КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет радіофізики, електроніки та комп'ютерних систем
Кафедра комп'ютерної інженерії

Інтегроване веб-середовище для роботи з параметризованими 3d-моделями

Дипломна робота магістра
студента 2 року навчання
Спеціальність: 123 «Комп'ютерна інженерія»
Владислава СОСНОВА

Науковий керівник
кандидат технічних наук Євген СЛЮСАР,
асистент кафедри комп'ютерної інженерії

Рецензент
кандидат фіз.-мат. наук,
доцент кафедри теорії та технології програмування
факультету комп'ютерних наук та кібернетики
Віктор ВОЛОХОВ

До захисту допускаю: Завідувач кафедрою

Юрій БОЙКО /

Ухвалено на засіданні кафедри "____"_____2022 р., протокол № ____

Київ 2022

РЕФЕРАТ

Обсяг роботи 81 сторінка, 31 ілюстрація, 1 таблиця, 17 джерел посилань, 5 додатків.

БАЗА ДАНИХ, ВЕБ-ЗАСТОСУВАННЯ, ІНТЕРФЕЙС КОРИСТУВАЧА, 3D-МОДЕЛЬ, GRASSHOPPER, HUMAN–COMPUTER INTERACTION, MODEL-VIEW-CONTROLLER, REACT.JS, RHINO3D, THREE.JS.

Об’єктом даної роботи є візуалізація та взаємодія у різних представленнях із 3d-моделями. Предметом роботи є програмний засіб для візуалізації та можливості зміни тривимірних об’єктів.

Метою роботи є розробка веб-додатку для відображення 3d-моделей з можливістю їх подальшої динамічної зміни, а також дослідження методик та етапів проектування для удосконалення інтерфейсу користувача.

Методи розроблення: метод ручного створення програмного продукту за допомогою платформи розробки веб-застосувань. Інструменти розроблення: безкоштовне, вільно поширюване інтегроване середовище розробки Visual Studio Code, мова програмування JavaScript, бібліотека Node.js для реалізації серверної частини, бібліотека React.js для створення інтерфейсу користувача.

Результати роботи: було проведено аналіз представлення об’єктів та етапи створення розділів, що дозволило впровадити новий дизайн додатку, базуючись на критеріях якості побудови інтерфейсів користувача. Розроблено та удосконалено інтегровану систему, до якої входить веб-сервіс для взаємодії з системами для створення 3d-об’єктів та односторінковий веб-інтерфейс, який дозволяє користувачам взаємодіяти з кількома тривимірними об’єктами одночасно у різних представленнях.

ЗМІСТ

С.

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

ІК – Інтерфейс користувача

HCI – Взаємодія людини і комп'ютера

GUI – Графічний інтерфейс користувача

UI – Користувацький інтерфейс (user interface)

UX – Користувацький досвід (user experience)

CAD – Система автоматизованого проектування і розрахунку

ПЗ – Програмне забезпечення

БД – База даних

MVC – Model View Controller

GLTF – GL Transmission Format

JS – JavaScript

CSS – Cascading Style Sheets

HTML – Hypertext Markup Language

HTTP – Hypertext Transporting Protocol

ВСТУП

Немає ніякого сумніву у тому, що за останнє сторіччя людство зробило великий крок вперед у створенні та впровадженні нових технологій. Зокрема, люди почали взаємодіяти з різноманітними електронними пристроями, такими як: персональні комп’ютери, ноутбуки, телефони, і задля того, щоб користувачеві було зрозуміло та зручно взаємодіяти з ними, інтерфейс має вирішальне значення. Настільні програми, Інтернет-браузери, та мобільні телефони використовують сучасні графічні інтерфейси користувачів (GUI).

З розвитком все нових та кращих технологій, замість розробки регулярних інтерфейсів, різні дослідницькі галузі мали по-різному зосереджуватися на концепціях мультимодальності, а не унімодальності, інтелектуальних адаптивних інтерфейсах, а не на основі команд / дій у терміналах, і нарешті на активних, а не на пасивних інтерфейсах.

У галузі взаємодії людини та комп'ютера (HCI), яка вивчає дизайн та використання комп'ютерних технологій, орієнтованих на інтерфейси між людьми та комп'ютерам важливим аспектом є задоволеність користувачів. “Оскільки взаємодія людини з комп'ютером вивчає спілкування людини і машини, воно спирається на підтримку знань як на машині, так і на людях. З боку машини актуальні методи комп'ютерної графіки, мов програмування і середовищ розробки. З людського боку актуальні теорія комунікації, дисципліни графічного і промислового дизайну, соціальні науки, когнітивна психологія, а також людські фактори, такі як задоволеність користувачів комп'ютера” [1]. Завдяки міждисциплінарному характеру даної галузі, люди з різною освітою та кваліфікацією сприяють її успіху.

Окрім цього, комп’ютерні 3d моделі також стали важливою частиною нашого повсякденного життя. Це, насамперед, пов’язано з безперервно зростаючими обсягами завдань, які треба вирішувати у багатьох галузях, починаючи від природничих наук та медицини, закінчуючи сферами дизайну, архітектури та мистецтва. Використання 3d-моделей значно покращує створення різного роду прототипів та об’єктів затрачуючи при цьому менше часу та коштів на реалізацію.

У сучасному світі підвищуються вимоги якості та збільшується потреба автоматизації процесів. Часом, архітектору чи дизайнеру треба створити декілька прототипів будівель, структур та навіть груп об’єктів за відносно невеликий проміжок часу. Для того щоб не робити кожен об’єкт з самого початку у відносно складних програмах та системах, було вирішено розробити та покращити проект designgallery, який дозволяє динамічно змінювати різні параметри об’єктів у браузері із забезпеченням надійності даних та впровадженою аутентифікацією. Більш того, завдяки оновленому дизайну, враховуючи критерії побудови інтерфейсу користувача, з ним зможуть взаємодіяти користувачі, у яких немає ніякого досвіду роботи з параметризованими 3d-моделями та комп’ютерним моделюванням в цілому.

РОЗДІЛ 1. ОГЛЯД ПРИНЦИПІВ ПОБУДОВИ ІНТЕРФЕЙСІВ КОРИСТУВАЧА

1.1 Взаємодія людини з комп'ютером

Термін HCI доволі часто використовується, якщо тема про наукові основи проектування інтерфейсів користувача. Human-Computer Interaction або "взаємодія людини і комп'ютера", спочатку використовували як спеціальність у галузі інформатики, що охоплювала когнітивну науку та інженерію людських факторів [8]. HCI у значній мірі зараз об'єднує колекцію досліджень і практик у візуалізації, інформаційних системах, системах спільної роботи, процесах розробки та в інших галузях орієнтованих на людину.[6]

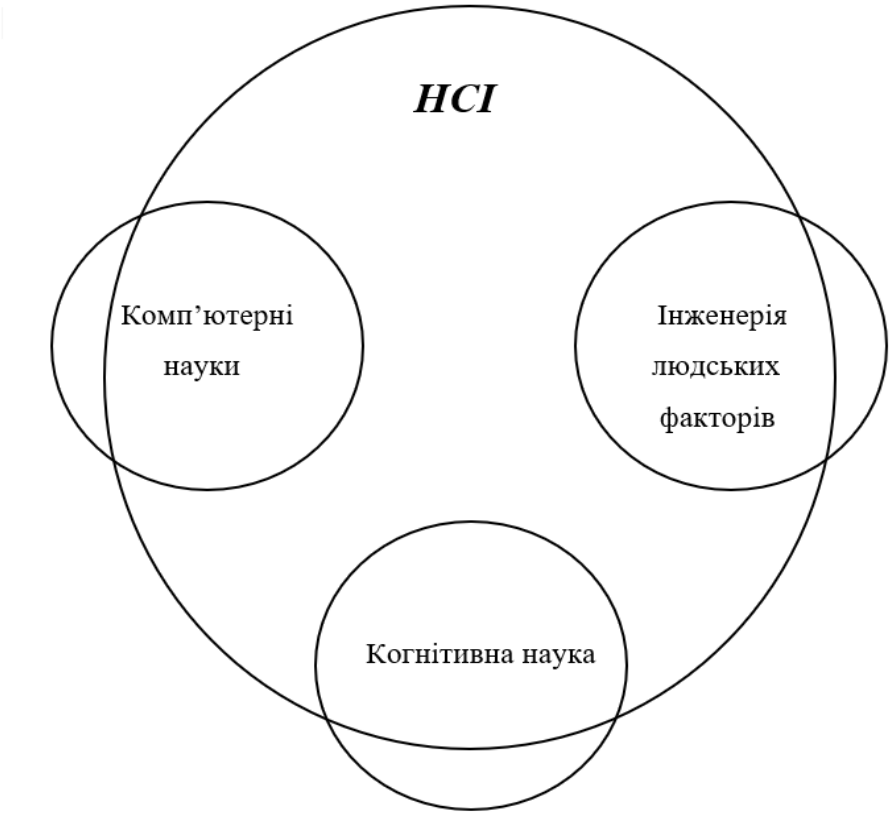

Рисунок 1 – Мультидисциплінарна область HCI

HCI стала предметом інтенсивних наукових досліджень. Ті, хто вивчав і працював у даній сфері, розглядали це як найважливіший інструмент для популяризації ідеї, що взаємодія між комп'ютером і користувачем повинна нагадувати відкритий діалог між людьми.

Автор та засновник галузі Human-computer interaction Джон М. Керрол говорив:

“Більше не має сенсу розглядати HCI як спеціальність інформатики; HCI зросла до більш ширшої, масштабнішої та набагато різноманітнішої галузі, ніж ніж сама комп'ютерна наука. HCI розширила свою первісну орієнтацію на індивідуальну і загальну поведінку користувачів, включивши в неї соціальні та організаційні обчислення, доступність для людей з обмеженими розумовими і фізичними можливостями та для всіх людей, а також для найширшого спектра людського досвіду і діяльності. Він розширився від ранніх графічних користувальницьких інтерфейсів і включив в себе безліч методів і механізмів взаємодії, мультимодальні взаємодії і безліч з'являються повсюдних, портативних і контекстно-залежних взаємодій.”[4]

1.1.1 Значення HCI для UX та пов'язаних областей

HCI - це широка область, яка перетинається з такими областями, як дизайн, орієнтований на користувача (UCD), дизайн інтерфейсу користувача (UI) та дизайн досвіду користувача (UX).

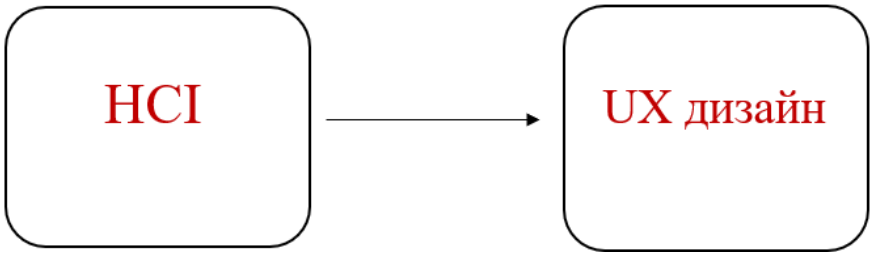

Рисунок 2 – Еволюція UX дизайну

UX (User experience) - дослівно означає «досвід користувача». У більш широкому сенсі це поняття включає в себе весь досвід, який отримує користувач при взаємодії із сайтом або додатком. UX-дизайн відповідає за функціональність, адаптивність продукту і те, які емоції він викликає у користувачів. Чим простіше інтерфейс, тим простіше користувачеві отримати результат і зробити цільову дію.

UI (User interface) перекладається як «призначений для користувача інтерфейс». Він може бути не тільки графічним, а й тактильним, голосовим, аудіо. UI-дизайнер відповідає за те, як виглядає інтерфейс продукту та як користувач взаємодіє з його елементами. Для цього необхідно організувати елементи інтерфейсу та витримати єдиний стиль та логіку їх взаємодії.

Дизайн користувальницького інтерфейсу, який часто плутають з UX-дизайном, більше стосується поверхні і загального відчуття дизайну, тоді як останній охоплює весь спектр користувацького досвіду. Однією з аналогій є уявлення дизайну UX як транспортного засобу з дизайном інтерфейсу в якості пульта управління.

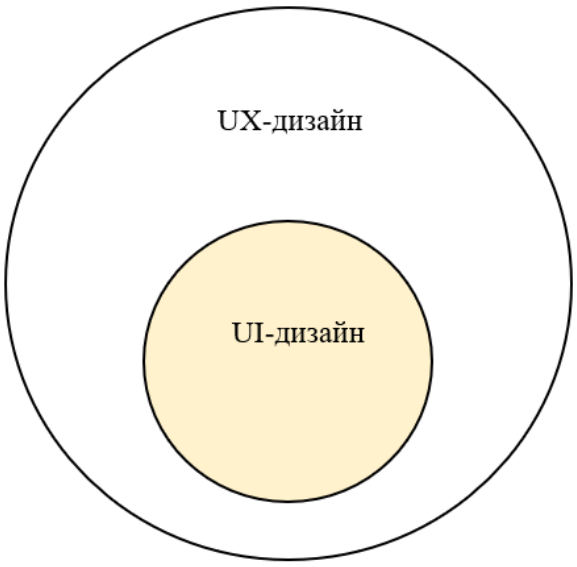

Рисунок 3 – Співвідношення UI/UX дизайну

Незважаючи на те що, багато в чому HCI був попередником UX-дизайну, деякі відмінності все ж залишаються між ними. Практики HCI, як правило, більш академічні. Вони беруть участь в наукових дослідженнях і розробці емпіричних уявлень користувачів. І навпаки, дизайнери UX майже завжди орієнтовані на промисловість і беруть участь в створенні продуктів або послуг.

1.2 Критерії якості побудови інтерфейсів користувача

Інтерфейс користувача (ІК) - це сукупність засобів, використовуючи які користувач взаємодіє з різними системами, пристроями та іншим складним інструментарієм. Національний банк стандартизованих науково-технічних термінів дає таке визначення інтерфейсу користувача – це комплекс програмних та апаратних засобів, які забезпечують взаємодію користувача з комп'ютером [2].

ІК поєднує компоненти та елементи програмного забезпечення, які можуть впливати на взаємодію користувача при роботі з ПЗ (рис. 4).

Мета створення інтерфейсу користувача - зробити ІК ефективним і приємним (зручним), який спрощує роботу та забезпечує бажаний результат.

