації геометричних моделей

керівник роботи (проекту) Лісняк Андрій Олександрович, к.ф.-м.н, доцент.
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 29 » травня 2019 року № 811-с

2. Строк подання студентом роботи 27.12.2019

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Реалізація продукту.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 29.05.2019

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	27.05.2019-09.07.2019	
2.	Збір вихідних даних.	09.07.2019-15.09.2019	
3.	Обробка методичних та теоретичних	15.09.2019-03.10.2019	
	джерел.		
4.	Розробка першого та другого розділу.	03.10.2019-05.11.2019	
5.	Розробка третього розділу.	05.11.2019-01.12.2019	
6.	Оформлення та нормоконтроль	01.12.2019-27.12.2019	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	15.01.2020	

Студент _____ Р.О. Онопрієнко
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.О. Лісняк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.В. Кудін
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка компонента VueJs для візуалізації геометричних моделей»: 43 с., 22 рис., 7 джерел, 4 додатка.

3D ГРАФІКА, VUEJS-КОМПОНЕНТ, ВЕБ-КОМПОНЕНТИ, ВІЗУАЛІЗАЦІЯ, ГЕОМЕТРИЧНІ МОДЕЛІ.

Об’єкти дослідження – мова для реалізації веб-компонентів Javascript, веб-компоненти, Vuejs – JavaScript фреймворк, бібліотека Three.js, програмна платформа Node.js, фреймворк Express, інструмент для збірки Webpack.

Мета роботи: створення Vuejs компоненту на основі веб компонентів за для реалізації прикладного середовища з можливістю моделювати геометричні моделей будь-якої складності та створення відповідних пакетів за для їх розповсюдження за допомогою менеджеру пакетів.

Метод дослідження – вивчення документацій вибраних середовищ та їх застосування за допомогою програмування.

У кваліфікаційній роботі розглядається створення VueJs компоненту та веб компонентів застосовуючи мову програмування Javascript та бібліотеку Three.js що використовується за для створення та відображення анімовано-комп'ютерної 3D графіки, а також використання допоміжних засобів таких як система збирання завдань Webpack та програмна платформа Node.js. У ході розробки було створено VueJs компонент з можливістю зчитування вхідних даних з віддаленого сервера за допомогою Uniform Resource Locator(URL) та можливості зчитування вхідних даних безпосередньо від користувача за допомогою вибору файлу користувачем через менеджер файлів та реалізовано спосіб оперування елементами інтерфейсу в інтерфейсах користувача з поміччю Drag-and-drop.

Перевагами Vuejs компоненту та веб-компонентів є простота та легкість при використанні, надійність роботи додатку та можливість його використання на будь-якому пристрої з встановленим браузером будь то персональний комп’ютер чи смартфон.

SUMMARY

Master's Qualification Thesis «Development of a VueJs Component for Geometric Models Visualization»: 43 pages, 22 figures, 7 references, 4 supplements.

3D GRAPHICS, VUEJS-КОМПОНЕНТ, WEB-COMPONENTS, VISUALIZATION, GEOMETRIC MODELS.

The objects of study are the language for the implementation of the web-components – Javascript, web-components, Vuejs – JavaScript framework, Three.js library, Node.js software platform, Express framework, build tool Webpack.

The aim of the study is the creation of Vuejs component based on web components for implementation of the application environment with the ability to simulate geometric models of any complexity and create appropriate packages for their distribution through the package manager.

The methods of research are the study of documentation of selected environments and their application through programming.

The thesis deals with the creation of VueJs component and web components using Javascript programming language and the Three.js library used to create and display animated computer 3D graphics, as well as the use of tools such as Webpack taskbar and Node.js software platform. During the development, a VueJs component was created with the ability to read input from a remote server using the Uniform Resource Locator (URL) and read input directly from the user by selecting a file through the file manager and implementing a method of handling interface elements in the user interfaces with help Drag-and-drop.

The advantages of the Vuejs component and web components are the simplicity and ease of use, the reliability of the application and the ability to use it on any device with a browser installed, whether it is a personal computer or a smartphone.

ЗМІСТ

ВСТУП

Практично кожна сучасна людина стикалася хоча б раз у своєму житті з 3D технологіями. VueJs компонент візуалізації реалізований у ході дипломної роботи може бути корисним, як і звичайним користувачам, які вже мають готові дискретні геометричні моделі так і розробникам які можуть власноруч спроектувати дискретні геометричні моделі шляхом їх опису.

Модель це система, дослідження якої служить засобом для отримання інформації про іншу систему, уявлення деякого реального процесу, пристрої або концепції. Модель є абстрактне уявлення реальності в будь-якій формі як: математичної, фізичної, символічної, графічної – призначене для подання певних аспектів цієї реальності і дозволяє отримати відповіді на досліджувані питання.

В процесі опису 3D моделей можливо створити наступні геометричні моделі:

а) каркасна модель являє форму виробу у вигляді кінцевої безлічі ліній, що лежать на поверхнях виробу. Для кожної лінії відомі координати кінцевих точок;

б) поверхнева модель відображає форму виробу за допомогою завдання обмежують її поверхонь, у вигляді сукупності даних про грані, ребра та вершини;

в) моделі з поверхнями складної форми, так званими скульптурними поверхнями.

Об'ємні моделі відрізняються тим, що в них в явній формі містяться відомості про належність елементів внутрішньому чи зовнішньому по відношенню до виробу простору.

На ринку існує великий спектр програм для візуалізацію геометричних дискретних моделей такі як: AutoCAD, WINGS 3D, DAZ STUDIO, OPEN SCAD, Blender. Загалом усі раніше перечисленні додатки є десктопними. Обмеженість десктопної реалізації додатку у наш час э значним недоліком тому що це по

перше обмеженість платформою реалізації. Саме тому актуальність реалізації веб додатків, а також перенесення реалізованих додатків під десктоп у веб простір та під мобільні пристрої росте з кожним роком.

Уже сьогодні можливо стверджувати що більш актуально реалізувати легковаговий додаток у веб просторі, тому що реалізований таким чином додаток фактично може бути використаний під будь-якою платформою та практично під будь-яким сучасним пристроєм.

Можливість використання 3D у веб просторі відкриває нові межі використання всесвітньої павутини та підносить реалізації інтерактивності у веб на новий рівень.

Виходячи з перерахованих вище обставин було розглянуту веб простір за для реалізації системи візуалізації геометричних моделей за допомогою VueJs компоненту.

VueJs – це JavaScript фреймворк з відкритим вихідним кодом для створення користувацьких інтерфейсів. VueJs компонент був реалізований на основі веб компонентів які у свою чергу написанні за допомогою Javascript – це легковагова, інтерпретована або Just-in-time(JIT) – компільована, об'єктно-орієнтована мова програмування. Найбільш широке застосування знаходить як мова сценаріїв веб-сторінок, але також використовується, і в інших програмних продуктах, наприклад Node.js. JavaScript це прототипна-орієнтована, мультіпарадігменна мова з динамічною типізацією, який підтримує об'єктно-орієнтований, імперативний і декларативний стилі програмування [1].

Але однієї мови JavaScript за для реалізації додатку з використанням 3D технології у просторі веб замало саме тому було розглянуто використання Web Graphics Library.

Web Graphics Library – це програмна бібліотека для мови JavaScript призначена для візуалізації інтерактивної тривимірної графіки і двовимірної графіки в межах сумісності веб-браузера без використання плагінів. WebGL приносить в веб тривимірну графіку, вводячи application programming interface

(API), який побудований на основі OpenGL ES 2.0, що дозволяє його використовувати в елементах canvas HyperText Markup Language(HTML) 5 [7].

Також у якості допоміжного засобу реалізації 3D у веб використовувався Three.js.

Three.js – це легковагова кросбраузерна бібліотека JavaScript, що використовується для створення та відображення анімованої комп'ютерної 3D графіки при розробці веб-додатків. Three.js скрипти можуть використовуватися спільно з елементом HTML5 canvas, SVG або WebGL [6].

Темою для дипломної роботи є «Розробка компонента VueJs для візуалізації геометричних моделей».

Актуальність створюваного VueJs компоненту полягає в наступному:

а) головними платформами розповсюджування розроблюваного компоненту виступає будь-яка платформа з сучасним браузером;

б) можливість візуалізації геометричних 3D моделей у 3D просторі за допомогою бібліотеки Three.js;

в) висока продуктивність яка не залежить від платформи;

г) використання VueJs компоненту в екосистемі VueJs та використання веб-компоненту у будь-якому веб середовищі;

д) безкоштовне розповсюджування та використання за допомогою пакетного менеджера.

1 ОГЛЯД ТЕХНОЛОГІЙ

1.1 Вирішення проблеми

Створення VueJs компоненту для візуалізації геометричних моделей – представляє собою головну проблему вирішення якої необхідно розглянути та виконати у ході виконання дипломної роботи.

Одним із найкращих рішень для поставленого завдання було обрано реалізацію веб компоненту, який буде використовуватися у VueJs компоненті.

У реалізованому веб компоненті буде впроваджена функціональність за для візуалізації геометричних моделей у веб просторі за допомогою бібліотеки Three.js.i

1.2 Огляд Vue та його порівняння з React

VueJs – це прогресивна основа для побудови користувальницьких інтерфейсів. На відміну від інших монолітних каркасів, VueJs розроблений з нуля, щоб бути поступово прийнятним або використовуватися разом з іншими веб технологіями. Основна бібліотека орієнтована лише на шар перегляду, і її легко підбирати та інтегрувати з іншими бібліотеками або існуючими проектами. З іншого боку, Vue також чудово здатний живити складні додатки для однієї сторінки, коли використовується в поєднанні із сучасними бібліотеками інструментів.

Порівнюючи Vue та React можливо побачити наступні збіги:

а) використовують virtual DOM;

б) надають реактивність і компонентну структуру;

в) фокусуються на кореневої бібліотеці, виносячи інші питання, такі як роутинг або управління глобальним станом додатка, в додаткові бібліотеки;

г) швидкі з точки зору швидкодії виконання операції.

Порівняємо Vue та React з точки зору зусиль для оптимізації. У React коли стан компонента змінюється, він запускає повторну отрисовку всього поддерева компонента, починаючи з себе. Щоб уникнути непотрібного повторного відтворення дочірніх компонентів, вам потрібно або використовувати PureComponent, або реалізовувати shouldComponentUpdate всюди де це можливо. Вам також може знадобитися використовувати незмінні структури даних, щоб зробити зміни вашого стану більш зручними до оптимізації. Однак, в деяких випадках ви не можете розраховувати на таку оптимізацію, тому що PureComponent / shouldComponentUpdate припускають, що відображення всього поддерева визначається даними поточного компонента. Якщо це не так, то така оптимізація може призвести до неузгодженого стану DOM.

У Vue залежності компонента автоматично відслідковуються під час відтворення, тому система точно знає, які компоненти дійсно необхідно повторно малювати при зміні стану. Кожен компонент можна розглядати як має shouldComponentUpdate, автоматично реалізований для вас, без обмежень для вкладених компонентів.

В цілому, це усуває необхідність цілого класу оптимізацій продуктивності для розробника, що дозволяє більше зосередитися на побудові самого додатка в міру його масштабування.

У React абсолютно все – це JavaScript. Не тільки структури HTML, виражені через JSX, останні тенденції також включають управління CSS всередині JavaScript. Цей підхід має свої переваги, але також змушує йти на компроміси, які можуть здатися неефективними для кожного розробника.

Vue охоплює класичні веб-технології та ґрунтується на них.

Виходячи з розглянутих факторі було обрано Vue тому що в нього краща оптимізації зі старту та мінімалістське базування на веб-технологіях, добре розвинута спільнота, чудові можливості за для масштабування та розробки проектів будь-якої складності.

1.3 Огляд стандартів веб компонентів

Веб-компоненти – сукупність стандартів, яка дозволяє створювати нові, призначені для користувача HTML-елементи зі своїми властивостями, методами та інкапсульованим DOM і стилями.