В галузі дизайну та розробки інтерфейсів з'явився такий термін як "зручний для користувача інтерфейс". Принципи хорошого інтерфейсу однакові і для веб-сайтів, і для програм. Досконалі інтерфейси повинні:

- бути мінімалістичним;
- показувати всі важливі опції;
- давати можливість персоналізації;
- надавати варіанти зручного управління.
- надавати оптимальну кількість варіантів вибору;
- уникати двозначності та бути інтуїтивно знайомим для користувача;

– надавати хороший користувальницький досвід (або User experience, UX).
– мати принципи локалізації – відображення інформації різними мовами;

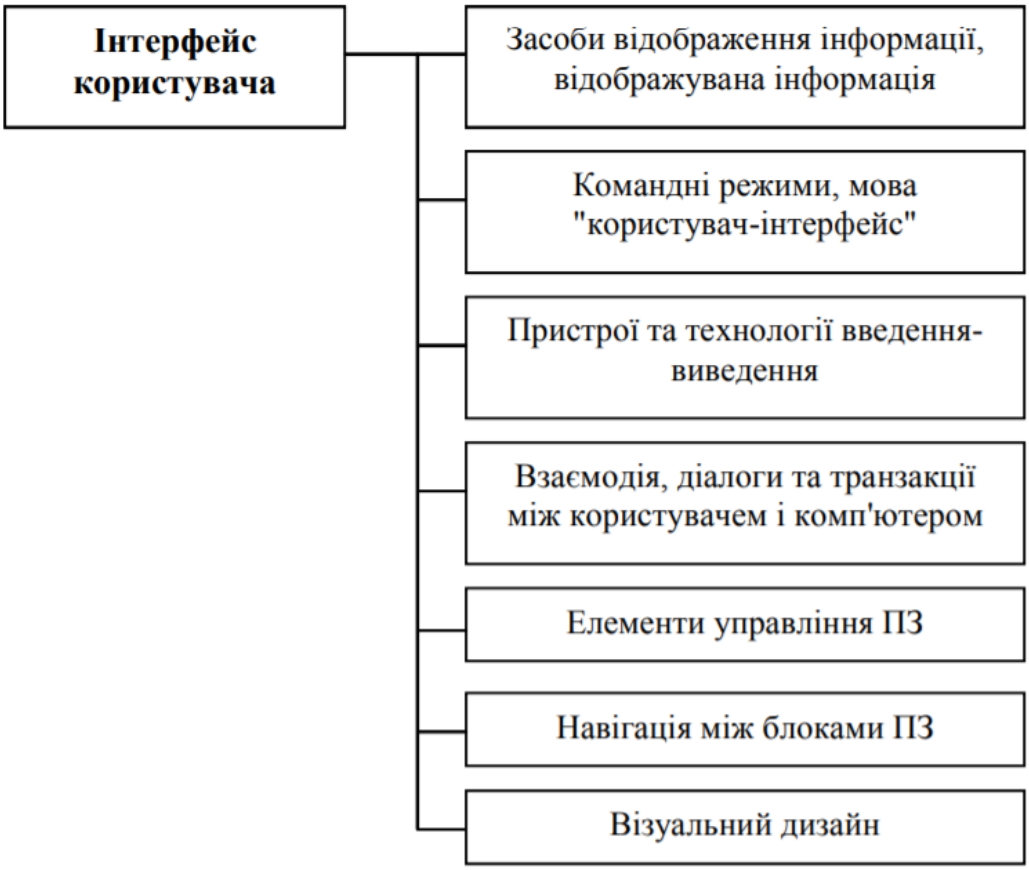

Рисунок 4 – Складові інтерфейсу користувача

Одним із видів користувацьких інтерфейсів є веб-інтерфейс. Веб-інтерфейси приймають вхідні дані та забезпечують виведення інформації за допомогою створення веб-сторінок, які переглядаються користувачем за допомогою веб-браузера. У сучасній розробці веб-ресурсів, використовуються: CSS3, HTML5, JavaScript та інші подібні технології. Адміністративні веб-інтерфейси для мережевих комп'ютерів і веб-серверів, доволі часто називають панелями керування.

1.2.1 “Золоті правила” Бена Шнейдермана

Вісім “золотих правил” Бена Шнейдермана вважають гарним методом створення зручних та продуктивних інтерфейсів користувача. Цими правилами користуються такі корпоративні гіганти як Google, Apple та Microsoft, що свідчить про їх якість.

Ці “правила” мають саме такий вигляд :

Швидка відміна виконаних дій. Інтерфейс повинен пропонувати користувачам очевидні способи для зміни своїх дії. Ці зміни повинні бути дозволені в різних точках, незалежно від того, чи відбувається це після однієї або певної послідовності дій.

Прагнення до послідовності. Інтерфейс повинен використовувати знайомі іконки, кольори, ієрархію меню, заклик до дій і призначені для користувача сценарії при розробці аналогічних ситуацій. Стандартизація способу передачі інформації гарантує, що користувачі можуть застосовувати знання від одного кліка до іншого; без необхідності вивчати нові уявлення для тих же дій.

Ініціація дій користувачем. Система має дозволяти користувачам бути ініціаторами дій та надати їм відчуття повного контролю подій, що відбуваються в цифровому просторі. Для цього необхідно спроектувати систему, яка буде вести себе так, як вони очікують.

Підтвердження виконаних дій. Система не має змушувати користувачів вгадувати, що відбулось, а навпаки інформувати до чого їх дії призвели. Наприклад, користувачі будуть вдячні за повідомлення «Запис успішно доданий в систему» і підтвердження після збереження дій.

Просте опрацювання помилок. Система повинна бути спроектована так, щоб бути максимально надійною. Але при виникненні неминучих помилок користувачам маю надаватися прості, інтуїтивно зрозумілі покрокові інструкції, що дозволяють вирішити проблему максимально швидко і безболісно. Хорошим прикладом є позначення обов’язкових текстових полів, в яких користувач забув ввести дані в онлайн-формі.

Зменшення навантаження на короткочасну пам’ять. Інтерфейси повинні бути максимально простими, з правильною інформаційної ієрархією і вибором впізнавання, а не згадування. Розпізнати що-небудь завжди легше, ніж згадати, тому що розпізнавання включає в себе сприйняття сигналів, які допомагають нам проникнути в нашу пам'ять і дозволяють спливати відповідній інформації.

Інформативний зворотній зв’язок. Користувач завжди повинен знати, де він знаходиться і що відбувається. Протягом розумного періоду часу для кожної дії повинна бути відповідна, легка для читання відповідь. Хорошим прикладом застосування цього, було б вказати користувачеві, де він знаходиться в процесі при роботі з багатосторінковим опитувальником, наприклад Google Forms тощо. Поганий приклад, який ми часто бачимо, - повідомлення про помилку показує код помилки замість зрозумілого і значимого повідомлення.

Можливість “короткого шляху”. Інтерфейс повинен надавати користувачам більш швидкі методи виконання завдань. Наприклад, як Windows, так і Mac надають користувачам поєднання клавіш для копіювання та вставки, так само й у міру того, як користувач стає більш досвідченим, він повинен мати змогу переміщатися і керувати призначеним для користувача інтерфейсом швидше і без зусиль.

1.3 Застосування аутентифікації

Сучасне суспільство неможливо уявити без різноманітних гаджетів та комп'ютерів. Технології проникли у наше життя та міцно закріпилися у всіх сферах життєдіяльності людей. Відповідно, з розвитком інформаційно-комунікативних технологій змінюються вимоги до безпеки даних, а отже й потреба в забезпеченні надійності персональних даних набуває все більшої значущості. Це, насамперед, пов'язано із тими безперервно зростаючими обсягами даних, якими оперують щодня.

Аутентифікація – це акт підтвердження особистості користувача комп'ютерної системи. Це може передбачати перевірку особистих документів, що підтверджують особу, перевірку автентичності веб-сайту за допомогою цифрового сертифіката або гарантування того, що товар чи документ не є підробленими [7].

Вибираючи тип аутентифікації, необхідно враховувати UX разом із безпекою. Деякі типи аутентифікації користувачів менш безпечні, ніж інші, але надмірна перевірка під час аутентифікації може призвести до не зовсім правильної роботи застосунку.

1.4 Аналіз існуючих моделей автентифікації

Аутентифікація актуальна для багатьох сфер, в часності, у сфері інформатики перевірка даних користувача необхідна для надання доступу до конфіденційної інформації.

I. Аутентифікація на основі пароля

Аутентифікація на основі пароля покладається на ім'я користувача та пароль або PIN-код – є найпоширенішим методом аутентифікації.

Аутентифікація на основі пароля — це найпростіший тип автентифікації, яким зловмисники можуть зловживати. Люди часто повторно використовують паролі та створюють паролі, які можна вгадати, зі словами словника та загальнодоступною особистою інформацією.

II. Двофакторна/багатофакторна аутентифікація

Двофакторна автентифікація (2FA) вимагає від користувачів надати принаймні один додатковий фактор аутентифікації, крім пароля. Додатковими факторами можуть бути будь-який із типів автентифікації користувача, або одноразовий пароль, надісланий користувачеві за допомогою тексту або електронної пошти. Цей тип аутентифікації посилює безпеку облікових записів, оскільки для доступу зловмисникам потрібні не тільки облікові дані.

Практично аутентифікацію можна розділити на такі типи:

- HTTP Authentication

Протокол передачі гіпертексту (HTTP), який зазвичай використовується для створення веб-додатків, забезпечує 2 форми автентифікації – основну (basic) та дайджест (digest).

a. HTTP Basic Authentication

Базова автентифікація HTTP - якщо браузер або програма надсилає запит на веб-сторінку, яка вимагає базової автентифікації, сервер відповідає помилкою, яка містить у заголовку атрибут «WWW-автентифікація». Потім користувач вводить ім’я користувача та пароль, які надсилаються на сервер у кодованій формі Base64.

Переваги:

– Це дуже легкий механізм автентифікації.
– Його можна ефективно використовувати в поєднанні з SSL.
– Більшість веб-серверів та платформ надають вбудовану підтримку; таким чином роблячи реалізацію дуже простою.

Недоліки:

– Base64 - це не техніка шифрування, а лише проста форма кодування.
– Можна легко перехопити і розшифрувати.
– Базова автентифікація використовує текстові файли для зберігання імен користувачів та паролів.
– В ОС потрібно створити індивідуальні облікові записи користувачів.

b. HTTP-дайджест-аутентифікація

HTTP Digest Authentication працює аналогічно Basic, але є сильнішою, оскільки використовує "хеші", надсилаючи ім'я користувача та пароль на сервер. Коли надсилається запит на веб-сторінку, сервер надсилає відповідь із атрибутом в заголовку та "nonce".

"Nonce" - це рядок, який відрізняється для кожного запиту. Клієнт використовує серію хешів, які включають ім'я користувача та пароль, запитувану URL-адресу, ім'я та nonce області автентифікації та надсилає запит знову. Сервер вибирає пароль зі свого джерела даних і знову проходить той самий процес хешування та порівнює результати.

Переваги:

– Пароль хешується з динамічним значенням nonce; тим самим захищаючи його при передачі та від атак повтору.

– Сервер також може зберігати хеш пароля разом із nonce; тим самим запобігаючи атакам веселкових таблиць.
– Більшість веб-серверів та платформ надають вбудовану підтримку

Недоліки:

– Пароль повинен зберігатися у текстовому файлі.
– В ОС потрібно створити індивідуальні облікові записи користувачів.
– Дайджест-автентифікація також не автентифікує ідентифікацію сервера

- Form-based Authentication

Аутентифікація на основі форми надає розробнику свободу побудувати більш безпечну схему автентифікації. Дана аутентифікація відноситься до будь-якого механізму, який спирається на фактори, зовнішні від протоколу HTTP, для автентифікації користувача. Програмі залишається вирішувати питання отримання облікових даних користувача та їх перевірки.

Переваги:

– Розробник може вільно впровадити сторінку входу бажаним чином.
– Усі фреймворки та мови підтримують аутентифікацію форми.

Недоліки:

– Шифрування або безпека не застосовуються за замовчуванням. Відповідальність за впровадження безпечного рішення належить розробнику.

- Одноразові паролі (OTP)

Одноразові паролі є формою двофакторної автентифікації. Вони імітують обмін секретними повідомленнями на льоту між двома цифровими об'єктами, використовуючи модель зовнішнього зв'язку (SMS, електронну пошту або паперові паролі)[7].

Користувач надає своє ім’я користувача та / або пароль під час процесу автентифікації. Сервер перевіряє ім’я користувача та генерує OTP, який надсилається на позасмуговий комунікаційний носій (SMS, електронна пошта тощо). У деяких випадках попередньо сформований набір OTP генерується на папері та фізично доставляється користувачеві вручну або поштою.

Переваги:

– Аутентифікація залежить від секретів двох різних систем.
– Тривалість дії OTP обмежує постійну компрометацію облікового запису користувача.
– OTP можна використовувати лише один раз для конкретної транзакції.

Недоліки:

– Схема залежить від наявності додаткового серверу та зовнішньої інфраструктури (SMS / електронна пошта).
– Доставка пароля може бути затримкою, що не залежить від програми.

У рамках дипломної роботи поставлені завдання передбачають впровадження аунтентифікації на основі форми, використовуючи стратегію для веб-серверу.

РОЗДІЛ 2. ОГЛЯД ТЕХНОЛОГІЙ 3D-МОДЕЛЮВАННЯ

2.1 Застосування 3D-графіки

Технології тривимірного моделювання проникли у наше життя та міцно закріпилися у всіх сферах життєдіяльності людей, особливо у творчих професіях - дизайні та архітектурі. До появи 3D моделювання люди розробляли майбутню форму виробу лише за малюнком чи кресленням.

3D-модель - це математичне зображення будь-якого тривимірного об'єкта; модель технічно не є графічною, поки вона не відображається. Вона може бути представлена у вигляді програмного коду або за допомогою двовимірного зображення, що створюється за допомогою процесу рендерингу.

Рендеринг (rendering), комп'ютерна візуалізація — це процес отримання зображення за моделлю з допомогою комп'ютерної програми. Модель являє собою опис тривимірних об'єктів у вигляді структури даних, що містить геометричні дані, інформацію про освітлення і положення точки спостерігача.

3D-об'єкти можна генерувати автоматично або створювати вручну, деформуючи сітку (mesh) або іншим чином маніпулюючи вершинами (vertices).

Основне завдання 3D моделювання - моделювання створювальної форми об’єкту чи предмету. Для подальшої розробки проекту в дизайні чи архітектурі потрібно чітко розуміти найдрібніші його конструктивні особливості.

Комп’ютерне моделювання дає змогу досягти максимально реалістичної розробки надаючи відчуття ефекту присутності.

2.2 Основні принципи 3D моделювання

Основні принципи 3D моделювання обумовлюють підбір способів моделювання в різних проектних ситуаціях, знаходити різні способи втілення

ідей та щоразу вдосконалювати новітні підходи до комп'ютерного моделювання як способу проектування [2]. В дизайн-проектуванні різні види 3D моделювання застосовуються в синтезі (табл. 1), що дозволяє сформувати індивідуальний підхід до створення оригінальної форми об'єктів та виробів.

Види моделювання	**Опис, елементи побудови**	**Програми**	**Модель**
Параметричне	Моделювання по набору заданих змінних параметрів операцій	CATIA	
Не параметричне	Моделювання без збереження параметрів побудови (історії побудови)	Rhinoceros	
Комбіноване	Історію побудови в будьякий момент можна видалити / відключити	Alias, Rhinoceros + Grasshopper	
Полігональне	Полігон, крива, (poly, nurbs, mesh)	Alias, 3ds Max, Maya, Rhino	Полігональна
Каркасне моделювання	Точка і лінія (line & point)	Alias ST, AutoCAD, CATIA	Каркасна
Поверхневе моделювання	Точка, лінія, поверхня (surface)	Alias ST, AutoCAD, CATIA	Поверхнева

Твердотільне моделювання	Solid (тверде тіло)	Alias ST, AutoCAD, CATIA	Твердотільна
Кінцево - елементне моделювання	Вузол, кінцевий елемент, сітка (point, lines, mesh)	Ansys, Rhino + Kangaroo	Кінцевоелементна
Генеративне моделювання	Компоненти, зв'язку між компонентами	Rhino + Grasshopper	Генеративна

На основі вищезазначених видів моделювання виявлено чотири принципи моделювання: традиційний, інверсійний, генеративний, інтерактивний.