Веб-компоненти складаються з трьох основних технологій, які можна використовувати разом для створення універсальних спеціальних елементів із інкапсульованою функціональністю, які можна використовувати повторно, де завгодно, не боячись зіткнення коду:

а) спеціальні елементи: набір API JavaScript, який дозволяє визначати спеціальні елементи та їх поведінку, які потім можна використовувати за своїм бажанням у користувальницькому інтерфейсі;

б) shadow DOM: набір API JavaScript для прикріплення інкапсульованого "тіньового" дерева DOM до елемента, який відображається окремо від основного документа DOM – та контролю відповідного функціоналу. Таким чином можливо зберегти приватні функції елемента, так що вони можуть бути написані та стилізовані, не боячись зіткнення з іншими частинами документа;

в) шаблони HTML представлені елементами <template> та <slot> дозволяють писати шаблони розмітки, які не відображаються на відображеній сторінці. Потім їх можна повторно використовувати як основу структури користувацького елемента.

Базовий підхід до реалізації веб-компонента виглядає приблизно так:

а) створити клас або функцію, у якій потрібно вказати функціональність веб-компонента;

б) зареєструвати новий спеціальний елемент за допомогою методу CustomElementRegistry.define(), передавши йому ім'я елемента, яке потрібно визначити, клас або функцію, в якій вказана його функціональність, і необов'язково, який елемент він успадковує;

в) якщо потрібно, приєднати тіньовий DOM до спеціального елемента за допомогою методу Element.attachShadow() та додати дочірні елементи, слухачів подій до тіньового DOM, використовуючи звичайні методи DOM;

г) якщо потрібно визначити шаблон HTML за допомогою <template> та <slot> знову використовуючи звичайні методи DOM, щоб клонувати шаблон і прикріпити його до своєї тіні DOM;

д) використовувати власний елемент, де завгодно на своїй сторінці, як і будь-який звичайний HTML-елемент.

У ході виконя дипломної роботи під час реалізації веб компоненту застосуємо усі з необхідних кроків перерахованих вище за для для реалізації веб-компонента.

1.4 Огляд мови програмування та впровадження 3D у веб-простір

У якості основного інструмента реалізації дипломної роботи використовувалась мова програмування JavaScript – це інтерпретована мова програмування загального призначення.

Для виконання програм, існують два способи:

а) компіляція – це коли вихідний код програми, за допомогою спеціального інструменту, іншої програми, яка називається компілятор, перетворюється в іншу мову, як правило – в машинний код. Цей машинний код потім поширюється і запускається. При цьому вихідний код програми залишається у розробника;

б) інтерпретація – це коли вихідний код програми отримує інший інструмент, який називають інтерпретатор, і виконує його як є. При цьому поширюється саме сам вихідний код. Цей підхід застосовується в браузерах для JavaScript [2].

Сучасні інтерпретатори перед виконанням перетворюють JavaScript в машинний код або близько до нього, оптимізують, а вже потім виконують.

І навіть під час виконання намагаються оптимізувати. Тому JavaScript працює дуже швидко.

У всі основні браузери вбудований інтерпретатор JavaScript, саме тому вони можуть виконувати скрипти на сторінці. Але, зрозуміло, JavaScript можна використовувати не тільки в браузері. Це повноцінна мова, програми на якому можна запускати і на сервері, і навіть в пральній машинці, якщо в ній встановлений відповідний інтерпретатор.

У якості основної технології для реалізації 3D у веб було використано WebGL.

На відміну від інших технологій для роботи з тривимірною графікою (таких як OpenGL і Direct3D), WebGL призначена для використання в веб-сторінках і не вимагає установки спеціалізованих розширень або бібліотек. Одна з переваг WebGL – додатки конструюються як веб-сторінки, тобто одна і та ж програма буде успішно виконуватися на самих різних пристроях (наприклад, на смартфонах, планшетних комп'ютерах та ігрових консолях). Це означає, що WebGL буде надавати все більш посилюється вплив на співтовариство розробників і стане одним з основних інструментів програмування графіки.

З розвитком HTML розробники отримали можливість створювати все більш складні веб-додатки. На зорі свого розвитку мова HTML пропонував тільки можливість відображення статичного контенту, але з додаванням підтримки JavaScript стало можливим реалізовувати більш складні взаємодії елементів і відображення динамічного контенту. Впровадження стандарту HTML5 дозволило використовувати нові можливості, включаючи підтримку двомірної графіки у вигляді тега canvas [4].

Створення технології WebGL дозволило відображати і маніпулювати тривимірною графікою на веб-сторінках за допомогою JavaScript. За допомогою WebGL розробники можуть створювати абсолютно нові інтерфейси, тривимірні ігри і використовувати тривимірну графіку для візуалізації різної інформації. Незважаючи на значні можливості, WebGL відрізняється від інших технологій доступністю і простотою використання, що сприяє її швидкому поширенню.

Переваги використання WebGL:

а) кросбраузерність і відсутність прив'язки до певної платформі. Windows, MacOS, Linux – все це не має значення, головне, підтримка браузером WebGL;

б) використання мови JavaScript, яка є досить поширеною;

в) автоматичне управління пам'яттю. На відміну від OpenGL, в WebGL не треба виконувати спеціальні дії для виділення і очищення пам'яті;

г) оскільки WebGL для рендеринга графіки використовує графічний процесор на відеокарти, для цієї технології характерна висока продуктивність, яка порівнянна з продуктивністю нативних додатків.

Для WebGL потрібно два шейдера при кожному відображенні: верховий шейдер і фрагментний шейдер. Кожен шейдер – це функція. Верховий і фрагментний шейдери об'єднані в шейдерну програму. Зазвичай додаток на WebGL містить безліч шейдерних програм, за для їх написання використовують OpenGL Shading Language(GLSL).

GLSL – це мова високого рівня для програмування шейдерів.

Шейдер – це програма, яка виконується в графічному конвеєрі. Вона повідомляє комп'ютеру, як рендерить кожен піксель. Ці програми називаються шейдерами, тому що їх часто використовують для управління ефектами освітлення і затінення, але нічого не заважає використовувати їх і для інших спецефектів.

Синтаксис мови GLSL базується на мові програмування ANSI C, однак, з-за його специфічної спрямованості, з нього виключені багато можливостей, для спрощення мови та підвищення продуктивності. В мову включені додаткові функції та типи даних, наприклад для роботи з векторами та матрицями.

Основна перевага GLSL перед іншими шейдерними мовами – переносимість коду між платформами та операційними системами.

Отже WebGL у парі з GLSL представляє гарні перспективи за для реалізації компонента VueJs за для візуалізації геометричних моделей. Проте у наш час за

для написання компонентів з складної графікою зазвичай використовуються допоміжні бібліотеки, які працюють на базі описаних вище технологій.

1.5 Порівняння допоміжних бібліотек

У якості допоміжних бібліотек для реалізації 3D було розглянуто Three.js та Babylon.js.

Three.js – кросбаузерная бібліотека JavaScript, яка використовується для створення та відображення анімованої комп'ютерної 3D-графіки при розробці веб-прикладних програм, вона є найбільш популярною і активно розвиваючися бібліотекою на сьогоднішній день. Детальна та доступна документація та величезна кількість робочих прикладів роблять цю бібліотеку одним з провідних серед аналогічних систем.

Основні функціональні можливості Three.js:

а) рендерерінг – canvas, svg або webgl;

б) додавання та видалення об'єктів в режимі реального часу;

в) перспективна або ортографічна камера;

г) каркасна анімація, різні види кінематики, кадрове анімація;

д) декілька типів джерел світла – зовнішній, спрямований, точковий;

е) об'єкти – мережі, частинки, лінії, скелетна анімація.

Babylon.js – це бібліотека з відкритим вихідним кодом для створення повноцінних 3D-додатків і ігор, що працюють у веб-браузері без використання сторонніх плагінів та розширень. Babylon.js по своїх можливостях близький до Three.js, однак у своїй арсеналі має деякі вбудовані функції, недоступні в Three.js з коробки. До такого приємного особливості відносяться вбудований фізичний движок oimo.js – досить простий спосіб створити реалістичний ландшафт, використовуючи карту висот. Авжеж в Three.js також присутні такі можливості, але реалізовані вони за допомогою різних додаткових додатків. Однак за функціональність бібліотеки приходитися платити нескромним вагою.

Основні функціональні можливості babylon.js:

а) використання на сцені готових мешів, туман, скайбокс;

б) фізичний движок (модуль oimo.js), звуковий та анімаційний движок;

в) апаратне масштабування;

г) покрокове завантаження сцени з її оптимізацією;

д) чотири джерела освітлення – точковий, що випромінюється всюди, прожектор і реалістичне.

Не можливо визнати одну з розглянутих бібліотек значно кращою за іншу тому що обидві бібліотеки добре оснащенні функціональними можливостями за для рішення задач широкого спектру потреб. Але було обрано Three.js тому що він більш легковаговий, має добре організовану документацію з прикладами, більш розповсюджений.

1.6 Технічне завдання

У ході поставлення технічного завдання було вирішено реалізувати VueJs компонент на основі технологій веб-компонентів та бібліотеки Three.js. Також було вирішено покрити тестами написаний VueJs компонент візуалізації геометричних моделей використовуючи вбудовані засоби тестування з екосистемі VueJs. Також оскільки написаний VueJs компонент буде базуватися на веб-компонентах необхідно буде створити відповідні пакети з веб-компонентами та VueJs компонентом за для їх використання під час дипломної роботи та безкоштовного вільного розповсюдження та використання будь-ким у будь-якому веб середовищі яке підтримує Javascript.

Реалізований VueJs компонент повинен мати наступний функціонал:

а) рендеринг 2D та 3D моделей;

б) можливість отримання вхідних даних за допогою URL;

в) можливість отримання даних шляхом перетаскування файлу з вхідними даними у область їх відображення тобто технологія Drag-and-drop;

г) можливість вибору файлу через файловий менеджер веб-сервісу користувачем;

д) можливість маніпулювання зі сценою користувачем;

е) можливість включення/виключення відображення моделі у каркасному режимі та поворотів моделі.

2 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ VUEJS КОМПОНЕНТА

2.1 Архітектура VueJs компонента

Розробка архітектури є дуже важливим аспектом тому що цей процес включає сукупність рішень про організацію програмних систем, вибір структурних елементів і їх інтерфейсів, а також їх поведінки в рамках співпраці структурних елементів які з'єднуються у все більш крупні системи.

Як було вирішено на початку виконання дипломної роботи в основу VueJs компонента візуалізації геометричних моделей буде закладено за допомогою веб-компонентів з використанням бібліотеки Three.js. Перейдемо до розгляду архітектури створених веб-компонентів.

Відповідальним файлом за запуск реалізованих веб-компонентів є app.js файл який виконує ініціалізацію веб-компонентів. У свою чергу викликані екземпляри класу веб-компонента візуалізації та екземпляр класу веб-компонента контролів створюють самі веб-компоненти кожний для себе, при цьому вони виконують ініціалізацію DOM та систему подій кожний для себе.

Веб-компонент візуалізації використовує сервіс Helper, веб-компонент контролів сервіс ControlsHelper кожний з яких виконує певні унікальні налаштування за для кожного з компонентів, а також обидва раніше названих сервіси за для попередньо до них названих обох веб-компонентів використовують один загальний сервіс CreateElement за для створення DOM елементів. Також сервіс Helper використовує у процесі ініціалізації зчитування можливих початкових налаштувань веб-компонента візуалізації за допомогою Settings сервіса.

Після успішної ініціалізації веб-компоненти готові до використання, при виконуванні одного із передбачуваних сценаріїв отримання даних відбуваються наступна послідовність дій веб-сервісу. Перевірка на необхідність їх

форматування, якщо форматування потрібно то використовується TypeConverter сервіс, який у свою чергу за для задання загальної структури даних використовує SkeletonObject сервіс. Якщо форматування даних не потрібно то відповідні вхідні данні передаються до VisualizerController класу, який з VisualizerHelper сервісом будує кінцеву геометричну дискретну візуальну модель для користувача.

Сукупність взаємодії сервісів зображено на рисунку 2.1.