Традиційний принципу включає 4 етапи (рис. 5): творче (Sketch modeling), або полігональне, моделювання (Polygonal and mesh modeling) —> поверхневе моделювання(Surface modeling) —> твердотільне моделювання (Solid modeling) —> прототипування (Prototyping).

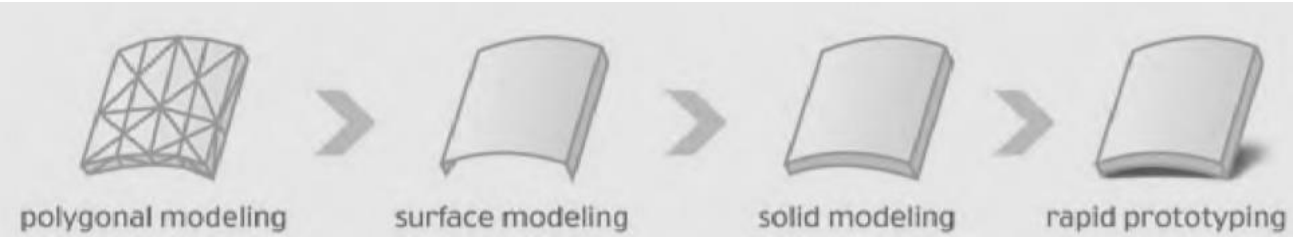

Рисунок 5 – Алгоритм традиційного принципу

Особливості традиційного принципу - зменшення часу на виготовлення прототипу, створення інформаційної моделі об’єкту з її подальшим використанням за допомогою поліскладних поверхонь класу:

– «A» - для видових, глянцевих поверхонь складної форми так як при візуалізації мають ледь помітні стики;
– «B» - в промисловому дизайні, проте при візуалізації отримують менш плавні відблиски;
– «C» - при візуалізації утворюються відблиски ламаної форми, застосовуються в малозначних місцях.

Знаючи традиційний принцип, в дизайнера є можливість реалізувати концепції з використанням 3D моделювання, дослідження формоутворення та прототипування об’єкту. Це найпоширеніший спосіб завдяки якому можна створити за короткий час різні форми за кресленнями, електронними та створеними вручну ескізами. Підходить для створення концепт-артів, авто, промислових виробів з різноманітною за складністю геометрією форми.

Інверсійний (зворотній) принцип (рис. 6): традиційне макетування (Modeling) —> об'ємне сканування (3D scaning) —> поверхневе моделювання (Surface modeling) —> твердотільне моделювання (Solid modeling) —> прототипування (Prototyping).

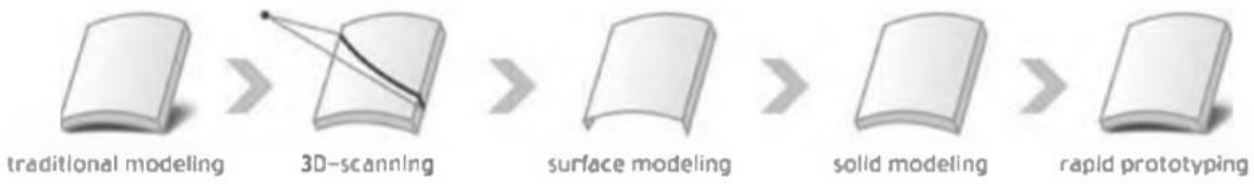

Рисунок 6 – Алгоритм інверсійного принципу

Інверсійний принцип подібний по етапах до традиційного. Проте має один нюанс. Для створення 3D моделі використовується дані котрі отримано за допомогою 3D сканування макету виготовленого власноруч. Даний принцип користується популярністю серед студентів та дизайн-студій котрі

використовують 3D сканери. Отримані, за допомогою сканування, дані лягають в основу максимально точного моделювання об’єкту.

Даний принцип використовується для моделювання об’єктів на заключних стадіях чи для ресайклінгу існуючої форми, а також для об’єктів з частково уніфікованими формами. Поверхневе моделювання відіграє значну роль в інверсійному принципі [5].

Генеративний принцип: (рис. 7): інформаційне моделювання (Information modeling) —> геометричне моделювання (Geometry modeling) —> прототипування (Prototyping).

Рисунок 7 – Алгоритм генеративного принципу

Велику популярність здобув серед дизайнерів та архітекторів і широко використовується при розробці генеративної та параметричної архітектури, інтерактивному та виставковому дизайні. Даний принцип актуальний при необхідності внесення змін до параметрів предметної системи чи при використанні системи об’єктів зі складною структурою, котрі не мають повторювальних елементів. Генеративне моделювання поєднало в собі геометричне та інформаційне моделювання. Інформаційне моделювання створює суть концепції об’єкту. Реалізується вона за допомогою ПО Rhino + Grasshopper.

Генеративна модель легко адаптується до різного роду виробництва чи 3D друку завдяки багатьом параметрам котрі можна редагувати в реальному часі. Це,

в свою чергу, пришвидшує процес та робить його значно ефективнішим, скорочує термін моделювання об’єктів, що складаються з великої кількості елементів. Надає можливість редагування параметрів вже змодельованої, геометричної моделі, що надає змогу підлаштовувати предмет під навколишнє середовище, оптимізувати налаштування, дублювати об’єкт з подібними рисами але вже з різними пропорціями чи параметрами.

Інтерактивний принцип (рис. 8): генеративне моделювання (Generative modeling) —> декодування інформаційної моделі в плагіні (Decoding) —> відкрита платформа для прототипування (Interactive prototyping).

Рисунок 8 – Алгоритм інтерактивного принципу

Інтерактивний принцип оснований на поєднанні генеративної моделі з платформою для прототипування. Процес інтеграції відбувається за допомогою декодування інформації моделі за допомогою плагіна. Завдяки подібним платформам є можливість створювати електронні системи (сенсори), а для взаємодії з навколишнім середовищем сервоприводи тощо. Даний спосіб надає можливість розробки генеративних моделей, що можуть взаємодіяти з природнім оточенням та людиною за допомогою сенсорних датчиків.

2.2.1 Класифікації комп'ютерного моделювання

Комп’ютерне моделювання – це використання комп'ютерів для моделювання та вивчення складних систем з використанням математики, фізики та інформатики. Обчислювальна модель містить численні змінні, що

характеризують систему, що вивчається. Моделювання проводиться шляхом коригування змінних самостійно або в поєднанні та дотримання результатів.

Комп’ютерне моделювання полягає в отриманні кількісних і якісних результатів з комп’ютерної або фізичної моделі.

Використання комп’ютерного моделювання має деякі переваги:

– дане моделювання є менш ресурсозатратним у порівнянні з експериментальними дослідженнями на обладнанні у лабораторіях;

– відтворення процесів дає змогу наблизитися до реальних умов;

– можливість багаторазового повторення експерименту [3, 20].

Загалом, комп’ютерне моделювання можна класифікувати як: структурно-функціональне, концептуальне, імітаційне, інформаційне та математичне (рис. 9)

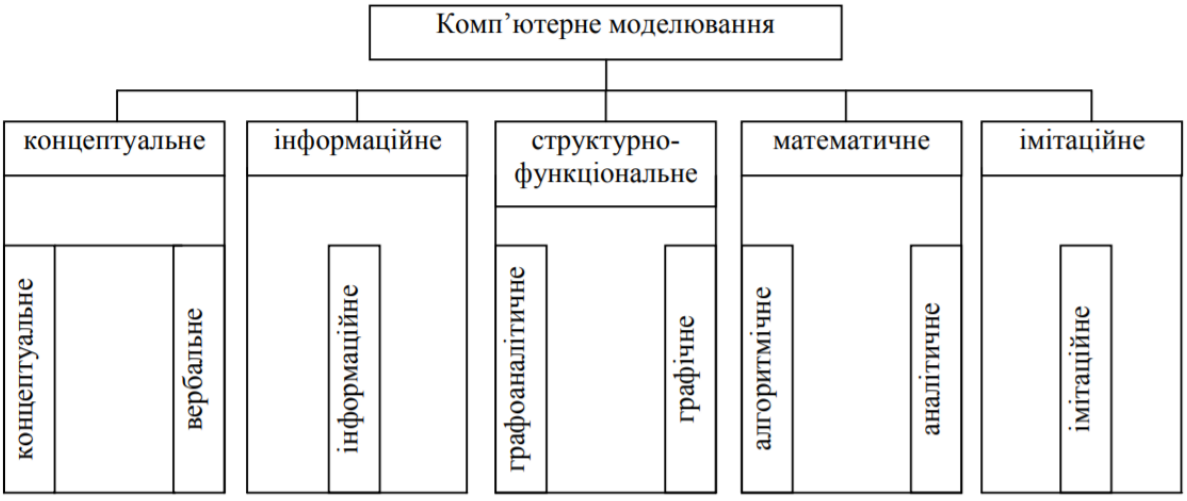

Рисунок 9 – Види комп’ютерного моделювання

Концептуальне моделювання – представлення системи за допомогою спеціальних символів, знаків, операцій над ними.

Основою інформаційного комп’ютерного моделювання є інформаційні моделі, які є різноманітними та багаточисельними за характером завдань (наприклад, бази даних, інформаційно-пошукові системи). Відносно нескладні алгоритми є загальною особливістю інформаційних моделей – пошук даних за деякими ознаками, актуалізація інформації, сортування даних тощо.

Структурно-функціональне комп’ютерне моделювання – умовний образ оригінала-об’єкта або деякої системи процесів чи об’єктів, описаних за допомогою взаємопов’язаних блок-схем, графіків, комп’ютерних таблиць, діаграм, тощо. До даного виду моделювання відносять графоаналітичні та графічні моделі.

Графоаналітичні моделі включають в себе усі можливі різновидності графіків та геометричних конструкцій, інтерпретації аналітичних залежностей, тощо. Ці моделі в основному застосовуються як інструмент для формування робочої моделі – аналітичної, алгоритмічної або імітаційної.

Графічний тип моделей створюється засобами комп’ютерної графіки. Абстрактний характер усіх структурних компонентів моделі, які представляють собою формально описані елементи, що визначають допустимі дії над даними елементами і порядок їх виконання є загальними ознаками графічних моделей.

Математичне (логіко-математичне) моделювання – побудова моделі здійснюється засобами математики і логіки;

Під імітаційним моделювання мається на увазі спеціальна форма ідеальної математичної моделі, за допомогою якої досліджуються алгоритми функціонування системи та зовнішні впливи. Наявні математичні методи числового та аналітичного рішення не використовуються з цими алгоритмами,

однак дозволяють здійснювати вимірювання необхідних характеристик завдяки імітуванню процесу функціонування системи.

Зазначені види моделювання можуть застосовуватися самостійно або одночасно, в деякій комбінації (наприклад, в імітаційному моделюванні використовуються практично всі перераховані види моделювання або окремі прийоми).

2.2.2 Параметризовані моделі та їх застосування

Модельована система може містити певні відносно ізольовані підсистеми, які базуються на основі параметрів . Так, параметричне моделювання дає можливість скоротити обсяг і тривалість (час) побудови структури.

Параметричне моделювання – це проектування моделі з використанням параметрів та співвідношень між ними елементів даного об’єкта. Параметризація дає можливість за корокий проміжок часу впровадити зміни параметрів моделі, випробувати різні комбінації геометричного співвідношення, внести відповідні корективи та уникнути подальших помилок.

Необхідність в моделюванні виникає на багатьох етапах створення та підтримки систем. Для прикладу, на етапі проектування – для оцінювання правильності прийнятих рішень, на етапі експлуатації – для оцінювання наслідків внесення змін в систему.

У рамках дипломної роботи поставлені завдання передбачають візуалізацію параметризованих 3d моделей у веб-застосунку, використовуючи імітаційне (програмне) моделювання та алгоритм генеративного принципу.

2.3 Аналіз існуючих редакторів параметризованих 3D-моделей

Область візуалізації даних дуже широка і постійно розширюється з року в рік. Згідно з дослідженням Global Market Insights [17], обсяг ринку 3D-моделювання збільшиться на 20% в період з 2019 по 2025 рік у порівнянні з 2018 роком, який оцінюється у більш ніж в 1,5 мільярда доларів США. Попит на технологію 3D-моделювання посилюється, оскільки розробляти детальні моделі об’єктів, пейзажі, будівлі тощо з можливістю багаторазового використання.

Зростаюче поширення тривімірного моделювання в секторі будівництва та нерухомості також грає важливу роль в прискоренні зростання ринку. Типова 3D-архітектурна модель включає в себе зображення та дані, які допомагають архітектору домогтися своїх вимог щодо матеріалів, тим самим оцінюючи приблизну вартість будівлі. Також, це допомагає знайти потенційні недоліки та лазівки в будівлі заздалегідь до її будівництва, тим самим допомагаючи врятувати кошти, час та головне життя людей.

2.3.1. Порівняльна характеристика існуючих засобів

Лідери ринку у сфері візуалізації даних, в часності 3d моделей, повинні орієнтуватися на користувачів, пропонуючи продукти, які можуть вирішити ряд нагальних проблем. Однак не всі компанії можуть надати відповідні можливості, необхідні у всіх ключових випадках використання. Деякі постачальники приділяють велику увагу об’єму даних, але при цьому менше фокусуються на швидкодії, або ж акцентуються на віртуалізації та реплікації даних, проте велика кількість інформації (моделей, даних) не синхронізується. Інші, задля поліпшення сприйняття даних, шукають нові шляхи, наприклад, за допомогою інтеграції віртуальної реальності, однак ця функціональність доступна не на всіх

платформах (системах), а також в таких випадках погіршується взаємодія користувача із системою через недосконаліть кабінету користувача.

Для розробки design gallery, саме таку назву має веб-застосунок розроблений та удосконалений протягом виконання кваліфікайної роботи магістра, був проведений аналіз інтерфейсу, безпеки даних та архітектури популярних веб-додатків у сфері 3d моделювання онлайн:

- Sketchfab (https://sketchfab.com/)
- Online Vizua3d (https://online.vizua3d.com/)

Sketchfab та Online Vizua3d являють собою платформу (маркетплейс) для розповсюдження 3D-об'єктів з підтримкою різноманітних форматів та не надають змогу користувачеві створювати / змінювати тривимірні моделі, а тільки завантажувати чи скачувати (купляти) їх. Аутентифікація – на основі форми, також є можливість логіну за допомогою соціальних мереж. Sketchfab надає можливість для логіну за допомогою SSO (Single sign-on) компанії.