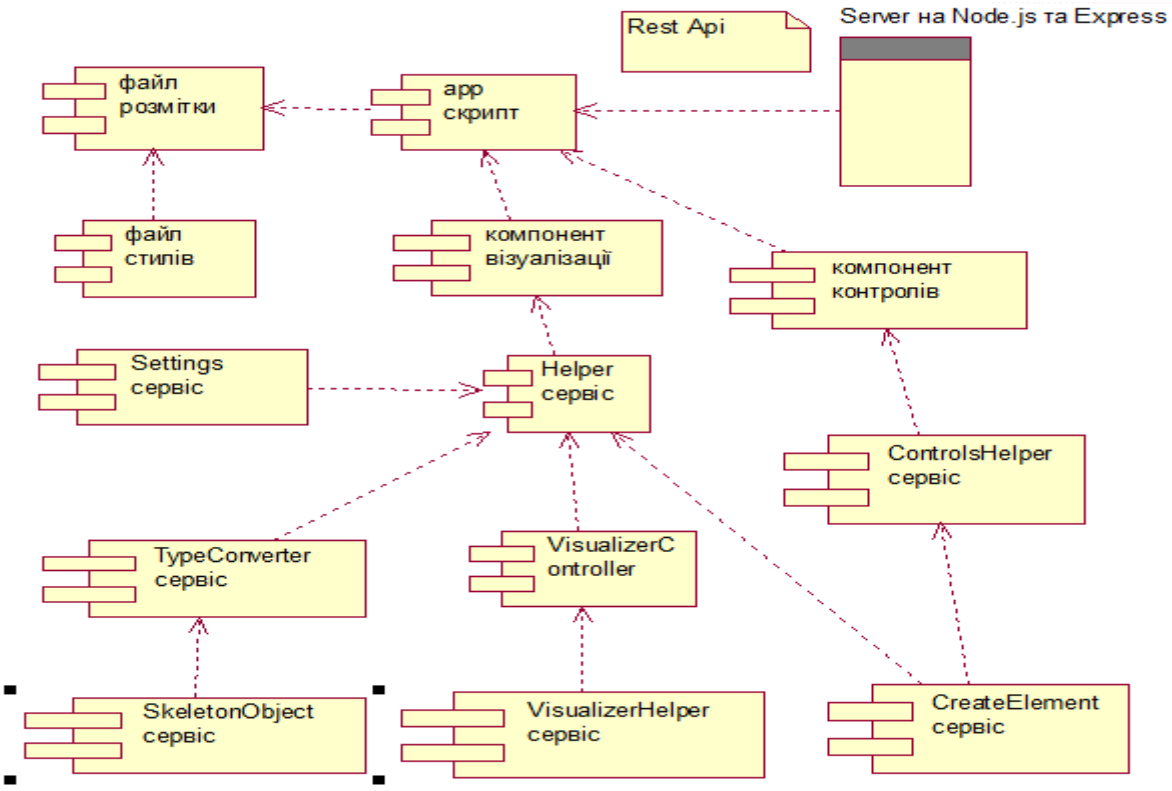

Рисунок 2.1 – Діаграма компонентів веб-сервісу візуалізації

Завдяки схемі зображеній вище можливо побачити загальну послідовність процесів роботи з описаними взаємодіями при ініціалізації веб-компонентів.

2.2 Проектування потоку подій з точки зору користувача

Проектування VueJs компоненту візуалізації геометричних моделей з точки зору користувача є дуже важливе тому що добре передбачивши усі

сценарії роботи з компонентом можливо організувати обробку усіх бажаних користувачем дій, тобто організувати сприйняття і дії у відповідь користувача, що виникають в результаті використання і/або подальшого використання продукції, системи або послуги. За для цього побудуємо один можливих потоків подій, визначених для прецеденту користувач, які відображені на діаграмі послідовності, відобразивши процес виконання цього прецеденту на рівні окремих дій у діаграмі діяльності, зображеної на рисунку 2.2.

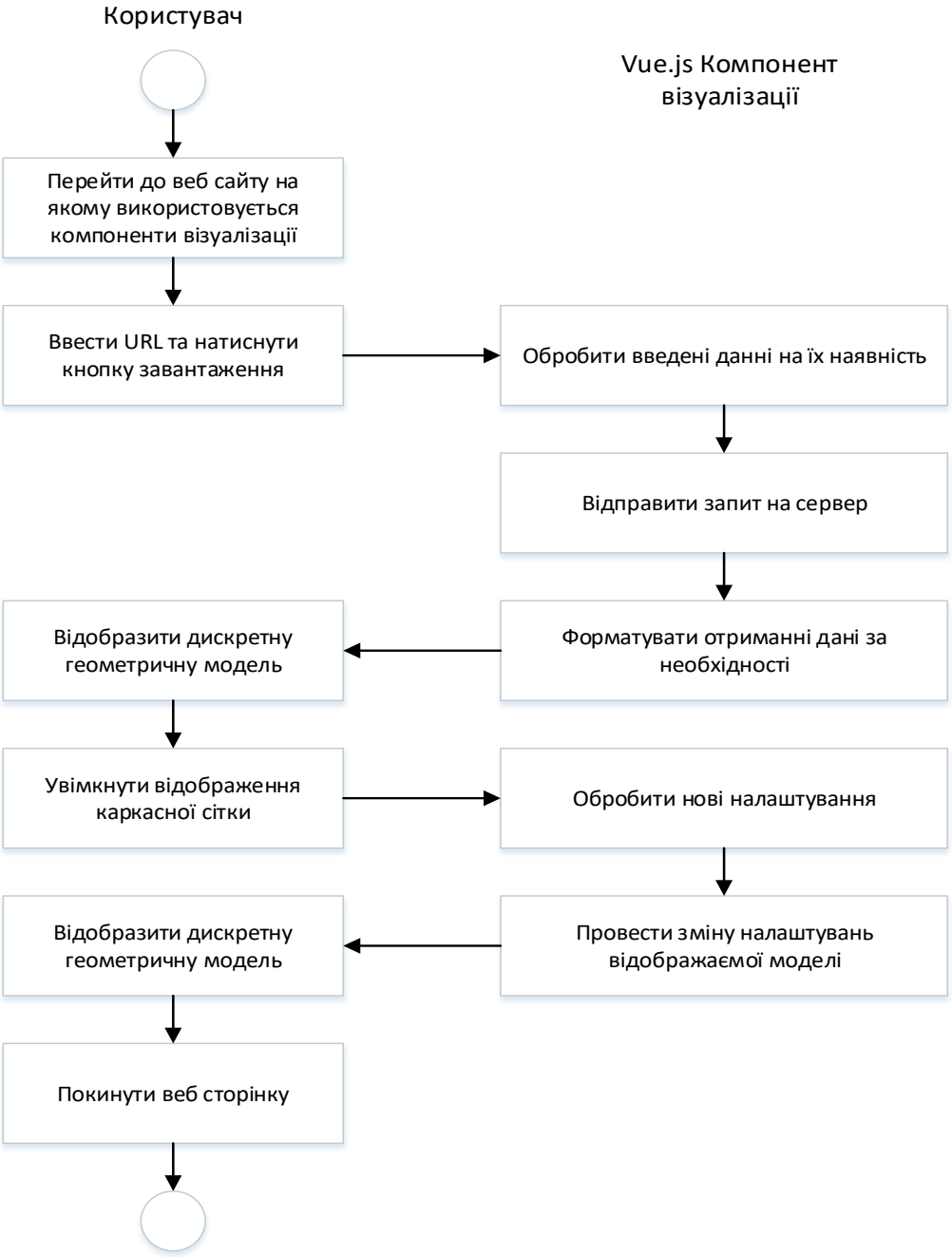

Рисунок 2.2 – Діаграма однієї з послідовностей діяльності користувача

2.3 Загальна структура VueJs компоненту та його ініціалізація

У ході виконання дипломної роботи було реалізовано VueJS компонент візуалізації геометричних моделей зображений на рисунках 2.3 та 2.4.

```
<script>

export default {
    name: 'Visualizer',
    props: {
        fildView: {
            type: Number,
            default: 40
        },
        wireframe: {
          type: Boolean,
          default: false
        },
        flatShading: {
            type: Boolean,
            default: false
        },
        width: {
            type: Number,
            default: 640
        },
        height: {
            type: Number,
            default: 480
        }
    },
    computed: {
        fildViewValue: {
            get() {return this.fildView;},
            set(value) {this.$emit('input', value);}
        },
        wireframeValue: {
            get() {return this.wireframe;},
            set(value) {this.$emit('input', value);}
        },
        flatShadingValue: {
            get() {return this.flatShading;},
            set(value) {this.$emit('input', value);}
        },
        widthValue: {
            get() {return this.width;},
            set(value) {this.$emit('input', value);}
        },
        heightValue: {
            get() {return this.height;},
            set(value) {this.$emit('input', value);}
        }
    }
}

</script>
```

Рисунок 2.3 – Функціональна складова VueJs компонента

```
<template>
    <div>
        <webgl-visualizer-controls></webgl-visualizer-controls>
        <webgl-visualizer :data-fild-view="fildViewValue || 40"
                          :data-width="widthValue || 640"
                          :data-height="heightValue || 480"
                          :data-wireframe="wireframeValue || false"
                          :data-flat-shading="flatShadingValue || false">
        </webgl-visualizer>
    </div>
</template>
```

Рисунок 2.4 – Шаблон VueJs компонента

Як можливо побачити з шаблону VueJs компонента його основа представлена двома веб-компонентами: webgl-visualizer та webgl-visualizer-controls – реалізованих по стандарту Custom Elements.

Custom Elements – це стандарт який дозволяє описувати для нових елементів свої властивості, методи, оголосити свій DOM, подібний конструктор та багато іншого.

Код створення webgl-visualizer компоненту зображено рисунку 2.5.

```
import Helper from './services/helper.js';

export default class InitializerComponent extends HTMLElement {
    constructor() {
        super();
    }

    connectedCallback() {
        this.initWebglVisualizer();
    }

    initWebglVisualizer() {
        let webglVisualizers = document.getElementsByTagName('webgl-
visualizer');

        let webglVisualizer = webglVisualizers[0];

        this.helper = new Helper(webglVisualizer);
        this.helper.setWindowSettings();
        webglVisualizer.appendChild(this.helper.createComponentFragment());
    }
}
```

Рисунок 2.5 – Клас веб-компоненту візуалізації

Важливим аспектом який спонукав до використання веб-компонентів це є те що вони можуть бути використані у будь-якому Javascript середовищі.

Під час підключення VueJs компонента та веб-компоненту візуалізації можливо передати початкові налаштування, але якщо вони не будуть передані у якості атрибутів VueJs компонента або веб-компоненту то у компонентах будуть використанні стандартні налаштування це надає певну гнучкість у використанні компонентів візуалізації.

Після ініціалізації компонентів візуалізації геометричних моделей з зображеними раніше налаштуваннями кінцевий користувач повинен побачити інтерфейс зображений на рисунку 2.6.

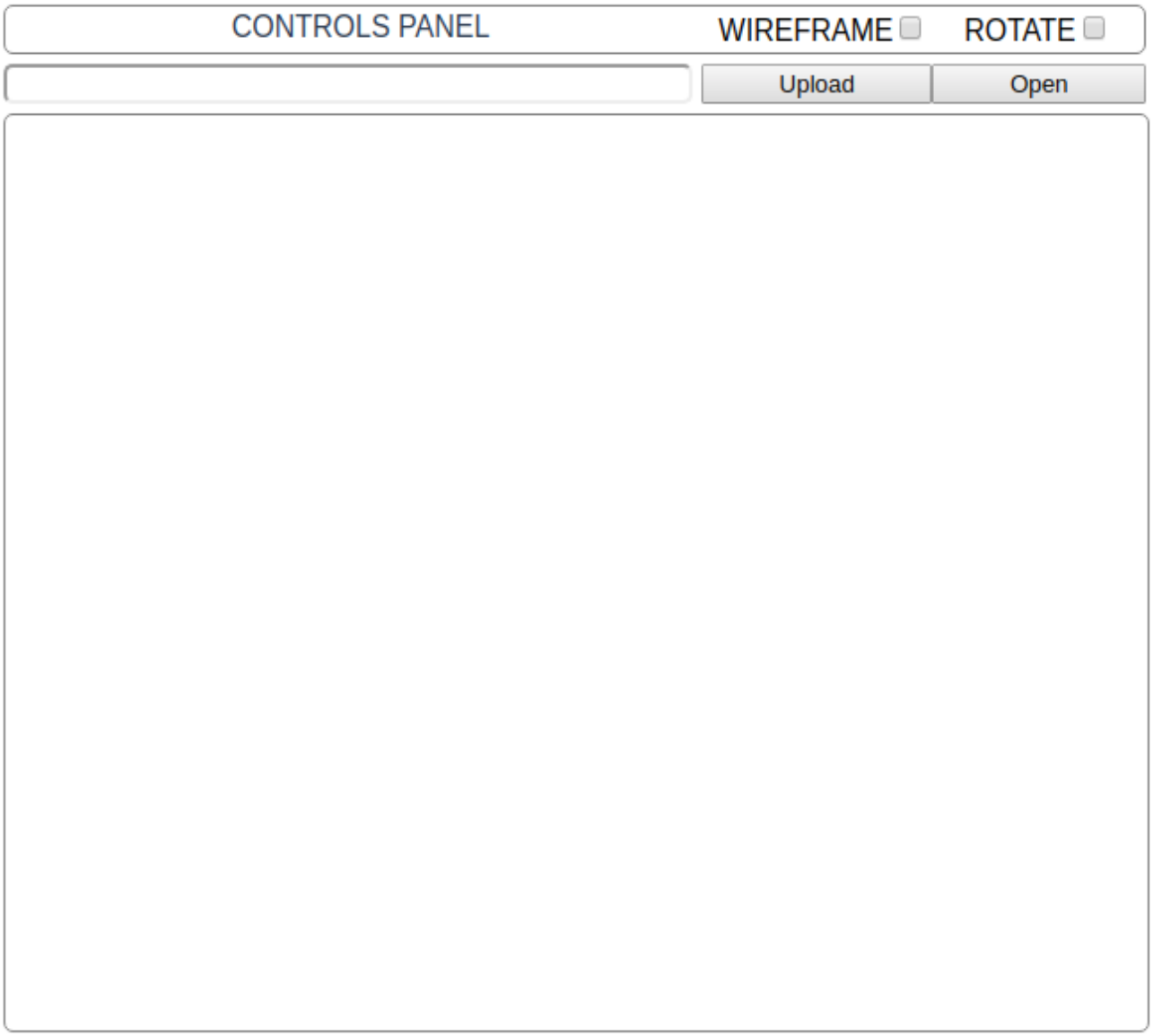

Рисунок 2.6 – Користувацький інтерфейс після ініціалізації веб-сервісу

2.4 Опис процесу отримання налаштувань компонентами

Під час процесу ініціалізації VueJs компонента та веб-компонентів описаних у попередньому підрозділі виконуються зчитування налаштувань візуалізації переданих у атрибутах або використання стандартних. Розглянемо передані атрибути на прикладі веб-компонента. Загалом до веб-компоненту візуалізації можливо передати наступні налаштування при ініціалізації такі як:

а) data-width – ширина веб-компонента;

б) data-height – висота веб-компонента;

в) data-fild-view – масштаб сцени, який видно на дисплеї в будь-який момент в градусах;

г) data-wireframe – включення та виключення каркасної сітки;

д) data-flat-shading – включення та виключення плоского затінення.

Код процесу ініціалізації описаних параметрів зображено на рисунку 2.7.

```
export default class Settings {
    constructor(component) {
        this.component = component;
    }

    initSettings() {
         this.settings = {
             fildView: this.component.getAttribute('data-fild-view') || 70,
             width: this.component.getAttribute('data-width') ||
window.innerWidth,
             height: this.component.getAttribute('data-height') ||
window.innerHeight,
             meshMaterialSettings: {
                 wireframe: this.getMeshSettings('data-wireframe'),
                 morphTargets: this.getMeshSettings('data-morph-targets'),
                 flatShading: this.getMeshSettings('data-flat-shading')
             }
         };
      return this.settings;
}
```

Рисунок 2.7 – Клас Settings – сервіс ініціалізації початкових налаштувань

```
    getMeshSettings(type) {
        let meshSetting = this.component.getAttribute(type);
            if(meshSetting !== null) {
                return meshSetting.toLowerCase() === 'true';
            } else {
                return false;
            }
       }
    }
```

Також під час проектування веб-компонентів було використано технологію Custom Event – які називають штучним подіями, по відношенню до подій, виробленим браузером. За допомогою технології Custom Event у якій веб-компонент webgl-visualizer-controls генерує події на кореневому елементі за допомогою dispatchEvent, а зовнішній код компонента webgl-visualizer ставить обробники за допомогою addEventListener було реалізовано систему подій, яка дає змогу змінювати налаштування під час роботи Vuejs компоненту та веб-компонентів візуалізації код цієї реалізації зображено на рисунках 2.8 та 2.9.

```
    wireframe.addEventListener('click', () => {

                this.wireframeStatus = wireframe.checked ? true :
                                                           false;

                let changeSettingsEvent = new CustomEvent('changeSettings',
                    {
                        detail: {
                            name: 'wireframe',
                            status: this.wireframeStatus
                        }
                    });

                webglVisualizer.dispatchEvent(changeSettingsEvent);

    }
);
```

Рисунок 2.8 – Код посилання Custom Event події

```
      setListeners() {
              let   webglVisualizers   =   document.getElementsByTagName('webgl-
visualizer');
              let webglVisualizer = webglVisualizers[0];

              webglVisualizer.addEventListener('changeSettings', (e) => {

                  this.mesh.material[e.detail.name] = e.detail.status;
              }, false);

              webglVisualizer.addEventListener('rotationSettings', (e) => {
                  this.rotate = e.detail.statusRotate;
              }, false);
            }
```

Рисунок 2.9 – Код обробки Custom Event події

За допомогою технологію Custom Event можливо створювати подієві системи будь-якої складності.

2.6 Користувацьке тестування роботи VueJs компоненту

Після ініціалізації VueJs компоненту візуалізації геометричних моделей користувач зможе побачити інтерфейс зображений у підрозділі ініціалізації, який був зображений раніше.

Користувач має змогу завантажити вхідні данні своєї дискретної геометричної моделі за наступними сценаріями:

а) ввівши URL до строки вводу, яка знаходиться у верхній частині компоненту візуалізації, та натиснувши кнопку upload за для здійснення запиту на сервер, якій повинен повернути необхідні данні;

б) перетягнувши файл до області візуалізації компонента тобто цей сценарій представляє Drag-and-drop;

в) обрати необхідний для візуалізації файл шляхом виклику файлового менеджеру який викликається взаємодією з кнопкою open розташованої у верхній лівій частині компоненту.

Виконуючи будь-який з описаних сценаріїв отримання моделі можливо відобразити наступний результат роботи VueJs компоненту, який зображений на рисунку 2.10.

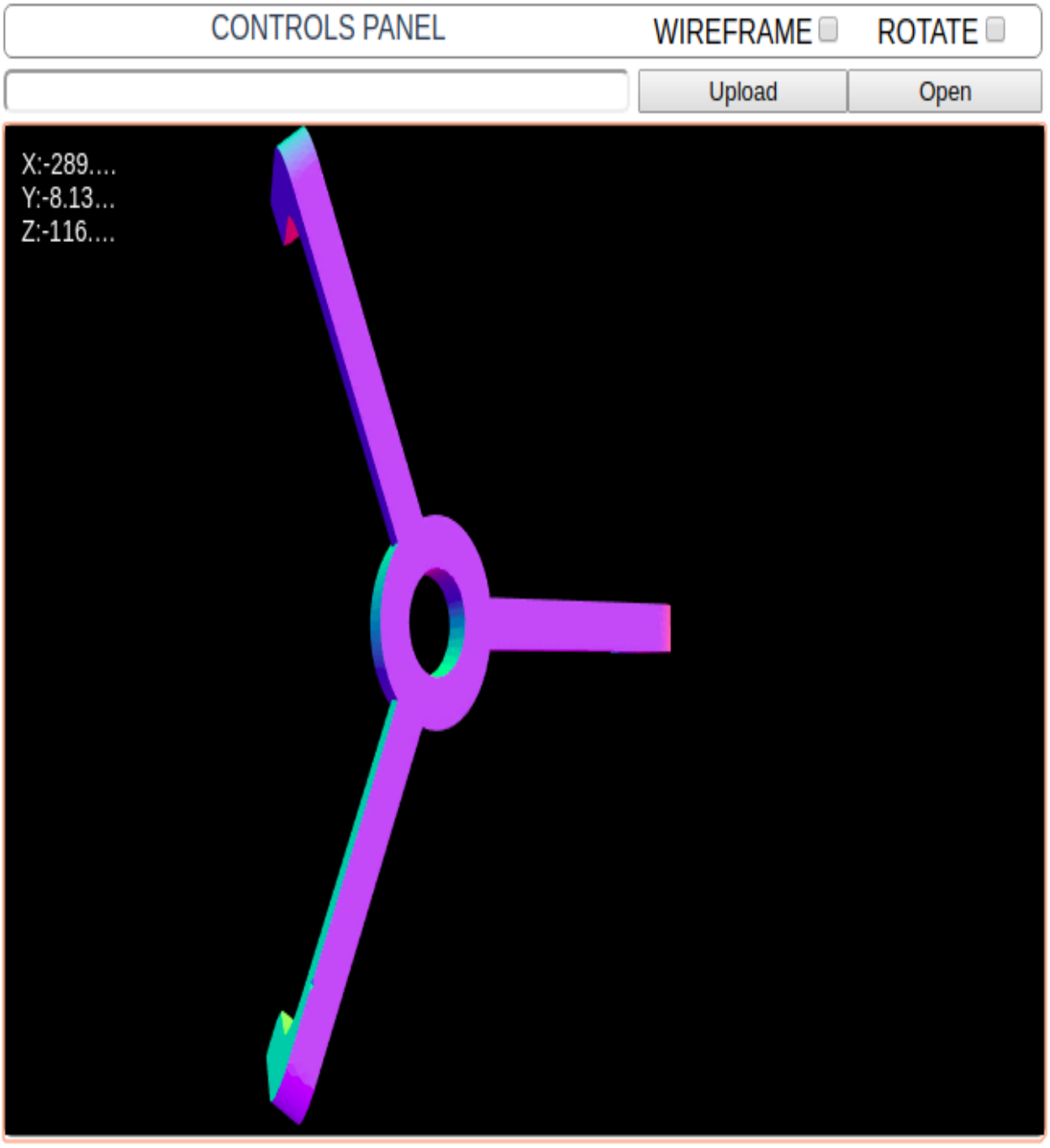

Рисунок 2.10 – Стандартний результат роботи програми

Важливо зауважити що необхідна дискретна геометрична модель якнайкраще повинна бути описана у форматі JavaScript Object Notation(JSON), проте якщо це будуть данні текстовому форматі компонент візуалізації трансформує з них необхідну структуру формату JSON.

Також важливим аспектом є те що після початкової ініціалізації та першої завантаженої геометричної дискретної моделі веб-сервіс може оброблювати нові вхідні данні усіма описаними раніше способами. З налаштувань доступних режимі роботи додатку можливо вимкнути/увімкнути обертання вибраної моделі чи вимкнути/увімкнути відображення каркасної сітки геометричної дискретної моделі, яку можливо побачити увімкненою на наступному рисунку 2.11 та 2.12.