- P3D (https://p3d.in/)
- Online 3D Viewer (https://3dviewer.net/)

У той час як Online 3D Viewer надає користувачеві можливість робити тільки імпорт власних файлів, а саме тривімірних об'єктів, для подальшого огляду їх відображення, у P3D можна також створювати різні папки/колеції для їх зберігання та подальшого використання. Юзер може використовувати "shortcut keys", такі як:

– R для повернення дефолтного відображення моделі;
– X для зуму (приближення відображення 3d моделі);
– Z для обертання фігури.

Окрім цього, користувачеві представлено багато варіантів для зміни матеріалу поверхні моделі, наприклад, параметр glossiness, що дозволяє контролювати блиск матеріалу, або ж specularity – для зміни кольору і насиченості матеріалу.

Однак, можливість змінити форму самої моделі, а також задіяти 2 чи більше моделей одночасно під час редагування (мультимодальність) відсутня.

В Online 3D Viewer відсутня можливість для аутентифікації, а в P3D є на основі форми або за допомогою соціальних мереж.

- Figuro.io (https://www.figuro.io/)
- Clara.io (https://clara.io/)

Figuro.io та Clara.io надають змогу користувачам створювати, анімувати та візуалізувати 3D-вміст на будь-якому сумісному пристрої за допомогою полігонального моделювання з цілим рядом інструментів для редагування. Юзери мають доступ до багатьох функцій, включаючи створення тривірних моделей різних типів та їх анімацію, підтримується імпорт та експорт 3d сцен та об’єктів. У Figuro.io присутня функція додавання галерей/колецій для зберігання та подальшого використання тривимірних моделей та сцен.

Проте, через велику кількість функцій інтерфейс є не дуже зрозумілим, що викликає дискомфорт у користувача та спостерігається дещо повільна швидкодія у порівнянні з іншими веб-додатками, наприклад, SelfCAD. Окрім цього, відсутня функція взаємодії з двома або більше тривімірними об’єктами одночасно як і у вище зазначених веб-застосунках.

Щодо аутентифікації, то в цих двох застосунках вона можлива тільки на основі форми.

- Vectary (https://www.vectary.com/)

Vectary являю собою програмне забезпечення для моделювання 3D сцен та об’єктів онлайн. Даний веб-застосунок пропонує багато вдосконалених інструментів, таких як параметричне моделювання, редагування сітки (mesh) завдяки слайдерам та вибору (selection), які зазвичай доступні лише у висококласному інженерному програмному забезпеченні. До того ж тут є можливість колаборації – декілька людей можуть працювати над одним й тим самим проектом та створення колекцій власних робіт з можливістю подальшого використання. Працюючи над проектом, користувач може використовувати “shortcut keys”, які відомі з іншого програмного забезпечення та інструментів:

- Відміна минулої дії - Ctrl+Z.
- Скасувати вибір або відмінити дію - Esc
- Копіювати вибір (selection) - Ctrl+C.
- Вставити вибір (selection) - Ctrl+V.
- Вирізати вибір (selection) - Ctrl+X.

До недоліків можна віднести високі вимоги до апаратних можливостей пристрою, а також те, що в першу чергу Vectary орієнтований більше на дизайнерів, тобто хоч і можна створювати складні моделі та сцени, вони не будуть дуже точними. Аутентифікація – на основі форми або за допомогою соціальних мереж.

- SelfCAD (https://www.selfcad.com/app/)

SelfCAD – це всебічний хмарний CAD-додаток, який дозволяє всім користувачам моделювати, ділити на частини та друкувати 3D-об’єкти, сцени та проекти онлайн. Завдяки доступу до всіх інструментів та можливостей, немає

необхідності запускати і використовувати різноманітні спеціальнихі програми. Присутня функція додавання колецій для зберігання та подальшого використання тривимірних моделей, сцен, тощо з можливістю поширення дизайнерських робіт в соціальних мережах, наприклад, Facebook або Instagram.

Однак, зважаючи на представлені потужні інструменти, не всі пристрої – персональні комп'ютери, ноутбуки, планшети можуть добре взаємодіяти з даним застосунком через недостатні характеристики "заліза". Мультимодальність тут представлена як можливість виділення деякої області 3d-сцени з подальшими змінами всіх об'єктів одразу, що не є досить зручно, а також безкоштовна версія програми надає тільки мінімальний набір функцій для взаємодії з тривімірними моделями та сценами.

Аутентифікація, як і в більшості представлених для порівняння додатків, на основі форми або за допомогою соціальних мереж.

2.4 Постановка задачі

Підводячи підсумок огляду існуючих продуктів (див. підрозділ 2.3.1), можна виокремити деякі загальні риси вищезгаданих веб-застосунків:

- Можливість створення 3d моделей та сцен чи імпорт існуючих
- Представлення галерей / колекцій тривимірних об'єктів з можливості їх подальшої зміни та використання
- Полігональне моделювання
- Постійна синхронізація даних
- Аутентифікація на основі форми або за допомогою соціальних мереж

Приймаючи до уваги вищепоставлені риси та дивлячись на практики окремих веб-додатків було вирішено удосконалити веб-застосунок design gallery.

Головна перевага цього веб-додатку є підтримка мультимодальності, яка дозволить користувачам взаємодіяти з кількома тривимірними об’єктами відразу різними способами, у тому числі за допомогою використання “short keys”, в різних представленнях, для прикладу у вигляді list та parallel view, що значно прискорить процес створення та редагування сцен та моделей з постійною синхронізацією даних. Оновлений дизайн односторінкового застосунку створений згідно зі стандартами і практиками щодо критеріїв якості побудови інтерфейсів користувача (див. підрозділ 1.2) та легкість даного веб-застосунку, тобто навіть пристрої, які не мають великих апаратних потужностей зможуть користуватись ним, дають змогу швидко та ефективно вирішувати поставлені завдання перед архітекторами та дизайнерами.

РОЗДІЛ 3. АРХІТЕКТУРА ТА РЕАЛІЗАЦІЯ СПЕЦІАЛІЗОВАНОГО ВЕБ-ДОДАТКУ

3.1. Структура веб-додатку редактора

Після постановки задачі (див. підрозділ 2.4) та перед тим як безпосередньо приступати до розробки аутентифікації, впровадження розділів, зміни дизайну та удосконалення онлайн застосунку необхідно оновити саму структуру проекту.

Архітектура веб-застосунку представлена на рисунку 10. Стрілками зображено напрямки потоків даних при взаємодії.

Структура у веб-додатку передбачає наявність користувацької частини (front-end) та серверної (back-end), яка у свою чергу взаємодіє з базою даних, відповідає за логіку аутентифікації та передачі даних.

Показану на рисунку архітектуру можна віднести до компонентів системи автоматизованого проектування (САПР, computer aided design), яка дозволяє проектувати технологічні процеси з меншими витратами часу та засобів, зі збільшенням точності спроектованих процесів і програм обробки, що скорочує витрати матеріалів.

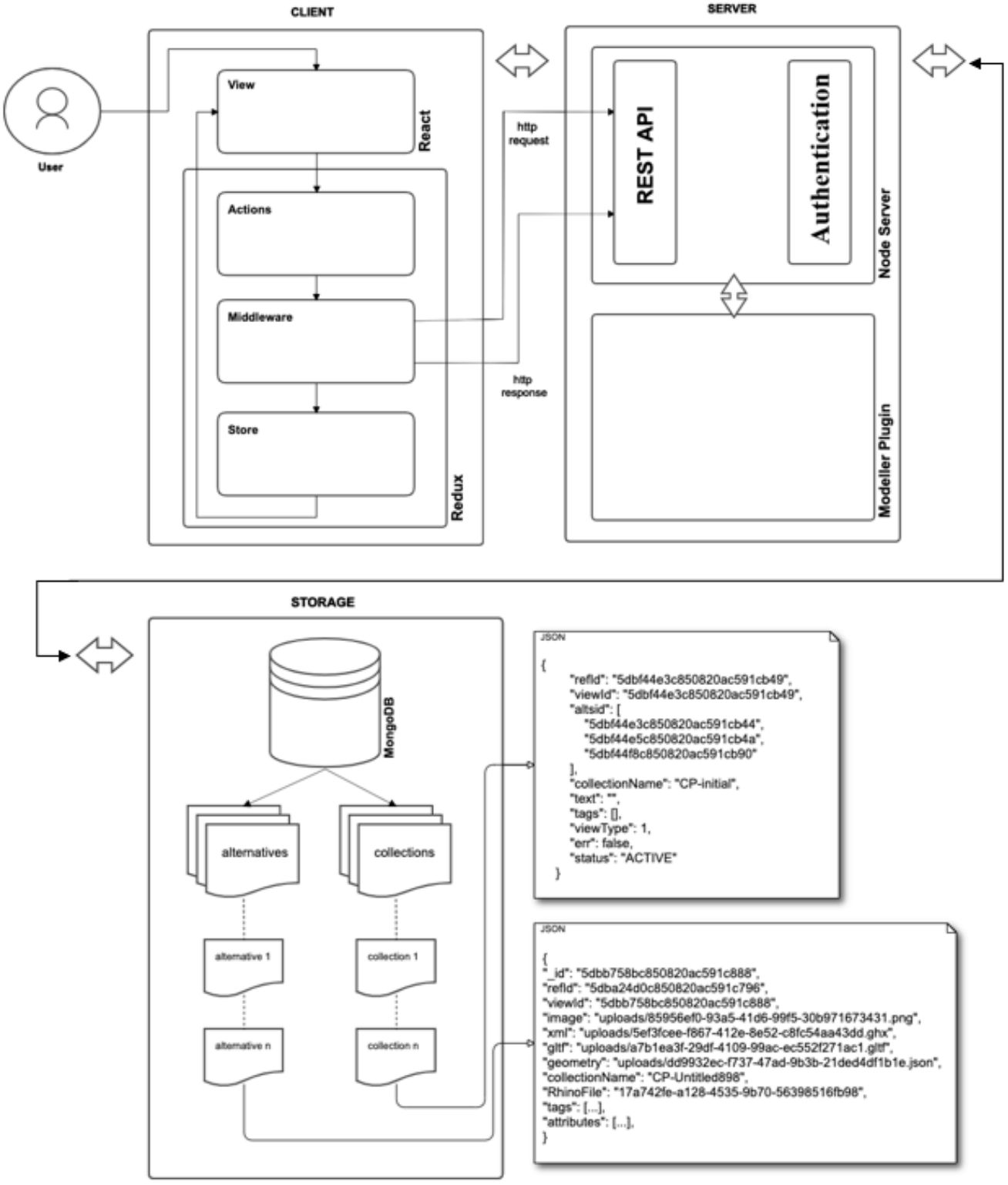

Рисунок 10 – Загальна архітектура веб-додатку design gallery

Ефективні системи автоматизованого проектування і розрахунку включають такі основні компоненти: інтерфейс користувача (UI) та логіку програми.

До логіки програми відносяться:

- База даних, яка є стандартним програмним компонентом для всіх програм, а не тільки для САПР.
- 3D Modeler (Modeler Plugin) є критично важливим компонентом САПР. Він дозволяє створювати, змінювати і запитувати геометричне подання об'єктів для їх візуалізації, моделювання або аналізу.
- Сервер у якому зосереджена основна бізнес-логіка логіка доступу до бази даних та безпека даних в загальному.

Користувач взаємодіє із представленням (View), що є частиною концепції MVC, яка дозволяє розділити дані (модель), представлення і обробку дій (вироблену контролером) користувача на три окремих компоненти:

– Модель (Model):
    - Надає дані та методи роботи з цими даними;
    - Реагує на запити, змінюючи свій стан;
    - Не містить інформації, як ці дані можна візуалізувати;
– Представлення (View):
    - Відповідає за відображення інформації (візуалізацію).
– Контролер (Controller)
    - забезпечує зв'язок між користувачем і системою: контролює введення даних користувачем і використовує модель і представлення для реалізації необхідної реакції.

Redux є контейнером стану для JavaScript-додатків. Він дозволяє створювати додатки, які ведуть себе однаково в різних середовищах (клієнт, сервер або ж нативні додатки).

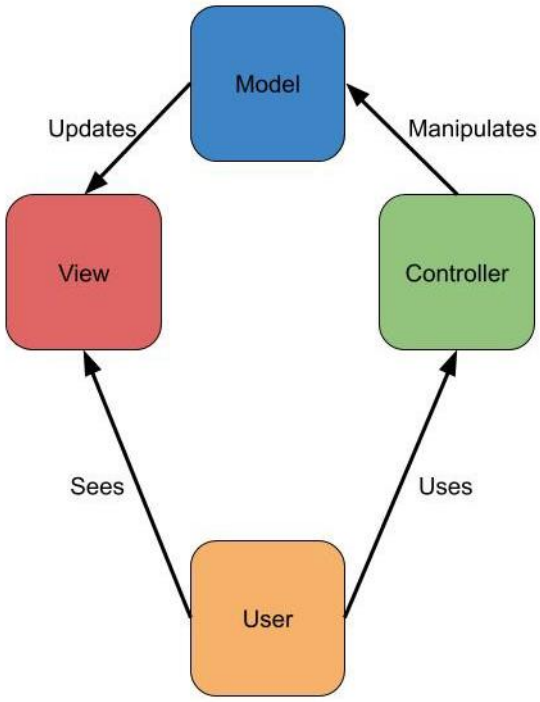

Рисунок 11 – Коцепція MVC для react-компонентів

Використовуючи підхід зображений на рисунку 11 надає такі переваги:

- Використання тільки Stateless-компонентів. Більшу частину яких можна написати у вигляді Functional-component, що є рекомендованим підходом, тому що вони швидше за все працюють і споживають найменше пам'яті
- React-компоненти можна перевикористати з різними контролерами або без них
- Легко писати тести, адже логіка і відображення не пов'язані між собою
- Жорсткі правила при розробці роблять код React-компонентів одноманітним
- Відсутні проблеми з серверним рендерингом

Після того, як користувач щось змінить і нові дані дійдуть до Redux (Model), буде здійнено Rest API виклик на серверну частину, яка вже у свою чергу оброблює їх та взаємодіє з базою даних і повертає результат на користувацьку частину.

3.2 Створення 3d-моделей

Для того, щоб користувач міг взаємодіяти з параметризованими 3d-моделями за допомогою веб-сторінки, спочатку треба створити ці моделі та передати їх дані на back-end, які будуть відповідно оброблені та відображені на користувацькій частині.

Для створення моделей було обрано середовище Grasshopper, яка працює в межах програмного забезпечення для автоматизованого дизайну (CAD) Rhinoceros 3D.

Rhinoceros 3D – це програмне забезпечення для 3D-моделювання, яке використовується в процесах автоматизованого проектування (CAD), автоматизованого виготовлення (CAM) та зворотного проектування у великій кількості галузей, включаючи архітектуру, промисловому та дизайні продукції, а також для мультимедійного та графічного дизайну.

Геометрія цього програмного забезпечення для 3D-моделювання заснована на математичній моделі NURBS, яка фокусується на створенні математично точного уявлення кривих і поверхонь довільної форми в комп'ютерній графіці (на відміну від додатків на основі багатокутних сіток).

Grasshopper – це середовище та візуальна мова програмування, яка працює в межах програмного забезпечення для автоматизованого дизайну (CAD)

Rhinoceros 3D. Програми створюються перетягуванням компонентів на полотно. Виходи цих компонентів потім підключаються до входів наступних компонентів.