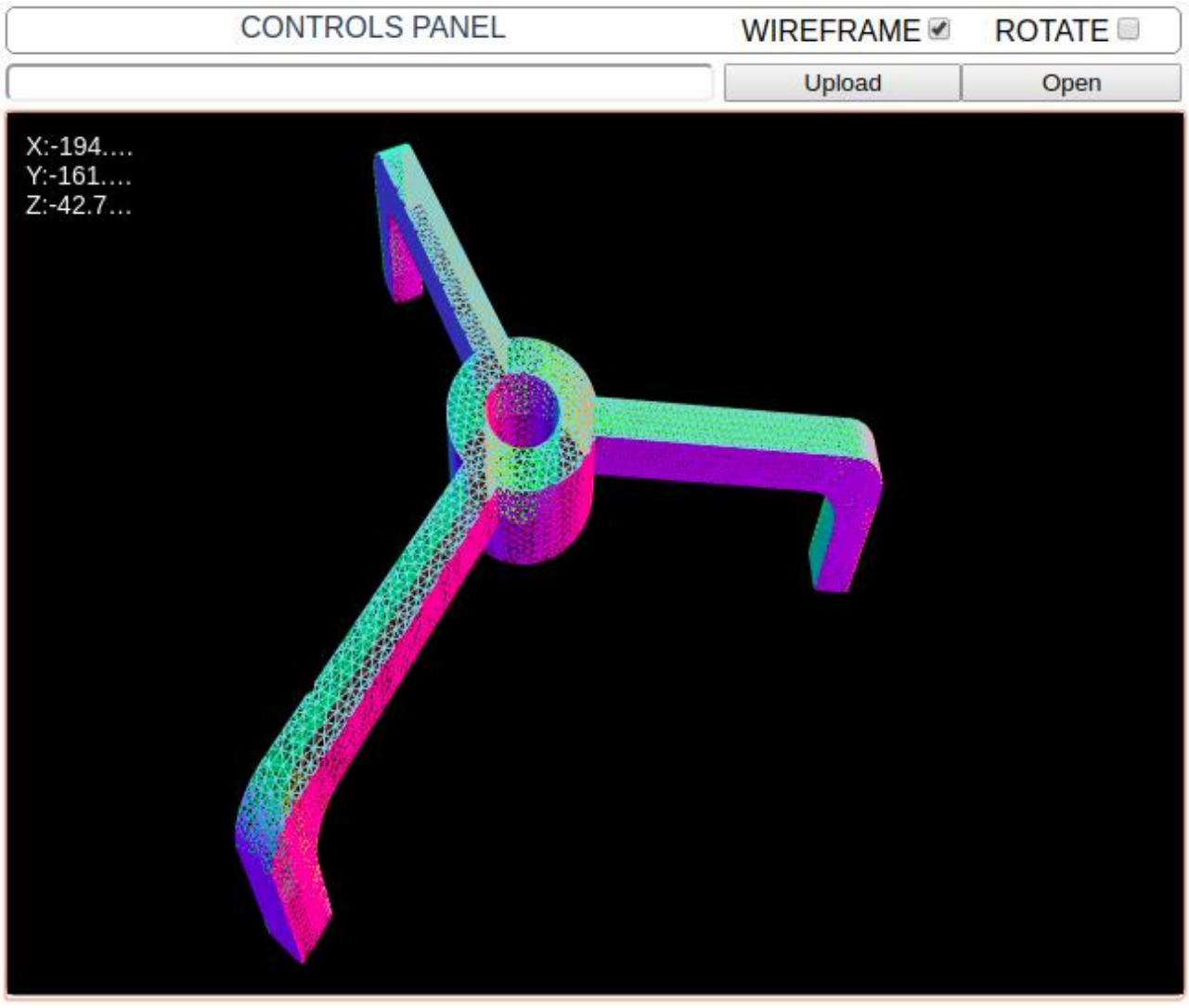

Рисунок 2.11 – Результат роботи програми з увімкненим відображенням каркасної сітки геометричної дискретної моделі

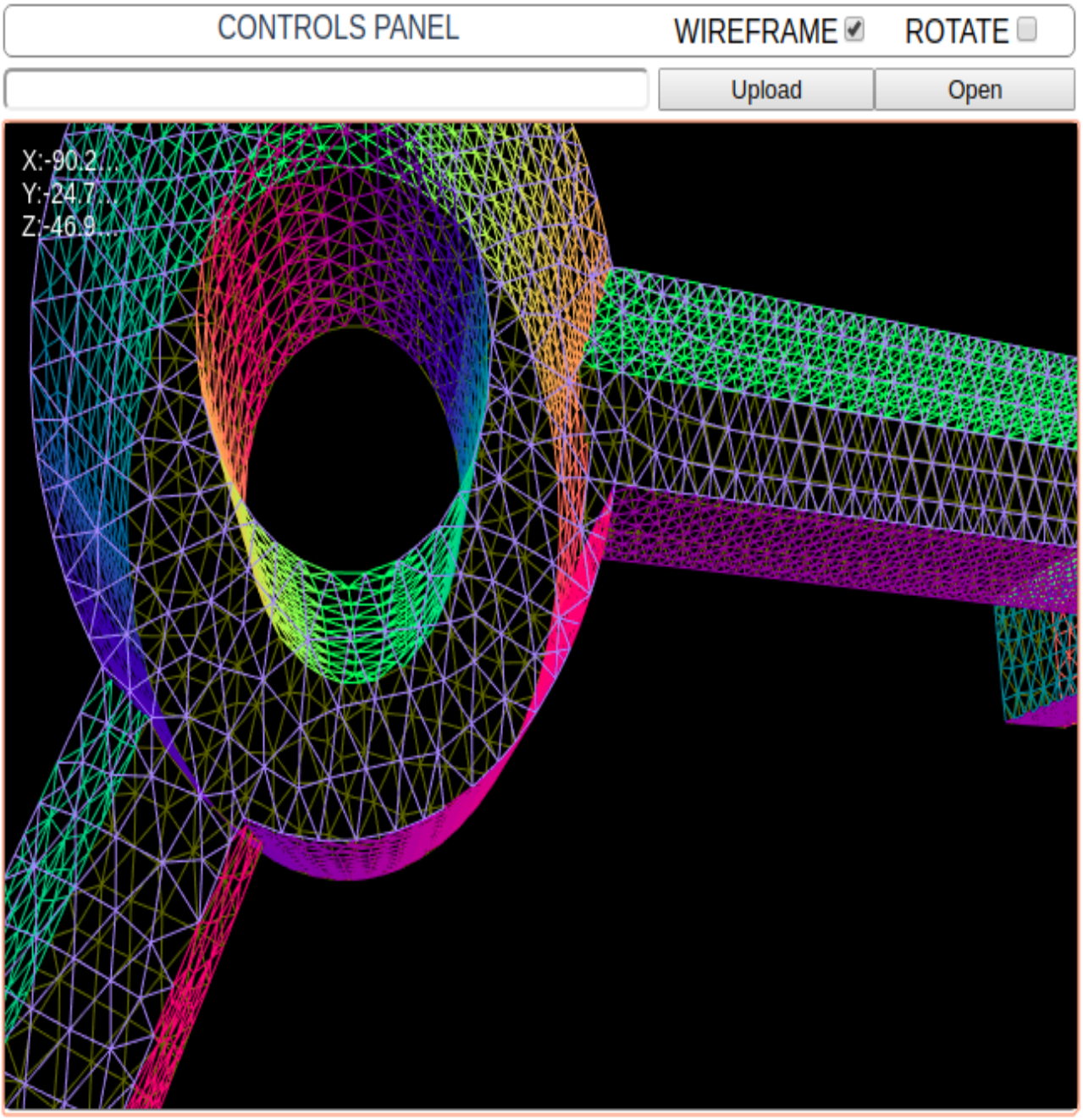

Рисунок 2.12 – Відображення каркасної сітки геометричної дискретної моделі при приближенні до неї

З не оголошених аспектів реалізації VueJs компоненту можливо побачити відображення поточних координат знаходження камери на сцені у лівому верхньому куті області візуалізації однойменного компонента. Маніпуляції з зміною координат камери на сцені відбуваються завдяки діям користувача маніпулятором миші, та програмно завдяки OrbitControls модулю який подається офіційно від Three.js.

Також передбачена можливість обробки натиснення кнопки upload при невведеному URL, результат цієї обробки зображено на рисунку 2.13.

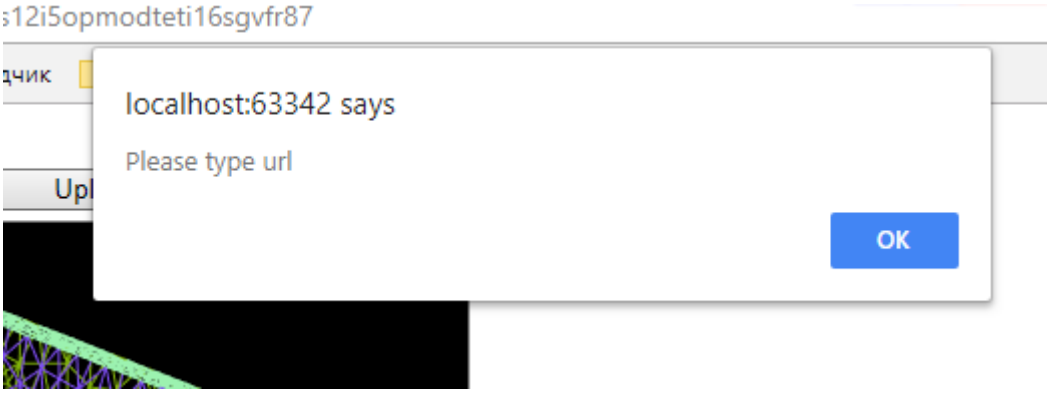

Рисунок 2.13 – Обробка невведеного URL при спробі відправити запит

Також у реалізованому компоненті можливо вибирати файл з файлового вікна шляхом його виклику, який зображено на наступних рисунках 2.14 та 2.15.

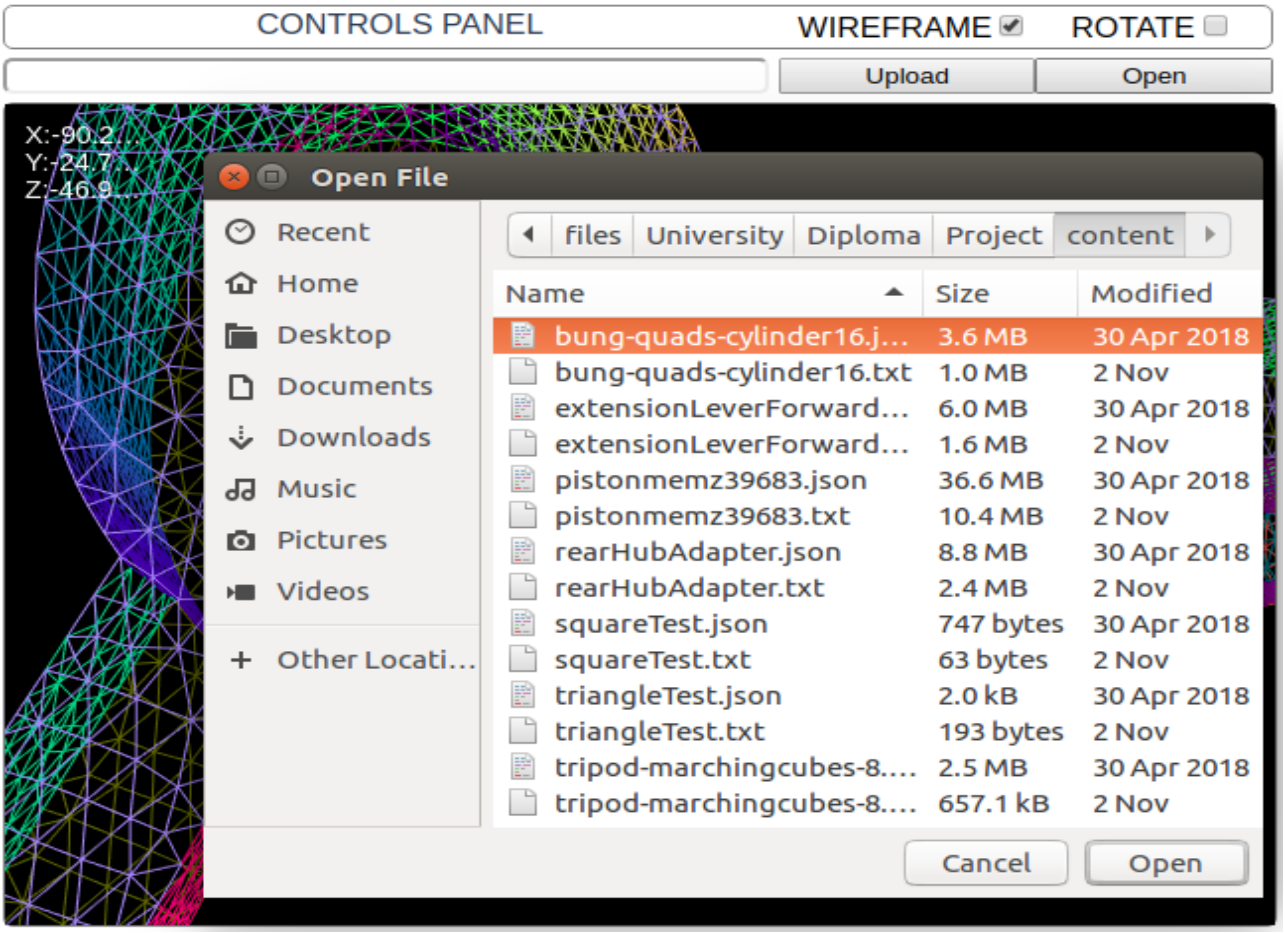

Рисунок 2.14 – Виклик вікна вибору файлів за для подальшої візуалізації

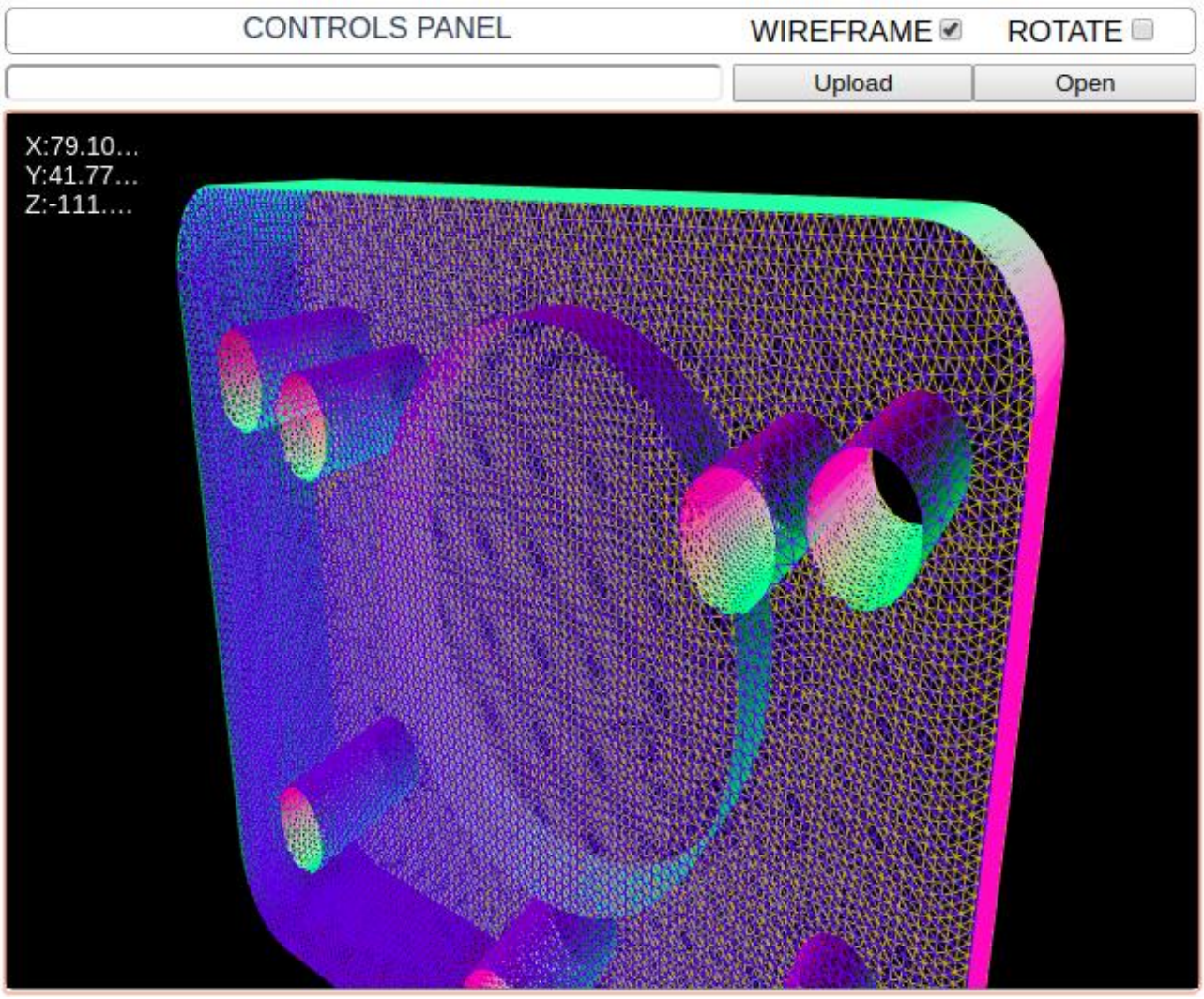

Рисунок 2.15 – Результат роботи програми

3 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ДОПОМІЖНИХ ЗАСОБІВ

3.1 Проектування допоміжного серверу

За для реалізації та виконання поставлених технічних завдань було спроектовано та реалізовано сервер на базі Node.js.

Node.js – програмна платформа, заснована на движку V8 який транслює JavaScript в машинний код, що перетворює JavaScript з вузькоспеціалізованої мови в мову загального призначення. Node.js додає можливість JavaScript взаємодіяти з пристроями введення-виведення через свій API, який може бути написаний на C ++, підключати інші зовнішні бібліотеки, написані на різних мовах, забезпечуючи виклики до них з коду JavaScript. Node.js застосовується переважно на сервері, виконуючи роль веб-сервера, але є можливість розробляти на Node.js віконні додатки використовуючи NW.js, AppJS або Electron для Linux, Windows і macOS і навіть програмувати мікроконтролери за допомогою tessel і espruino. В основі Node.js лежить подієво-орієнтоване, а також асинхронне програмування з неблокуючим введенням / виведенням даних [5].

Також для написання серверу на Node.js був використаний Express.

Express – це мінімалістичний і гнучкий веб-фреймворк для додатків Node.js, що надає великий набір функцій для мобільних і веб-додатків. Express має в своєму розпорядженні безліч службових методів HyperText Transfer Protocol (HTTP) і проміжних оброблювачів, створити надійний API можна швидко і легко. Також він надає фундаментальні функцій веб-додатків які добре поєднуються з Node.js [3].

Першочергова задача яка була виконана за допомогою Node.js це написання серверу який повертав би користувачу необхідні данні файлів, доступ до яких був обраний за вказаним користувачем URL, які сервер зберігає у своїй відповідній директорії. Тобто з написанням цього функціоналу можливо

перевірити можливість компонентів візуалізації отримання даних за вказаним URL. За для вище перерахованих цілей був написаний наступний сервер, основна частина логіки якого зображена кодом на наступному рисунку 3.1.

```
app.use(cors());
app.get('/', (req, res) => res.send('Working!'));
app.use('/static', express.static(constant.contentFolder));
app.listen(3000, () => {console.log('App listening on port 3000!');});
```

Рисунок 3.1 – Код сервера який повертає запитанні користувачем файли

Код сервера який зображений на попередньому рисунку є небагатослівним тобто у сенсі кількості коду, проте він є змістовим. Зображена змістовність досягається вдалим вибором інструментів таких як Node.js та Express та їх розумним використанням.

Також з використанням Node.js був написаний скрипт-адаптер, який аналогічно описаним механізмам, які використовуються у веб-компоненті перетворює файли з хаотичними даними у текстовому форматі у структуровано описаний об’єкт формату JSON. А саме вибирає з вказаної директорії усі файли з розширенням текстового формату та перетворює їх формат JSON з необхідною для користувача структурою, для його миттєвого використання отриманих даних без перетворювань основний код описаного допоміжного скрипта зображений на рисунку 3.2.

```
let adapterHelper = new AdapterHelper(),
    resJson;

fs.readdirSync(path.join(__dirname,
constant.contentFolder)).forEach((file) => {
    if(file.match(/.txt$/)) {
        adapt(file.slice(0, -4));
    }
});
```

Рисунок 3.2 – Код допоміжного серверного скрипта перетворювача

```
      function adapt(name) {
          try{
              fs.readFile(`${constant.contentFolder}${name}.txt`, 'utf8',
function (err, data) {
                  if(err) throw err;
                  resJson = adapterHelper.convertTxtJson(data);
                  fs.writeFile(`${constant.contentFolder}${name}.json`, resJson,
function (err) {
                      if(err) throw err;
                      console.log(`File ${name}.json saved succeed!`);
                  });
              });
          } catch (err) {console.log(err);}}
```

Отже з використанням таких інструментів як Node,js та Express було побудовано необхідний сервер для компонентів візуалізації. З головних переваг Express хотілось би відзначити можливість швидкої розробки додатків за допомогою вбудованих модулів та спільноту з відкритим кодом. З головних переваг Node.js хотілось би відзначити його асинхронність, яка ідеально підходить за для додатків на кшталт серверу для VueJs компоненту який оброблює роботу файлів наступним шляхом:

а) відправляє завдання до файлової системи комп'ютера;

б) готовий обробляти наступний запит;

в) коли файлова система відкриє та прочитає файл, а потім форматує файл якщо потрібно, сервер повертає вміст клієнту.

Таким чином використання Node.js та Express впроваджують наступні переваги при їх застосуванні:

а) усування очікування, просте продовження обробки списку запитів, після отримання попереднього;

б) однопотоковість та неблокованість;

в) асинхронність;

г) ефективність використання пам'яті;

д) легкість при масштабуванні завдяки модульній системі.

3.2 Тестування VueJs компоненту unit тестами

Під час реалізації будь-якого сучасного додатку зазвичай використовується не тільки користувацьке тестування, а й програмне, розробка даного типу тестів не була пропущена, було реалізовано unit тести. Unit тести дозволяють перевірити на коректність окремі модулі вихідного коду програми, набори з одного або більше програмних модулів разом з відповідними керуючими даними, процедурами використання і обробки. За для реалізації тестів використовувався Vue Test Utils який постачається разом з Vue CLI – повноцінною системою для швидкої розробки на Vue.js.

А у самому Vue Test Utils використовується Jest – тестовий-бігун який легко конфігурується, встановлює JSDOM за замовчуванням, надає вбудовані твердження та має великий досвід користування командним рядком.

Впроваджені тести та результати роботи тестів можливо побачити на рисунках 3.3 та 3.4.

```
france@france:~/Documents/files/University/Diploma/Project/visualizer-vue$ npm run test:unit

> @mister-france/vue-visualizer@0.0.7 test:unit /home/france/Documents/files/University/Diploma/Project/visualizer-vue
> vue-cli-service test:unit

 PASS  tests/unit/visualizer.spec.js
  Visualizer init
    ✓ has a webgl-visualizer-controls (6ms)
    ✓ has a webgl-visualizer (1ms)
  Visualizer props
    ✓ width is equal 400 (1ms)
    ✓ height is equal 200
    ✓ wireframe is equal true (1ms)
  Visualizer events
    ✓ Visualizer has event update:wireframe

Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        0.8s, estimated 1s
Ran all test suites.
```

Рисунок 3.3 – Результат роботи тестів

```
import Vue from 'vue';
import { mount, shallowMount} from '@vue/test-utils';
import Visualizer from '../../src/components/Visualizer';

describe('Visualizer init', () => {
  const wrapper = mount(Visualizer);

  it('has a webgl-visualizer-controls', () => {
    expect(wrapper.contains('webgl-visualizer-controls')).toBe(true);
  });

  it('has a webgl-visualizer', () => {
    expect(wrapper.contains('webgl-visualizer')).toBe(true);
  });
});

describe('Visualizer props', () => {
  const wrapper = shallowMount(Visualizer, {
    propsData: {
      width: 400,
      height: 200
    }
  });

  it('width is equal 400', () => {
    expect(wrapper.find('webgl-visualizer').attributes('data-
width')).toBe('400');
  });

  it('height is equal 200', () => {
    expect(wrapper.find('webgl-visualizer').attributes('data-
height')).toBe('200');
  });

  wrapper.setProps({wireframe: true});

  it ('wireframe is equal true', () => {
    expect(wrapper.find('webgl-visualizer').attributes('data-
wireframe')).toBe('true');
  });
});

describe('Visualizer events', () => {
  const wrapper = mount(Visualizer);

  it('Visualizer has event update:wireframe', () => {
    Vue.nextTick(() => {
      expect(wrapper.emitted('update:wireframe')).toBe(undefined);
    });
  });
});
```

Рисунок 3.4 – тести VueJs компонента

У Представлених тестах можливо виділити три групи такі як тести ініціалізації, тести вхідних параметрів та тести подій VueJs компонента візуалізації геометричних моделей. Перша група тестів – ініціалізаційні тести перевіряють знаходження веб компонентів webgl-visualizer та webgl-visualizer-controls у VueJs компоненті. Друга група тестів – тести вхідних параметрів, розроблена наступним чином під час ініціалізації обгортки Visualizer VueJs компонента йому передаються вхідні параметри, а потім виконується перевірка на установку переданих параметрів веб компонентами які знаходяться у VueJs компоненті. Також після перевірки ініціалізаційних параметрів виконується подальше налаштування параметрів та їх перевірка за впровадженою вище схемою, але вже під час роботи VueJs компонента. Третьої групою – виступають подієві тести у цій групі перевіряється обробка події update:wireframe яка відбувається після ініціалізації VueJs компоненту.