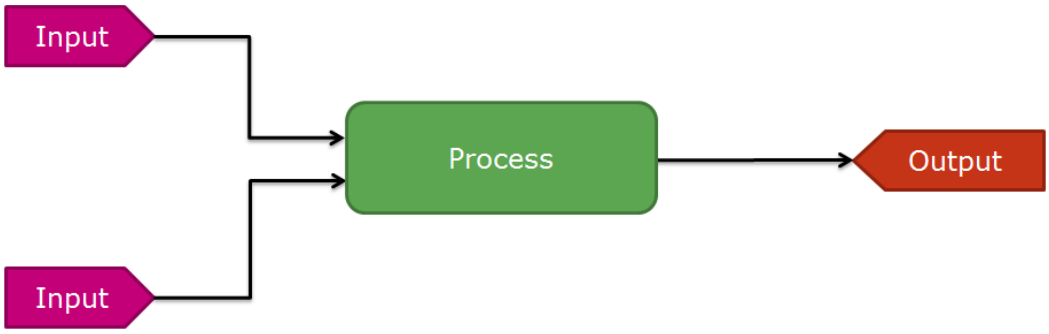

Рисунок 12 – Структура взаємодії елементів у Grasshopper

Дане середовище використовується насамперед для побудови генеративних алгоритмів, багато компонентів Grasshopper створюють тривимірну геометрію. Програми також можуть містити інші типи алгоритмів, включаючи числові, текстові, аудіовізуальні та тактичні програми.

Завдяки даній візуальній мові програмування можна створювати не лише дуже точні об’єкти та їх колекції, а й прототипи ландшафтів та різноманітних комплексів.

Приклади об’єктів та проттипів зображені на рисунках нижче:

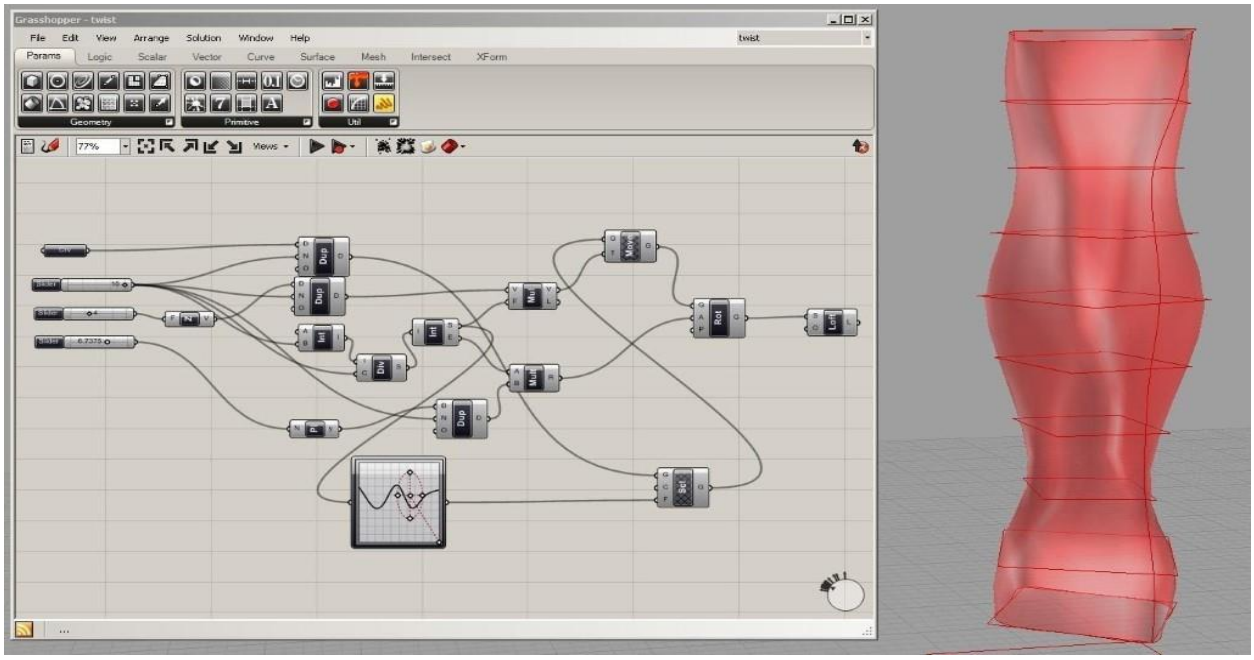

Рисунок 13 – Приклад схеми створення об’єкта у програмі Rhino за допомогою мови Grasshopper

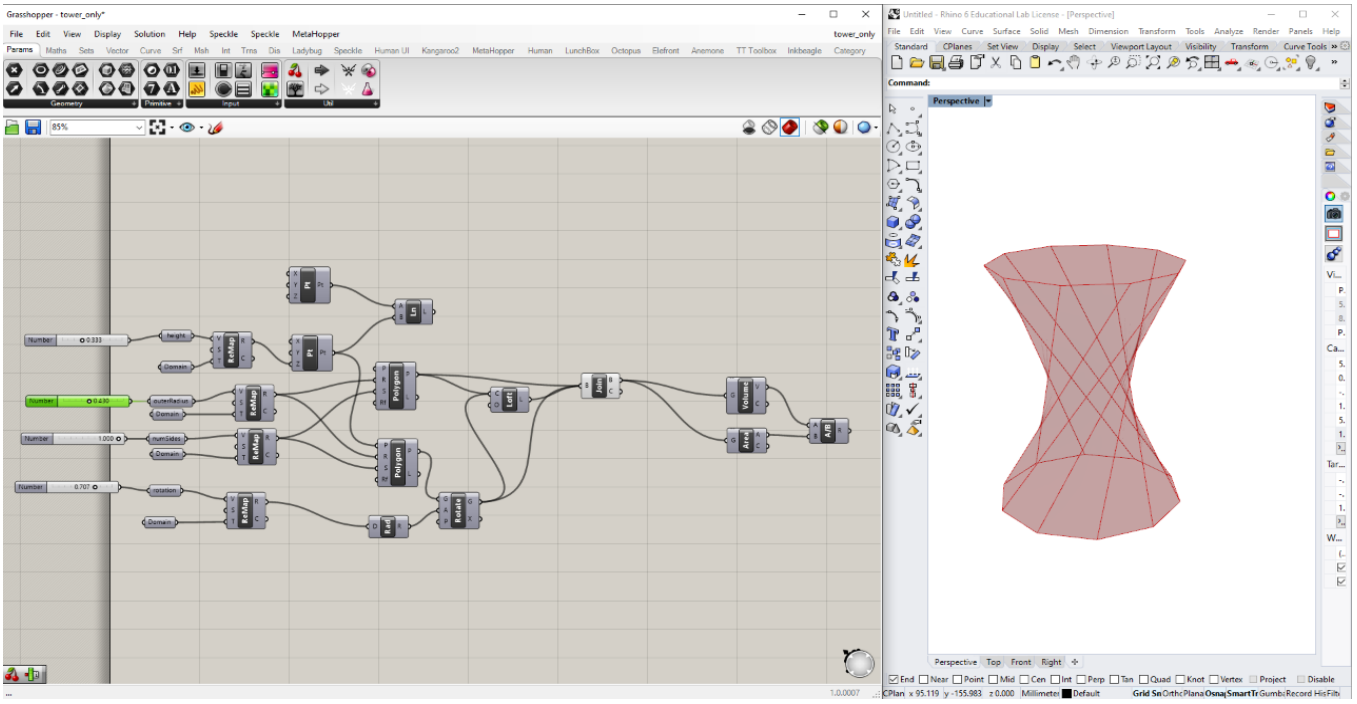

Рисунок 14 – Приклад схеми створення колони

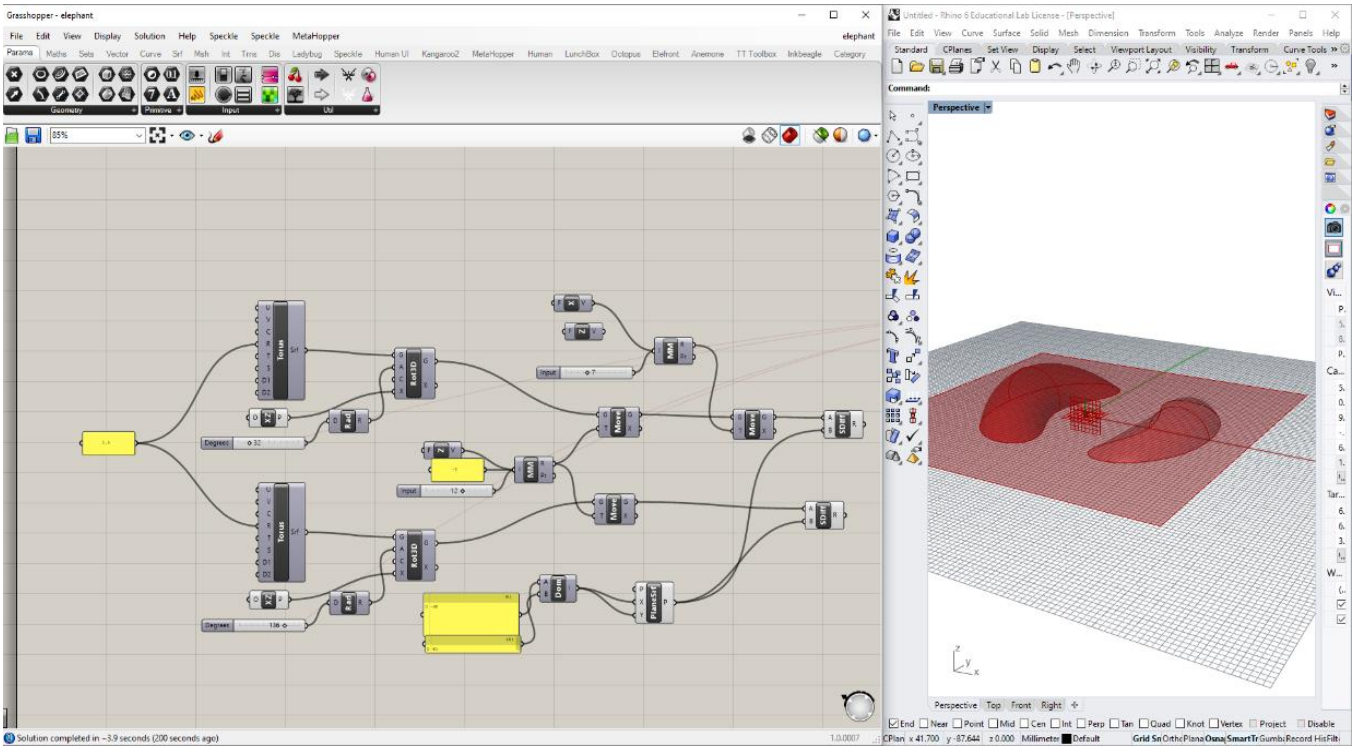

Рисунок 15 – Приклад схеми створення ладншафту

3.3 Реалізація серверної частини

3.3.1 База даних

Однак, навіть після створення 3d-моделей, їх не можна було передати для відображення на користувацьку частину, адже в основному, файли мали формат 3MF, що унеможливлювало їх рендеринг та були досить “важкими”, що значно впливало на швидкодію ресурсу.

Саме тому було вирішено зберігати дані об’єкти в основному у вигляді точок вершин і граней (vertices and faces), що являли собою цілі числа (н-д, 1,0,4 і т.д.) у базі даних MongoDB.

У “Додатку А” наведений приклад зберігання даних тривимірних моделей у веб-застосунку design gallery.

MongoDB — це кросплатформна, документо-орієнтована СКБД (система керування базами даних), яка реалізує підхід до побудови БД, де немає запитів SQL, схем, та інших компонентів, які використовуються в об'єктно-реляційних БД [10]. У MongoDB впроваджена документо-орієнтована модель даних, завдяки чому вона працює швидше та має кращу масштабованість у порівнянні з реляційними базами даних.

Нереляційна база даних – це база даних, у якій застосовується модель зберігання, оптимізована під конкретні вимоги типу збережених даних. Наприклад, дані можуть зберігатися як прості пари "ключ - значення", документи JSON або граф, що складається з ребер і вершин.

Система MongoDB може складатися не тільки з однієї бази даних, що знаходиться на одному сервері в хмарі або фізично, а дозволяє розташувати кілька БД відразу на декількох серверах. У цих базах запезпечена цілісність та синхронність для обмінювання даними.

Для обміну даними та їх зберігання в MongoDB використовується BSON (скорочення від binary JSON - JavaScript Object Notation) формат. Порівняно з JSON, він дозволяє працювати з даними швидше: виконується пошук і обробка.

3.3.2 Веб-сервіс

Для реалізації серверної частини (back-end) та аутентифікації було вирішено використовувати середовище розробки Node.js.

Node.js — це асинхронне середовище і платформа для створення та виконання масштабованих мережевих додатків, написаних мовою JavaScript [11]. Мова Javascript раніше застосовувалася тільки для обробки даних в браузері. З появою Node.js з’явилася можливість виконувати JavaScript-сценарії на сервері

та надсилати результат їх виконання користувачеві. Власне, через використання JavaScript на бекенді, дане середовище набуло дуже великої популярності серед розробників, адже тепер можна реалізувати повноцінний проект за допомогою однієї і тієї ж самої мови.

Інтерес до цієї технології досяг найвищого рівня в 2017 році, відповідно до Google Trends (https://trends.google.com/trends/explore?q=node.js), і все ще залишається високим.

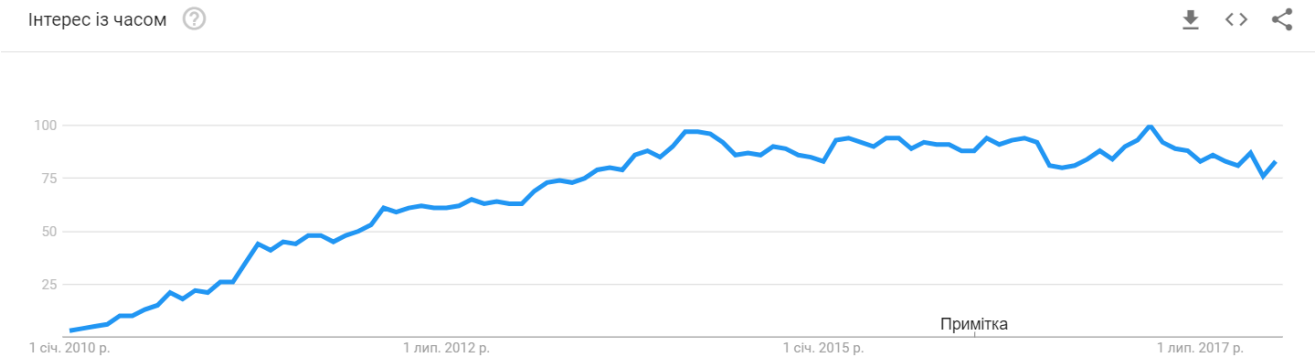

Рисунок 16 – Графік популярності пошукового терміна Node.js відносно найвищої точки на графіку для періоду з 2010 по 2019 рік.

Асинхронна модель запуску коду використовується для обробки великої кількості даних та паралельних запитів у Node.js, яка заснована на визначенні обробників зворотніх викликів та обробці подій в неблокуючому режимі.

Під час обговорення Node.js не слід пропускати одну річ - вбудовану підтримку управління пакетами за допомогою інструменту npm (node package manager), який поставляється за замовчуванням при кожній установці Node.js. Ідея модулів npm дуже схожа на ідею Ruby Gems: набір загальнодоступних, повторно використовуваних компонентів, доступних за допомогою простої установки через онлайн-сховище, з управлінням версіями і залежностями.

До інших переваг Node.js можна віднести масштабованість та розширюваність. Додатки легко масштабуються як по горизонталі, так і по вертикалі. При розробці програми з Node.js не треба створювати велике монолітне ядро. Натомість можна розробити набір модулів та мікросервісів, кожен з яких працює у своєму процесі. Усі ці невеликі сервіси спілкуються з легкими механізмами та складають додаток. Додавання додаткового мікросервісу настільки ж просто, наскільки це можливо. Таким чином, процес розробки стає набагато гнучкішим.

На рисунку 17, що зображений нижче, представлена візуальна різниця між монолітною та мікросервісною архітектурою.

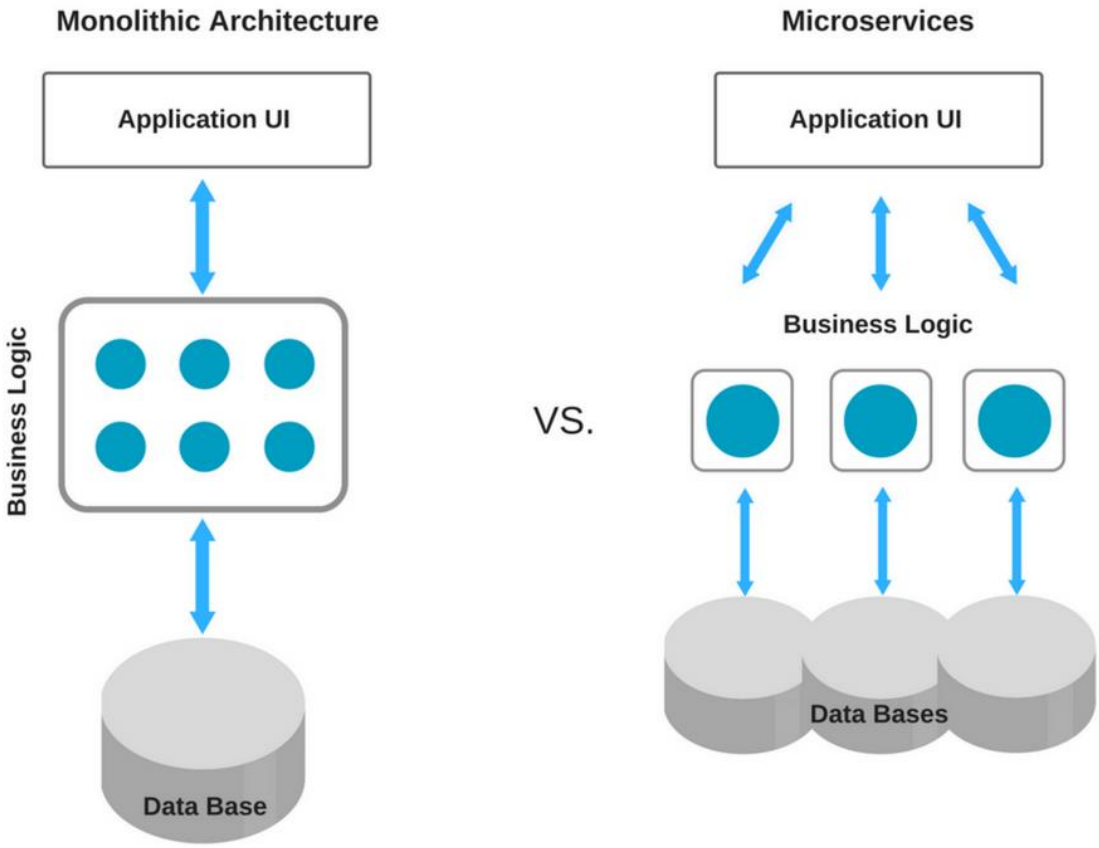

Рисунок 17 – Різниця між монолітною та мікросервісною архітектурою

3.3.2.1 Алгоритм створення 3D-моделей

Для того, щоб відрендерити 3D моделі на користувацькій частині було вирішено використовувати JavaScript бібліотеку three.js за допомогою якої, на основі даних, які бралися з БД, можна було створювати тривимірні об’єкти у форматі .glTF, що власне й дозволяло відображати їх у браузері.

Частина програмного коду, за допомогою якого було створено 3d-моделі у форматі .glTf й експортовано для відображення у браузері знаходиться у “Додатку Б”.

Оновлений алгоритм також вплинув на швидкодію додатку – завантаження моделей стало швидше, що відображається у метриках. (див. підрозділ 3.6)

Щоб аргументувати вибір даної бібліотеки необхідно навести базовий огляд існуючих рішень, в часності WebGL.

WebGL (Web Graphics Library) – це програмна бібліотека для JavaScript, яка дозволяє створювати 3D графіку, що функціонує в браузерах. Дана бібліотека заснована на архітектурі бібліотеки OpenGL. WebGL цікавий тим, що код моделюється безпосередньо в браузері і для цього бібліотека використовує об'єкт canvas, який був введений в HTML5. Робота з WebGL, і з шейдерами зокрема, - це досить трудомісткий процес. При розробці необхідно описати кожну точку, лінію, грань і так далі. Щоб все це візуалізувати, необхідно написати досить великий об'єм коду.Саме для підвищення швидкості розробки, була розроблена бібліотека Three.js.

Three.js — це бібліотека JavaScript, що містить набір готових класів для створення і відображення інтерактивної 3D графіки в WebGL, а також яка підтримує кросбраузерність та інтерфейс прикладного програмування (API), що

використовується для створення та відображення анімованої 3D-комп'ютерної графіки у веб-браузері [12].

GLTF (GL Transmission Format) - це формат файлу для зберігання 3D сцен і моделей, який є простим в розумінні (структура записана в стандарті JSON), розширюваним і легко взаємодіє з сучасними веб-технологіями [13]. Даний формат добре стискає дані і мінімізує як розмір об’єктів 3D, так і обробку часу виконання, необхідну для розпакування та використання цих ресурсів, що використовують WebGL і інші API.

glTF базується на 2х форматах файлів: JSON для опису структури всієї 3d сцени і бінарного файлу, для зберігання всіх даних з сцени. Існує і бінарна версія glTF, яка називається GLB, єдина відмінність якого в тому, що все зберігається в одному файлі з розширенням GLB.

В якості додаткових плюсів в glTF можна виділити:

– Чітка ієрархія об'єктів в структурі 3d сцени
– Зберігання такої інформації про сцену, як джерела світла і камери
– Більш надійні матеріали і шейдери

Уже конвертовані дані у glTF формат, тобто самі 3d моделі, передаються на користувацьку частину за допомогою Rest API.

REST (Representational State Transfer) — інтерфейс управління інформацією без використання будь-яких додаткових внутрішніх шарів та являє собою архітектурний підхід мережевих протоколів, який включає універсальні способи передачі станів ресурсів і обробки по HTTP.

REST підхід надає такі можливості:

– Масштабованість взаємодії компонентів системи (додатка)
– Спільність інтерфейсів
– Незалежне впровадження компонентів
– Проміжні компоненти, що знижують затримку, посилюючи безпеку

Також, до переваги REST підходу відноситься управління інформацією ресурсу, що цілком і повністю ґрунтується на протоколі передачі даних. Найбільш поширений протокол звичайно ж HTTP. Для HTTP дію над даними задається за допомогою методів: GET (отримати), PUT (додати, замінити), POST (додати, змінити, видалити), DELETE (видалити). Таким чином, дії CRUD (Create-Read-Update-Delete) можуть виконуватися як з усіма 4-ма методами, так і тільки за допомогою GET і POST.

HTTP метод GET використовується для отримання (або читання) представлення ресурсу. Якщо все ок, GET повертає представлення ресурсу у форматі XML або JSON в поєднанні з кодом стану HTTP 200 (OK). Якщо ні – 404/400.

Наприклад, у веб-додатку design gallery GET запит використовується для отримання об’єкту зі списку по айді.

Реалізація цього запиту наведено нижче:

```
/**
  * find alternative object from a list from alternativeID
  * @param {ImmutableList} alternatives
  * @param {int} alternativeID
  * @return {ImmutableMap}
  */
 export const findAlternativesFromID = (alternatives, alternativeID) => (
```

```
   alternatives.find((alternative) =>
     alternative.get(alternativeObjectConstant.get('ID')) ===
alternativeID
  )
 );
```

HTTP метод PUT зазвичай використовується для надання можливості оновлення ресурсу. Тіло запиту при відправленні PUT-запиту до існуючого ресурсу URI має містити оновлені дані оригінального ресурсу (повністю, або тільки оновлювану частина).

Для створення нових екземплярів ресурсу краще використовувати POST запит. PUT не є безпечною операцією, тому що в результаті її виконання відбувається зміна (або створення) екземплярів ресурсу на стороні сервера.

HTTP метод POST найбільш часто використовується для створення нових ресурсів. На практиці він використовується для створення вкладених ресурсів.. При успішному створенні ресурсу повертається HTTP код 201, а також в заголовку `Location` передається адреса створеного ресурсу.

Приклад реалізації POST у додатку design gallery для оновлення колекції:

```
/**
  * update the collection with alternative IDs
  * @param {int} altId: ID of the alternative
  * @param {Object} parentCollection
  * @return {Promise}
  */
 export const updateAlts = (altId, parentCollection) => (dispatch,
getState) => (
   fetch(urls.get('collections'), {
```

```
    method: 'POST',
    headers: {'Content-Type': 'application/json'},
    body: JSON.stringify({
      altId,
      parentCollection,
    }),
  }));
```

HTTP метод DELETE використовується для видалення ресурсу, ідентифікованого конкретним URI (ID). При успішному видаленні повертається код 200 (OK), спільно з тілом відповіді, що містить дані видаленого ресурсу.

Нижче наведена реалізація DELETE методу у веб-застосунку design gallery:

```
export const deleteCollection = (id) => `${webProtocol}${host}/
collections/{$id}/delete`
 /**
  * @param {string} collectionID collectionID of collection to delete
  * @return {redux-thunk}
  */
 export const deleteCollections = (collectionID) => (dispatch) => {
  return fetch(urls.get('deleteCollection')(collectionID), {
     headers: {'Content-Type': 'application/json'},
     body: JSON.stringify({'_id': collectionID}),
  }).then(() => {
     dispatch(deleteCollection(collectionID));
     dispatch(successMessage(toast.success('Collection deleted'),
'collection deleted' + time));
   }).catch((err) => {
     dispatch(errorMessage(toast.error('Collection is not deleted'),
'collection not  deleted ' + time));
```

```
});
```

3.3.2.2 Впровадження аутентифікації

Для аутентифікації було обрано бібліотеку Passport.js, яка повністю сумісна з Node.js, та, як стверджує офіційна документація, має більш ніж 500 стратегій для впровадження безпеки даних. [14]

Passport.js – це проста, ненав’язлива аутентифікація для Node.js, надзвичайно гнучка і модульна, її можна реалізувати в будь-якому веб-додатку на базі Express. Повний набір стратегій підтримує автентифікацію на основі форми, за допомогою соціальних мереж, єдиний вхід, багатофакторна аутентифікація та інші.

Passport.js аутентифікує запити за допомогою стратегії. Стратегія встановлює правила, як аутентифікуватися, і залежно від результату, що робити після цього.

Вибір даної бібліотеки аргументується її перевагами, а саме:

- Єдиний вхід за допомогою OpenID і OAuth
- Підтримує постійні сеанси
- Динамічний діапазон і дозволи
- Можна реалізувати спеціальні стратегії

Для додатку design gallery було реалізовано автентифікацію на основі форми та за допомогою сервісів Azure та Google.

На рисунку нижче зображена схема аутентифікації на основі форми, що передбачає введення логіну та паролю користувачем та має назву – локальна стратегія.

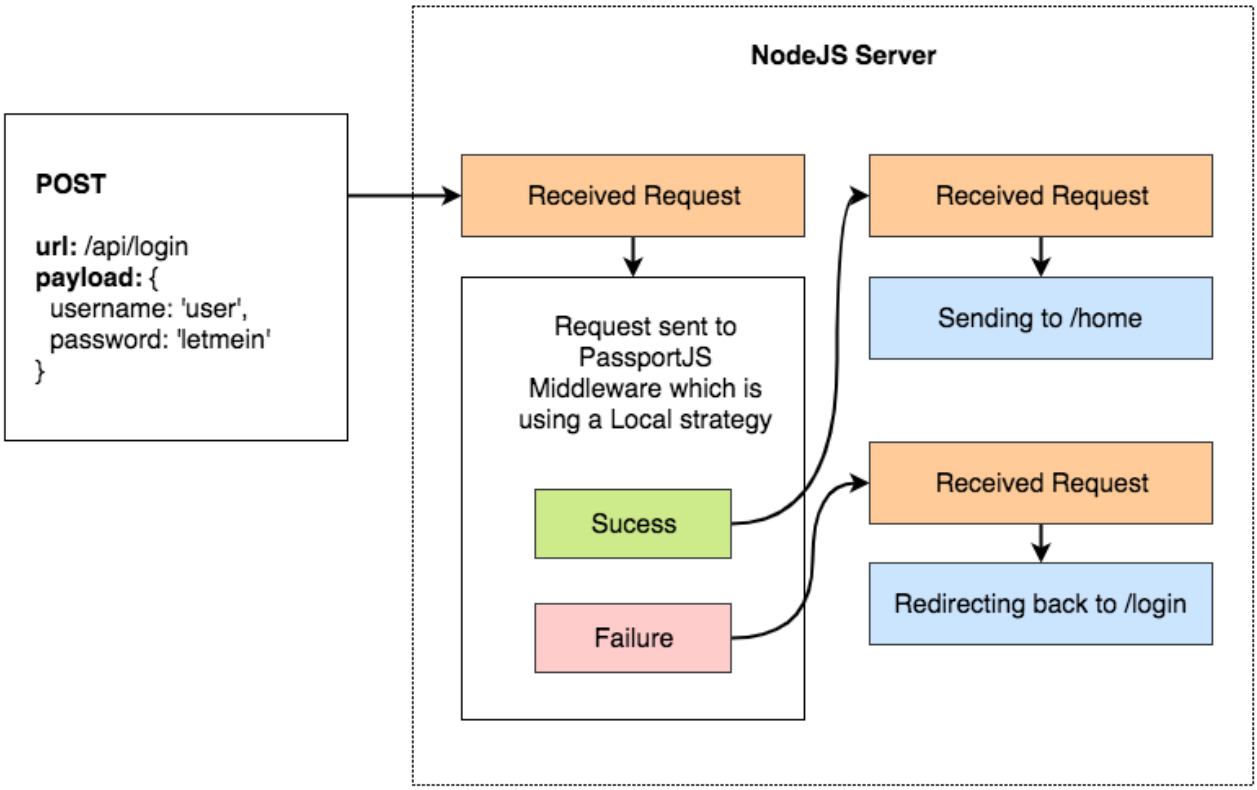

Рисунок 18 – Схема Passport.js використовуючи локальну стратегію

Приклад частини коду використовуючи локальну стратегію знаходиться в “Додатку В”.