Виходячи з представлених даних по тестам та їх результатів роботи можливо побачити що усі тести пройшли успішно це підтверджує правильність написання додатку.

3.3 Розповсюдження реалізованих компонентів

Для розповсюдження реалізованого VueJs компоненту та веб-компонентів їх було опубліковано як наступні два модуля @mister-france/visualizer та @mister-france/vue-visualizer на Node package manager(NPM). NPM – це менеджер пакетів, що знаходяться в складі Node.js. Як зазначалось раніше головною метою публікації розробленого функціоналу є його вільне використання будь-ким, а також використання у ході дипломної роботи, тобто використання першого модулю @mister-france/visualizer з веб-компонентами у складі проекту з VueJs компонентом, а потім його публікація як @mister-france/vue-visualizer, а також перевірка у використанні в середовищі VueJs. Усі наведені вище кроки було реалізовано. Як можливо було зазначити вище ім'я

кожного з пакетів починається з префіксу @mister-france це було зроблено за для ідентифікування пакетів автором, це є однією з рекомендації за для опублікування пакетів. Під публікації представлених раніше пакетів було створено аккаунт на NPM, а потім розширено файли package.json кожного з розроблених проектів. Розглянемо для прикладу один з них, файл модуля @mister-france/visualizer який був розширений до виду зображеного на наступному рисунку 3.5.

```
{
  "name": "@mister-france/visualizer",
  "version": "0.0.13",
  "private": false,
  "publishConfig": {"access": "public"},
  "description": "",
  "main": "src/app.js",
  "scripts": {
    "dev": "webpack --watch",
    "dev-server": "webpack-dev-server --inline --hot",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "files": [
    "build/*",
    "src/*",
    "*.json",
    "*.js"
  ],
  "repository": {
    "type": "git",
    "url": "git+https://bitbucket.org/france1337/webgl.git"
  },
  "author": "Onopriienko Ruslan",
  "license": "ISC",
  "homepage": "https://bitbucket.org/france1337/webgl#readme",
  "devDependencies": {
    "babel-core": "^6.26.3",
    "babel-loader": "^7.1.4",
    "babel-preset-env": "^1.7.0",
    "uglifyjs-webpack-plugin": "^1.2.7",
    "webpack": "^4.41.2",
    "webpack-cli": "^3.0.8",
    "webpack-dev-server": "^3.9.0"
  },
  "dependencies": {
    "@webcomponents/webcomponentsjs": "^2.4.0",
    "three": "^0.110.0"}}
```

Рисунок 3.5 – package.json файл модулю @mister-france/visualizer

Важливими за для публікації пакету є наступні поля:

а) name – назва пакету;

б) version – версія пакету складається з трьох цифр, перша змінюється тільки коли порушують зворотну сумісність, друга зворотно сумісна змінюється за для впровадження нового функціоналу, третя зворотно сумісна змінюється за для виправлення помилок;

в) author – ім’я автора;

г) files – файли які будуть зібрані до пакету;

д) main – точка входу до функціоналу впровадженого у пакеті;

е) license – ліцензія за для розповсюдження пакету.

Сама публікації виконується після входу у NPM систему за допомогою команди npm login, після чого можливо опублікувати пакети за допомогою команди npm publish. Розглянемо результати публікації @mister-france/visualizer пакету на наступних рисунках 3.6 та 3.7.

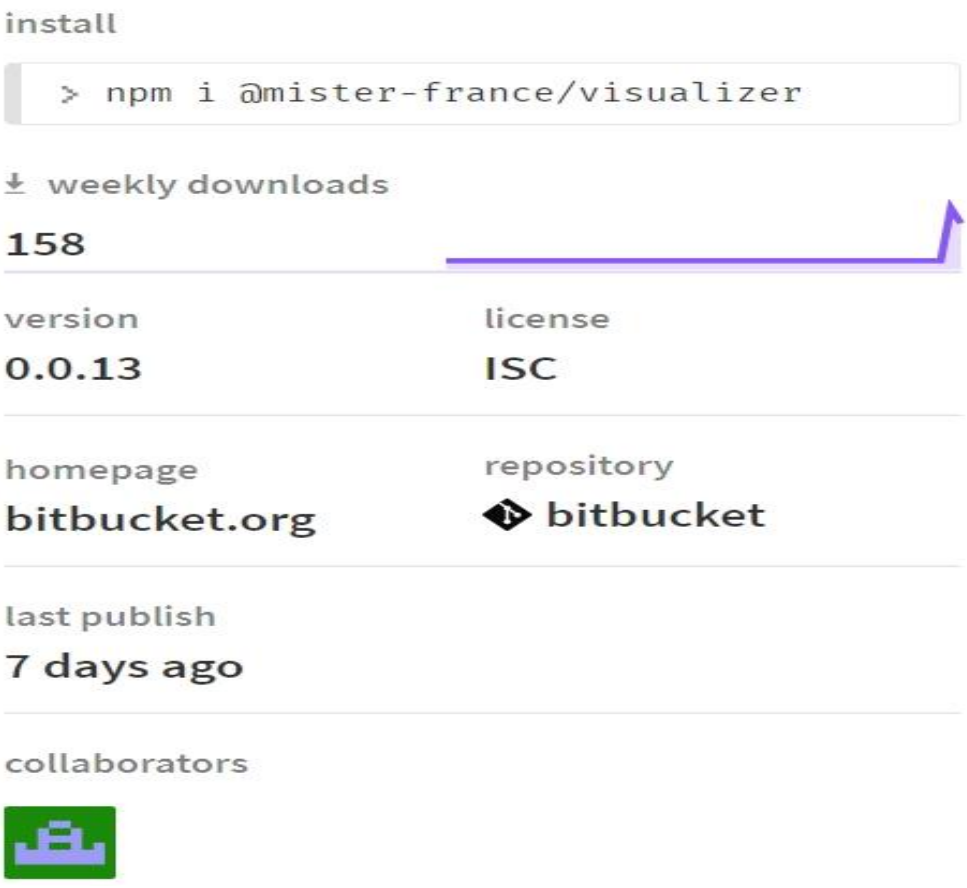

Рисунок 3.6 –Загальна інформація @mister-france/visualizer пакету

Project @mister-france/visualizer

@mister-france/visualizer - allow the creating of an application web with the ability to model discrete geometric models of any complexity.

Installing

```
npm i @mister-france/visualizer
```

Setup

```
import '@mister-france/visualizer'
```

Usage

```
<webgl-visualizer-controls></webgl-visualizer-controls>
<webgl-visualizer data-fild-view="40"
                  data-width="640"
                  data-height="480"
                  data-wireframe="false"
                  data-flat-shading="false">
</webgl-visualizer>
```

Рисунок 3.7 – Інформація по використанню @mister-france/visualizer пакету

Згідно з представленою інформації за допомогою NPM можливо побачити близько 150 скачувань за тиждень пакету, аналогічна ситуацію спостерігається із з пакетом @mister-france/vue-visualizer данні цифри можливо зчитати гарним стартом життя розроблених пакетів.

ВИСНОВКИ

У процесі виконання дипломної робити за темою: «Розробка компонента VueJs для візуалізації геометричних моделей» було здобуті нові та повторенні вже знайомі корисні навички робити та їх взаємодії перед усім з такими технологіями як: JavaScript та Vue.js, Three.js, Node.js та Express, а також Webpack, Babel та інші.

Під час виконання дипломної роботи були вирішенні наступні поставленні задачі як:

а) відображення 2D та 3D моделей у веб просторі та можливість маніпулювання зі сценою;

б) можливість отримання вхідних даних за допомогою URL;

в) можливість отримання даних шляхом перетаскування файлу з вхідними даними у область їх відображення тобто технологія Drag-and-drop;

г) можливість вибору файлу через файловий менеджер веб-сервісу користувачем;

д) можливість включення/виключення відображення моделі у каркасному режимі та її поворотів;

е) публікація пакету з VueJs компонентом та пакету з веб-компонентом за для можливості використання безкоштовно будь-ким у будь-якому веб середовищі.

У подальшій перспективі функціонал Vue.js компонента візуалізації геометричних моделей може бути розширений.

Серед його переваг хотілось би відзначити новизну ідеї аналогів якої на просторах всесвітнього павутиння не було знайдено, простоту та легкість дизайну, надійність роботи додатку, добру анімацію та графіку, функціональні можливості компоненту.

ПЕРЕЛІК ПОСИЛАНЬ

1. Резіг Д. Секрети JavaScript ніндзя. Москва: Вільямс, 2016. 416 с.

2. Фленаган Д. JavaScript. Детальний посібник. Москва: Вільямс, 2013. 1080 с.

3. Express. URL : https://expressjs.com/ (дата звернення: 27.03.2018).

4. JavaScript info. URL : https://javascript.info/ (дата звернення: 21.03.2018).

5. Node.js documentation. URL : https://nodejs.org/dist/latest-v8.x/docs/api/ (дата звернення: 25.03.2018).

6. Three.js documentation. URL : https://threejs.org/docs/index.html#manual/introduction/Creating-a-scene (дата звернення: 24.03.2018).

7. WebGL Api. URL : https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API (дата звернення: 22.03.2018).

ДОДАТОК А

Код контролера візуалізації

```
import VisualizerHelper from './services/visualizerHelper.js';

export default class VisualizerController {
    constructor(jsonObject, settings) {
        this.jsonObject = jsonObject;
        this.settings = settings;
        this.visualizerHelper = new VisualizerHelper(jsonObject);
        this.getVisualizerArea();
        this.findCoordinatesArea();
        this.startCreatingScene();
        this.visualization();
        this.endCreatingScene();
        this.animate();
        this.setListeners();
        this.rotate = false;
    }

    startCreatingScene() {
        this.camera = new THREE.PerspectiveCamera(this.settings.fildView, window.innerWidth /
window.innerHeight, 0.1 , 1000);
        this.camera.position.z = 10;
        this.controls = new THREE.OrbitControls(this.camera);
        this.scene = new THREE.Scene();
    }

    visualization() {
        let geometry = this.jsonObject.countNodes == 3 ? this.visualizerHelper.createTriangleMesh() :
                                                this.visualizerHelper.createSquareMesh();
        geometry.computeFaceNormals();
```

```
        this.mesh = new THREE.Mesh(geometry, new
THREE.MeshNormalMaterial(this.settings.meshMaterialSettings));
        this.mesh.geometry.dynamic = true;
        this.mesh.geometry.verticesNeedUpdate = true;
        this.scene.add(this.mesh);
    }

    endCreatingScene() {
        this.renderer = new THREE.WebGLRenderer({antialias: true});
        this.renderer.setSize(this.settings.width, this.settings.height);
        this.visualizerArea.appendChild(this.renderer.domElement);
    }

    getVisualizerArea() {
        let visualizerAreas = document.getElementsByClassName('webgl-visualizer-drop-area');
        this.visualizerArea = visualizerAreas[0];
    }

    updateData(jsonObject, settings) {
        this.jsonObject = jsonObject;
        this.settings = settings;
        this.visualizerHelper.setJsonObject(this.jsonObject);
        this.updateView();
    }

    updateView() {
        this.scene = new THREE.Scene();
        let geometry = this.jsonObject.countNodes == 3 ? this.visualizerHelper.createTriangleMesh() :
                                        this.visualizerHelper.createSquareMesh();
        geometry.computeFaceNormals();

        this.mesh = new THREE.Mesh(geometry, new
THREE.MeshNormalMaterial(this.settings.meshMaterialSettings));
        this.scene.add(this.mesh);
```

```
}

animate() {
    requestAnimationFrame(this.animate.bind(this));
    this.mesh.position.z = -50;
    this.mesh.rotation.y = -Math.PI * .5;
    this.rotation();
    this.controls.update();
    this.renderer.render(this.scene, this.camera);
    this.setDisplayCoordinates(this.camera.position);
}

rotation() {
    if(this.rotate) {
        this.mesh.rotation.x += 0.05;
        this.mesh.rotation.y += 0.05;
    }
}

findCoordinatesArea() {
    this.coordinateX = document.getElementsByClassName('webgl-visualizer-coordinate-x')[0];
    this.coordinateY = document.getElementsByClassName('webgl-visualizer-coordinate-y')[0];
    this.coordinateZ = document.getElementsByClassName('webgl-visualizer-coordinate-z')[0];
}

setDisplayCoordinates(position) {
    this.coordinateX.innerHTML = position.x;
    this.coordinateY.innerHTML = position.y;
    this.coordinateZ.innerHTML = position.z;
}

setListeners() {
    let webglVisualizers = document.getElementsByTagName('webgl-visualizer');
    let webglVisualizer = webglVisualizers[0];
```

```
        webglVisualizer.addEventListener('changeSettings', (e) => {
            this.mesh.material[e.detail.name] = e.detail.status;
        }, false);

        webglVisualizer.addEventListener('rotationSettings', (e) => {
            this.rotate = e.detail.statusRotate;
        }, false);
    }
}
```