Для того, щоб скористатися даним функціоналом, необхідно додати код HTML, як показано нижче, щоб користувач мав змогу ввести дані.

```
<form action="/auth/local">
  <label for='username'>Username</label>
  <input name="username" />
  <label for='password'>Password</label>
  <input type="password" name="password" />
  <button type='submit'>Log in</button>
</form>
```

Passport.js може передавати дані, які має користувач, на сервер аутентифікації, а також URL-адресу зворотного виклику. Після завершення роботи сервера аутентифікації він може переспрямувати до отриманого зворотного виклику з корисним навантаженням, таким як дані користувача або повідомлення про помилку.

Наприклад, використовуючи стратегію Google відбувається перехід на веб-сайт, а під час аутентифікації надішле дані назад на URL-адресу зворотного виклику.

Схема процесу аутентифікації використовуючи стратегію Google зображена на рисунку 19.

Приклад частини реалізації для даної стратегії знаходиться в “Додатку Г”.

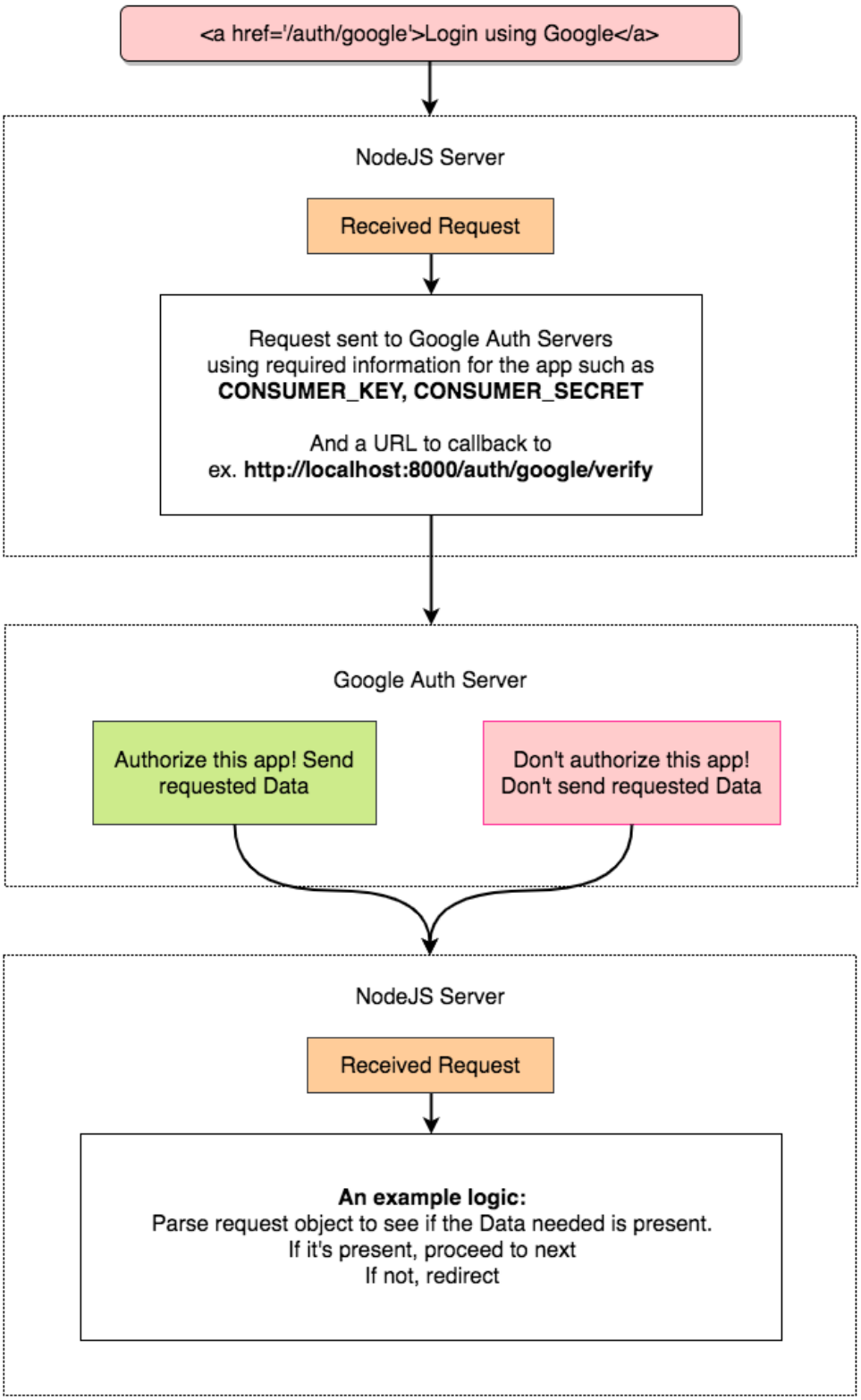

Рисунок 19 – Схема Passport.js використовуючи Google OAuth2 стратегію

3.4 Розробка користувацької частини

Після того, як 3d-модель була створена на сервері та експортована на front-end, її треба відобразити (відрендерити), а також створити user-friendly інтерфейс задля зручної взаємодії користувача з нею.

Для відображення моделей та й взагалі всього користувацького інтерфейсу, а саме односторінкового застосунку (single-page application, SPA) було обрано відкриту JavaScript бібліотеку React.js.

Частина програмного коду, за допомогою якого було створено сцену та рендеринг кожної 3d-моделі можна переглянути у “Додатку Д”.

Односторінковий застосунок (SPA), також відомий як односторінковий інтерфейс (single-page interface, SPI) - це веб-додаток, який має всього одну сторінку для того, щоб забезпечити користувачу досвід схожий до користування настільною програмою. Прикладом односторінкового додатку можна назвати карти Google.

Однією з найкращих переваг правильно налаштованого SPA є швидкість самого застосунку та користувацький досвід (UX). Увесь код - HTML, CSS, та JavaScript - завантажується разом зі сторінкою, або довантажується залежно від дій користувача. Взаємодія зі SPA базується на динамічному зв'язку з веб-сервером. Односторінковий веб-додаток може ефективно кешувати будь-які локальні сховища. Застосунок надсилає лише один запит, зберігає всі дані, після цього можна використовувати ці дані навіть офлайн.

HTML (HyperText Markup Language - «мова гіпертекстової розмітки») – стандартизована мова розмітки документів у Всесвітній павутині. Мова HTML інтерпретується браузерами, отриманий в результаті інтерпретації форматований

текст відображається на екрані монітора комп'ютера або мобільного пристрою. Дана мова розмітки за потреби вбудовує код, написаний на мові сценаріїв, що впливає на вміст та поведінку веб-сторінок. Включення Cascading Style Sheets визначає компонування та вигляд вмісту.

CSS (Cascading Style Sheets) — це спеціальна мова, що використовується для опису зовнішнього вигляду та стилю сторінок. Найчастіше дану мову використовують для візуальної представлення сторінок, написаних мовою гіпертекстової розмітки.

Одна з головних переваг CSS — розділення даних сторінки (змісту) та їх візуального відображення. Блочна верстка дозволяє підлаштувати контент до різних дисплеїв (на екрані настільного монітора, мобільного пристрою).

JavaScript (JS) — об'єктно-орієнтована, динамічна мова програмування, базується на прототипах та є реалізацією стандарту ECMAScript. В основному використовується для створення веб-додатків на основі сценаріїв (скриптів), що в свою чергу дає можливість на клієнтській стороні керувати браузером, взаємодіяти з користувачем, асинхронно обмінюватися даними з сервером, змінювати зовнішній вигляд і структуру веб-сторінки.

Так як JavaScript є дуже популярна протягом багатьох років, до неї з’являється все більше нових бібліотек (наприклад, React.js) та фреймворків (наприклад, Vue.js та Angular), які значно спрощують написання сценаріїв веб-сторінок.

Для створення веб-застосунку було обрано бібліотеку React.js, яка мала деякі переваги у порівнянні з іншими технологіями. У перш чергу через те, що це саме бібліотека, а не фреймворк. З техночної точки зору їх різниця полягає у терміні,

так званому інверсія управління. Тобто, використовуючи бібліотеку, розробник відповідаєте за потік (flow) програми. Він вибирає, коли і де використовувати методи бібліотеки. Коли ж розробник використовує фреймворк, то вже сам код відповідає за потік (flow).

React.js — це JavaScript бібліотека для створення інтерфейсів користувача [15]. Дана бібліотека дозволяє розробникам створювати веб-застосунки різної маштабованості використовуючи дані, які змінюються з часом, без перезавантаження сторінки. React впливає тільки на користувацький інтерфейс у веб-застосунках, що відповідає представленню у шаблоні MVC (модель-представлення-контролер). Найчастіше React використовують з такими бібліотеками як Redux, адже вона була розроблена для підтримання контейнерів стану додатків створених мовою JavaScript.

Так і при розробці архітектури веб-додатку design gallery було вирішено також використовувати Redux (див. підрозділ 3.1, рис. 10)

Сам по собі Redux – це окрема бібліотека, яку можна використовувати з будь-яким рівнем інтерфейсу користувача, включаючи React, Angular, Vue та vanilla JS. Хоча Redux і React зазвичай використовуються разом, вони незалежні один від одного.

Загалом, треба використовувати Redux тоді, коли є розумні обсяги даних, які змінюються з часом, тоді нас потрібне єдине джерело істини, і виявляється, що такі підходи, як утримання всього в стані компонента React верхнього рівня, вже не достатньо [16].

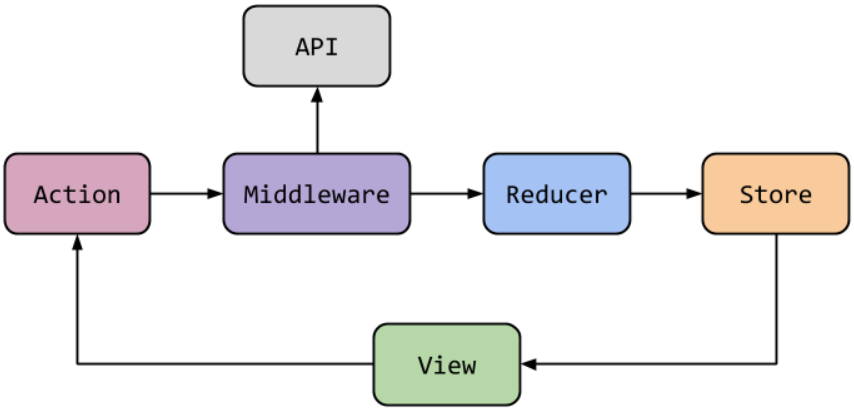

Рисунок 20 – Взаємодія компонентів Redux

3.4.1 Розширення односторінкового застосунку

Після постановки задачі (див. підрозділ 2.4) та аналізу досліджень і практик щодо критеріїв якості побудови інтерфейсів користувача було створено прототипи нового вигляду веб-застосунку та впроваджено їх у додатку.

Так, при першому вході в додаток відображається стартова сторінка.

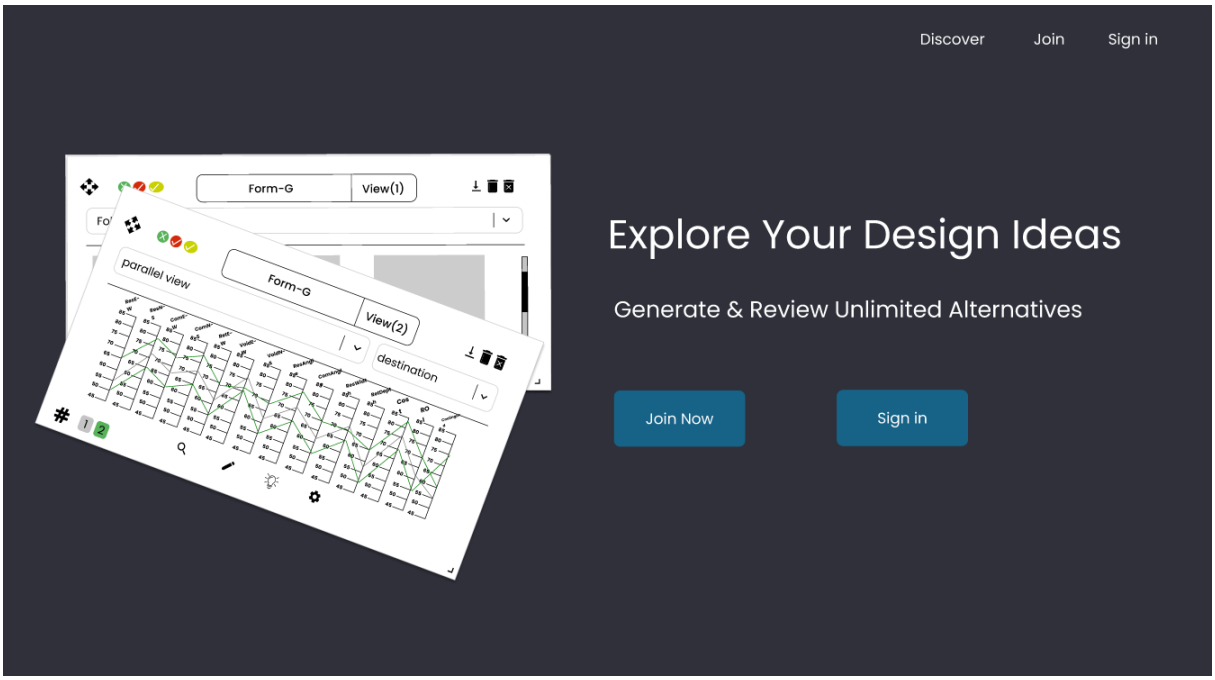

Рисунок 21 – Вигляд стартової сторінки

При аутентифікації, впровадження якої було розглянуто (див. підрозділ 3.3.2.2) сторінка виглядає, як показано на рисунку нижче.