ДОДАТОК Б

Код візуалізатора геометрії

```
export default class VisualizerHelper {
    constructor(jsonObject) {
        this.jsonObject = jsonObject;
        this.geometry = new THREE.Geometry();
    }

    createTriangleMesh() {
        this.createVertices();

        for(let j = 0; j < this.jsonObject.figures.length; j++) {
            this.geometry.faces.push(new THREE.Face3(this.jsonObject.figures[j][0],
this.jsonObject.figures[j][1], this.jsonObject.figures[j][2]));
        }

        return this.geometry;
    }

    createSquareMesh() {
        this.createVertices();

        for(let j = 0; j < this.jsonObject.figures.length; j++) {
            this.geometry.faces.push(new THREE.Face3(this.jsonObject.figures[j][0],
this.jsonObject.figures[j][1], this.jsonObject.figures[j][2]));
            this.geometry.faces.push(new THREE.Face3(this.jsonObject.figures[j][0],
this.jsonObject.figures[j][2], this.jsonObject.figures[j][3]));
        }

        return this.geometry;
    }
```

```
    createVertices() {
        for(let i = 0; i < this.jsonObject.points.length; i++) {
            this.geometry.vertices.push(new THREE.Vector3(this.jsonObject.points[i].x,
this.jsonObject.points[i].y, this.jsonObject.points[i].z));
        }
    }

    setJsonObject(jsonObject) {
        this.geometry = new THREE.Geometry();
        this.jsonObject = jsonObject;
    }
}
```

ДОДАТОК В

Код конструювання компонента візуалізації

```
import VisualizerController from '../visualizerController';
import TypeConverter from './typeConverter';
import Settings from './settingsService';
import CreateElement from './createElementService.js';

export default class Helper {
    constructor(webglVisualizer) {
        let settings = new Settings(webglVisualizer);
        this.settingsObj = settings.initSettings();
        this.typeConverter = new TypeConverter();
        this.createService= new CreateElement();

        this.regexpJson = /.json$/i;
        this.regexpTxt = /.txt$/i;
        this.fileName = '';
    }

    createWebglVisualizer(url) {
        this.typeConverter.readDataTxt(url)
            .then((data) => this.visualizerCtr(data))
            .catch((err) => console.log('err ', err));
    }

    visualizerCtr(data) {
        if(!this.visualizer) {
            this.visualizer = new VisualizerController(data, this.settingsObj);
        } else {
            this.visualizer.updateData(data, this.settingsObj);
        }
```

```
    }

    loadChooser(result) {
        if(~this.fileName.search(this.regexpTxt)) {
            this.visualizerCtr(this.typeConverter.convertText(result));
        } else if(~this.fileName.search(this.regexpJson)) {
            this.visualizerCtr(JSON.parse(result));
        } else {
            alert("Type isn't supported !");
        }
    }

    createComponentFragment() {
        let fragment = document.createDocumentFragment();
        let input = this.createService.createElement({type: 'input', className: 'webgl-visualizer-input',
                                    marginRight: '5px', borderRadius: '5px', flexGrow: '6'});

        let btnUpload = this.createService.createElement({type: 'button', className: 'webgl-visualizer-
btn-upload',
                                        innerHTML: 'Upload', flexGrow: '2'});

        btnUpload.addEventListener('click', () => {
            let input = document.getElementsByClassName('webgl-visualizer-input')[0];
            let url = input.value;
            input.value = '';
            input.focus();

            if(url.length > 0) {
                this.createWebglVisualizer(url);
            } else {
                alert('Please type url');
            }
        });

        let btnOpen = this.createService.createElement({type: 'button', className: 'webgl-visualizer-
```

```
btn-open',
                                innerHTML: 'Open', flexGrow: '2'});

    btnOpen.addEventListener('click', (e) => {
        fileSelector.click();
        return false;
    });

    let fileSelector = document.createElement('input');
    fileSelector.setAttribute('type', 'file');

    fileSelector.addEventListener('change', (e) => {
        e.stopPropagation();
        e.preventDefault();

        let file = e.target.files[0];
        this.fileName = file.name;

        let reader = new FileReader();
        let context = this;

        reader.onload = function (event) {
            context.loadChooser(event.target.result);
        };
        reader.readAsText(file);
    });

    let urlRow = this.createService.createElement({type: 'div', className: 'webgl-visualizer-url-
row',
                                display: 'flex', marginBottom: '5px'});

    urlRow.appendChild(input);
    urlRow.appendChild(btnUpload);
    urlRow.appendChild(btnOpen);
```

```
let dropArea = this.createService.createElement({type: 'div', className: 'webgl-visualizer-drop-area',
                                position: 'relative', width: '100%', height: '100%',
                                border: '1px solid grey', borderRadius: '5px'});

dropArea.addEventListener('drop', (e) => {
    e.stopPropagation();
    e.preventDefault();

    let files = e.target.files || e.dataTransfer.files;

    if(files.length) {
        let file = files[0];
        this.fileName = file.name;

        let reader = new FileReader();
        let context = this;

        reader.onload = function (event) {
            context.loadChooser(event.target.result);
        };
        reader.readAsText(file);
    } else {
        alert('Problem with getting data');
    }
});

dropArea.addEventListener('change', (e) => {
    e.stopPropagation();
    e.preventDefault();

    let file = e.target.files[0];
    this.fileName = file.name;

    let reader = new FileReader();
```

```
        let context = this;

        reader.onload = function (event) {
            context.loadChooser(event.target.result);
        };
        reader.readAsText(file);
    });

    let coordinatesBlock = this.createService.createElement({type: 'div', className: 'webgl-visualizer-coordinates-block',
                                    position: 'absolute', width: '50px',
                                    height: '50px', padding: '10px', overflow: 'hidden'});

    let blockCoordX = this.createService.createElement({type: 'div', className: 'webgl-visualizer-block-coord-x',
                                display: 'flex'});

    let nameCoordX = this.createService.createElement({type: 'span', className: 'webgl-visualizer-name-coord-x',
                                width: '20px', color: '#FFF', fontSize: '14px', innerHTML: 'X: '});

    let coordinateX = this.createService.createElement({type: 'span', className: 'webgl-visualizer-coordinate-x',
                                color: '#FFF', fontSize: '14px', whiteSpace: 'nowrap',
                                overflow: 'hidden', textOverflow: 'ellipsis', minWidth: '50px'});

    blockCoordX.appendChild(nameCoordX);
    blockCoordX.appendChild(coordinateX);

    let blockCoordY = this.createService.createElement({type: 'div', className: 'webgl-visualizer-block-coord-y',
                                display: 'flex'});

    let nameCoordY = this.createService.createElement({type: 'span', className: 'webgl-visualizer-name-coord-y', width: '20px',
```

```
                                color: '#FFF', fontSize: '14px', innerHTML: 'Y: '});

    let coordinateY = this.createService.createElement({type: 'span', className: 'webgl-visualizer-
coordinate-y',
                                color: '#FFF', fontSize: '14px', whiteSpace: 'nowrap',
                                overflow: 'hidden', textOverflow: 'ellipsis', minWidth: '50px'});
    blockCoordY.appendChild(nameCoordY);
    blockCoordY.appendChild(coordinateY);

    let blockCoordZ = this.createService.createElement({type: 'div', className: 'webgl-visualizer-
block-coord-z',
                                display: 'flex'});

    let nameCoordZ = this.createService.createElement({type: 'span', className: 'webgl-
visualizer-name-coord-z',
                               width: '20px', color: '#FFF', fontSize: '14px', innerHTML: 'Z: '});

    let coordinateZ = this.createService.createElement({type: 'span', className: 'webgl-visualizer-
coordinate-z',
                                color: '#FFF', fontSize: '14px', whiteSpace: 'nowrap',
                                overflow: 'hidden', textOverflow: 'ellipsis', minWidth: '50px'});

    blockCoordZ.appendChild(nameCoordZ);
    blockCoordZ.appendChild(coordinateZ);

    coordinatesBlock.appendChild(blockCoordX);
    coordinatesBlock.appendChild(blockCoordY);
    coordinatesBlock.appendChild(blockCoordZ);

    dropArea.appendChild(coordinatesBlock);

    let visualizerBox = this.createService.createElement({type: 'div', className: 'visualizer-box',
margin: '5px',
                                  width: `${this.settingsObj.width}px`,
                                  height: `${this.settingsObj.height}px`});
```

```
        visualizerBox.appendChild(urlRow);
        visualizerBox.appendChild(dropArea);

        fragment.appendChild(visualizerBox);

        return fragment;
    }

    setWindowSettings() {
        window.addEventListener('drop', (e) => {
            e = e || event;
            e.preventDefault();
        }, false);

        window.addEventListener('dragover', (e)=> {
            e = e || event;
            e.preventDefault();
        }, false);
    }
}
```

ДОДАТОК Г

Код перетворювача типів

```
import SkeletonObject from './skeletonObject.js';

export default class TypeConverter {
   constructor() {}

   readDataTxt(url) {
      let contentType;
      return fetch(url)
         .then((response) => {
            contentType = response.headers.get("content-type");
            if(contentType === 'application/json; charset=UTF-8') {
               return response.json();
            } else if(contentType === 'text/plain; charset=UTF-8') {
               return response.text();
            }
         })
         .then((data) => {
            if(contentType === 'application/json; charset=UTF-8') {
               return data;
            } else if(contentType === 'text/plain; charset=UTF-8') {
               return this.convertText(data);
            }
         })
         .catch((err) => {
            console.log('Error of getting data ', err);
            return Promise.reject(err);
         });
   }
```

```
convertText(text) {
    this.skeletonObject = new SkeletonObject();

    let columnCount, index, pointCoordinatesData;
    let line = text.split('\n');
    let lineCount = line.length;
    let tmpLine = line[0].split(' ');

    this.skeletonObject.modelTypeD = tmpLine[0];
    this.skeletonObject.countNodes = tmpLine[1];
    this.skeletonObject.countFaces = tmpLine[2];
    this.skeletonObject.countPoints = line[1];

    for(index = 2; index < lineCount; index++) {
        tmpLine = line[index].split(' ');
        columnCount = tmpLine.length;

        if(columnCount === 4) {
            pointCoordinatesData = {
                x: tmpLine[0],
                y: tmpLine[1],
                z: tmpLine[2],
                type: tmpLine[3]
            };

            this.skeletonObject.points.push(pointCoordinatesData);
        } else if(columnCount === 2) {
            this.skeletonObject.countFigures = tmpLine[0];
            this.skeletonObject.mysticValue = tmpLine[1];
            index++;
            break;
        }
    }

    for(let i = index; i < lineCount; i++) {
```

```
            line[i] = line[i].trim();
            tmpLine = line[i].split(' ');

            if(tmpLine.length > 1) {
                this.skeletonObject.figures.push(tmpLine);
            }
        }

        return this.skeletonObject;
            }
}
```