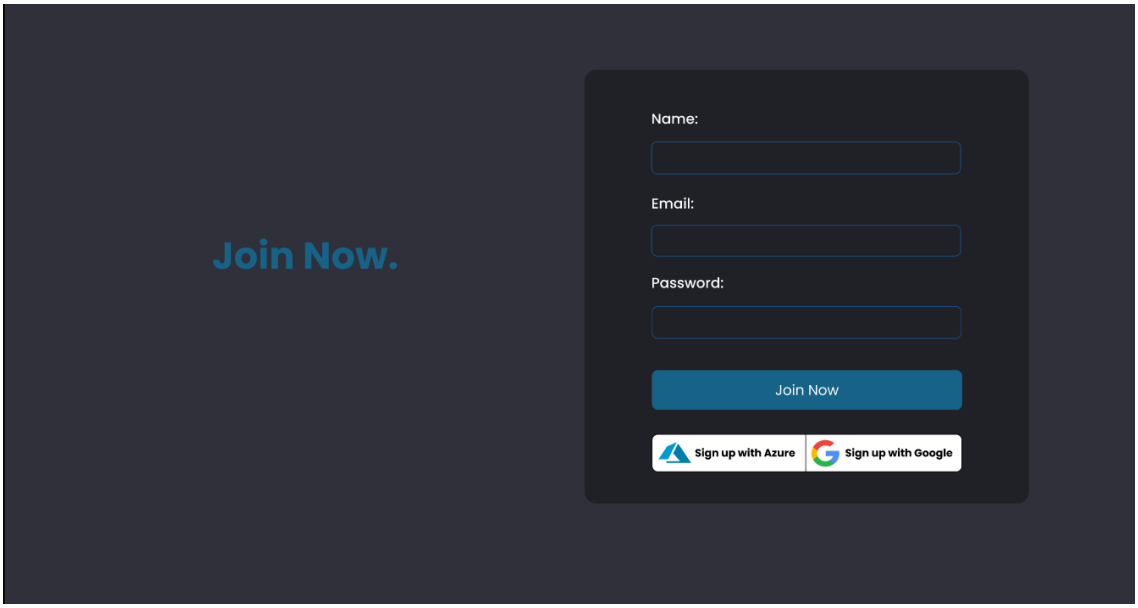

Рисунок 22 – Вигляд сторінки автентифікації

Зовнішній вигляд додатку зазнав змін. Новий інтерфейс головної сторінки:

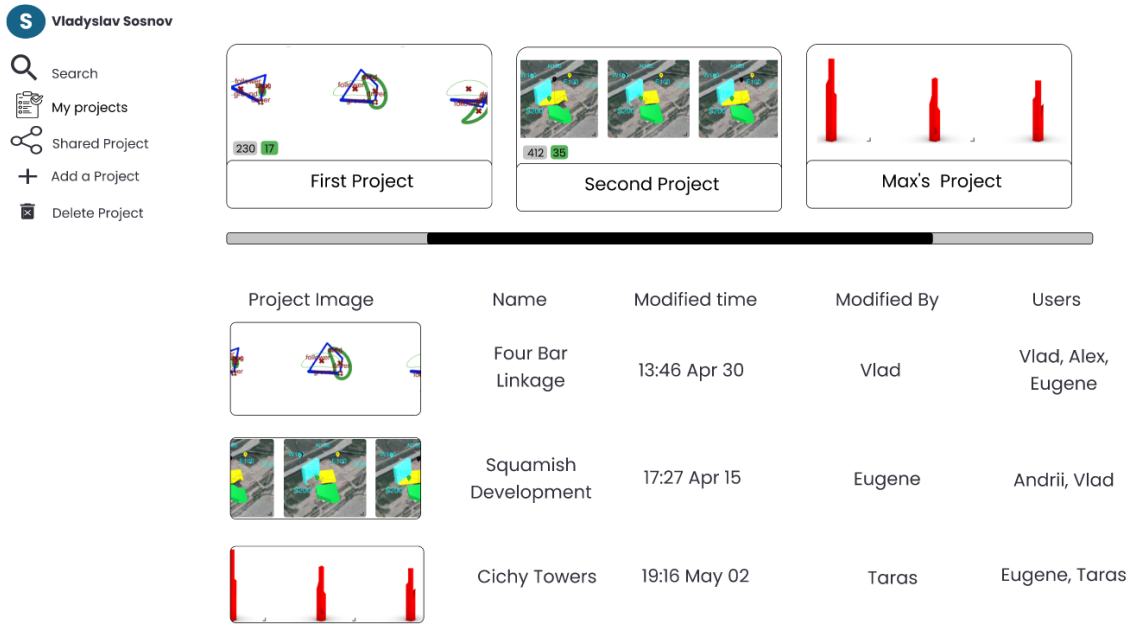

Рисунок 23 – Вигляд головної сторінки

Була реалізована оновлена логіка роутінгу та додано нові розділи відповідно до критеріїв проектування.

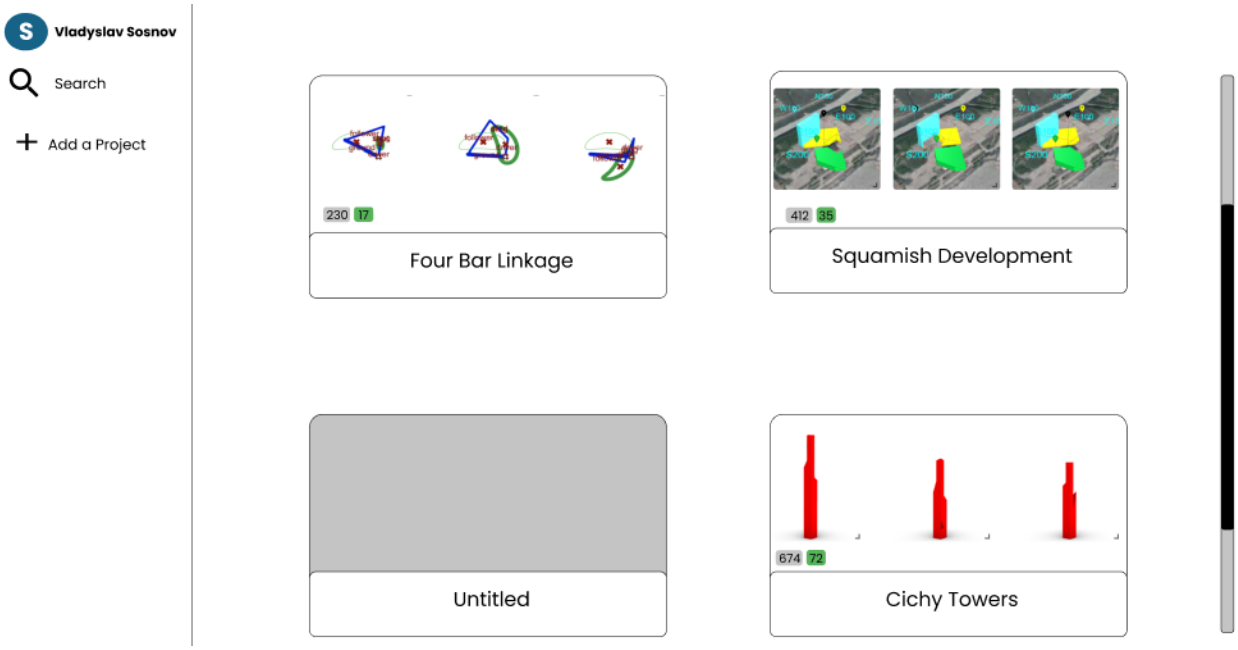

Рисунок 24 – Вигляд розділу «My projects»

Рисунок 25 – Вигляд розділу «Shared project»

Окрім цього, беручи до уваги ”Золоті правила” Бена Шнейдермана, було імплементовано новий процес створення проектів користувачем.

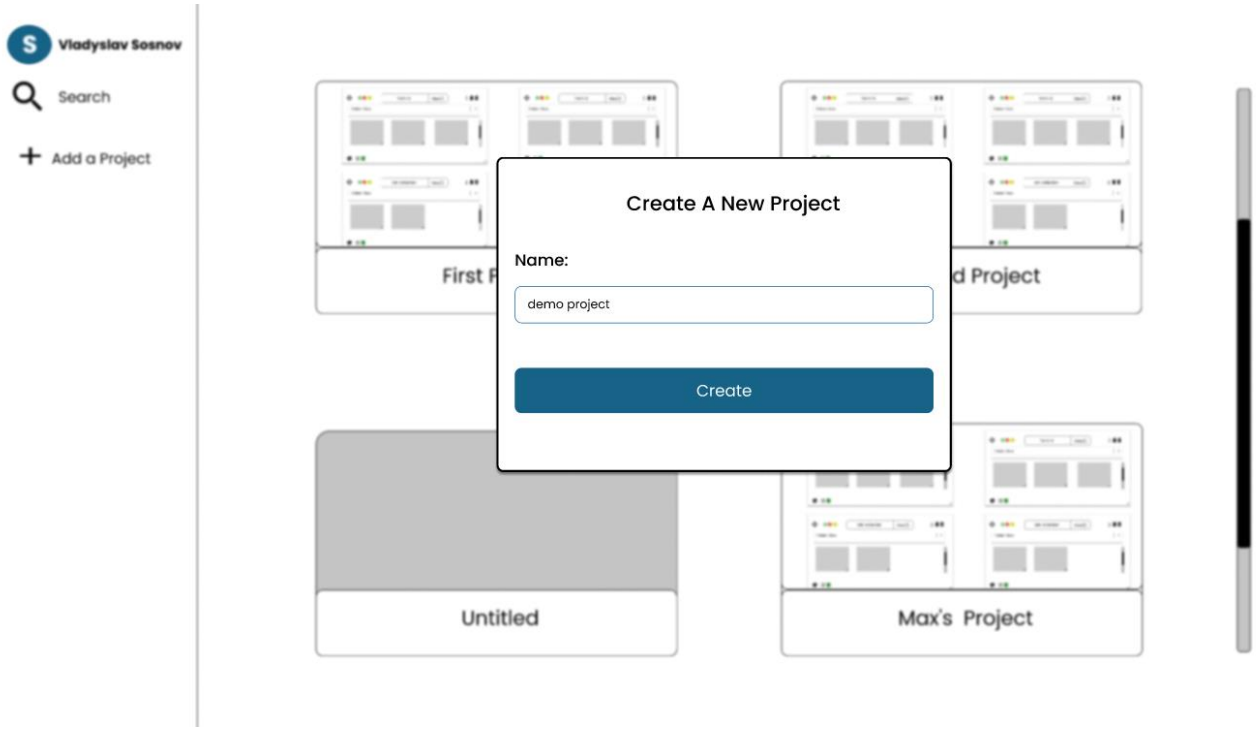

Рисунок 26 – Створення нового проекту

3.5 Основні функції веб-додатку

Згідно з поставленою задачею (див. підрозділ 2.4) головною метою даного застосунку є взаємодія відразу з декількома моделями одночасно з можливістю зміни їх параметрів використовуючи новий інтерфейс, розроблений згідно до критеріїв якості побудови ІК.

Тому, було удосконалено можливість створювати колекції (collections) та додавати моделі всередину (alternatives).

За для того, щоб користувачеві не приходилось кожного разу обертати всі моделі, щоб знайти необхідну, було додано 2D-зображення тривимірного об’єкту, що відразу дає представлення, як виглядає той чи інший об’єкт.

Ще однією перевагою даного веб-застосунку є можливість створювати не тільки копії, а й клони існуючих тривірних моделей, які наявні у колекціях. Якщо при копії моделі чи колекції створюється повністю новий об’єкт, то при клонуванні новий об’єкт буде посилатися на існуючий. Тобто, при зміні нової моделі, також буде змінюватись і та, на основі якої була створена. Це особливо зручно при клонуванні моделей в інші колекції.

Також наявна можливість використання “short keys”, таких як:

- Ctrl + C, Ctrl + X, Ctrl + V – для копії, вирізу та вставлення колекції
- Alt + C, Alt + X, Alt + V – для копії, вирізу та вставлення об’єкту в колекцію
- Ctrl + B – для клону колекції
- Alt + B – для клону тривімірного об’єкту
- DEL – видалення колекцій / об’єктів

Більш того, для зручності користуванням веб-застосунком, можна створити нову колекцію просто зробивши Double-Click на існуючій та перетягувати їх за допомогою імплементованої функції – Click-Drag.

У додаток до реалізованих пунктів вище, була додана можливість запрошення інших користувачів на редагування та одночасна взаємодія користувачів при роботі з 3d-моделями у веб-застосунку.

Рисунок 27 – Запрошення на редагування

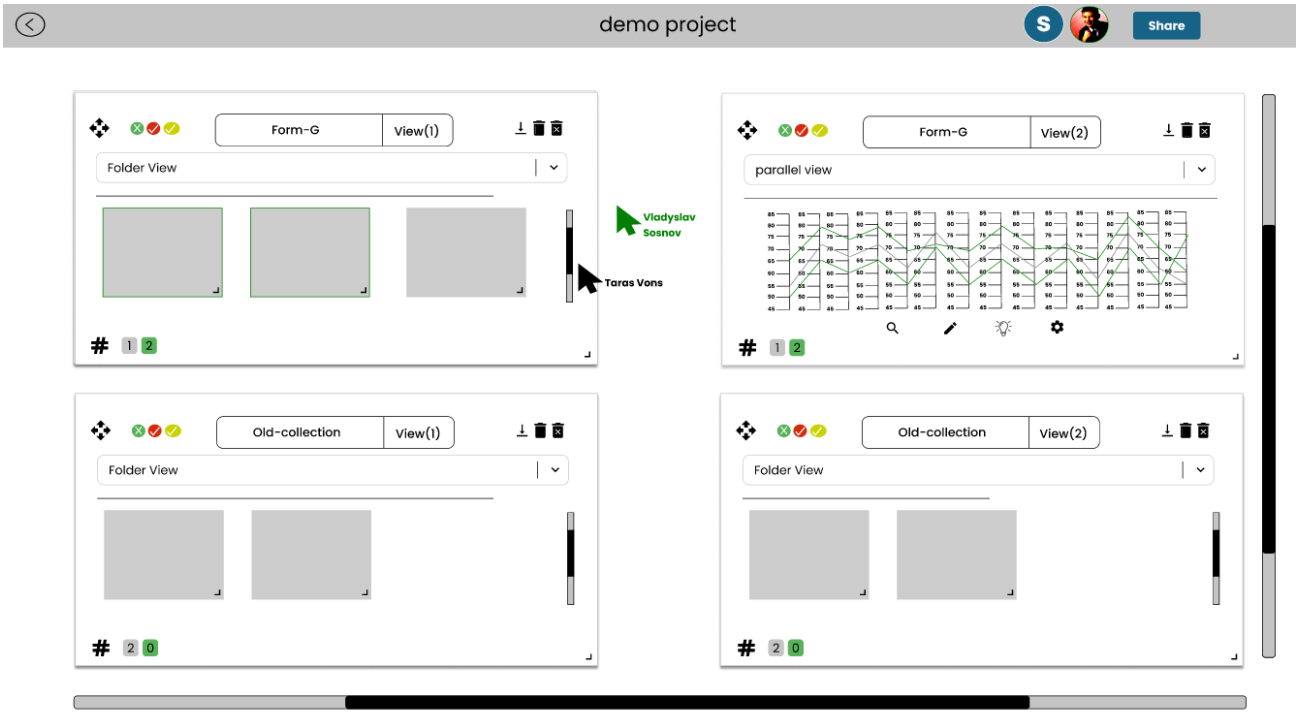

Рисунок 28 – Вигляд одночасної взаємодії користувачів при роботі з 3d-моделями

Однією з головних умов покращення додатку є впровадження колекцій у вигляді List View. Дане вдосконалення дозволить архітекторам наглядно змінювати параметри та дає змогу для більш структурованих змін.

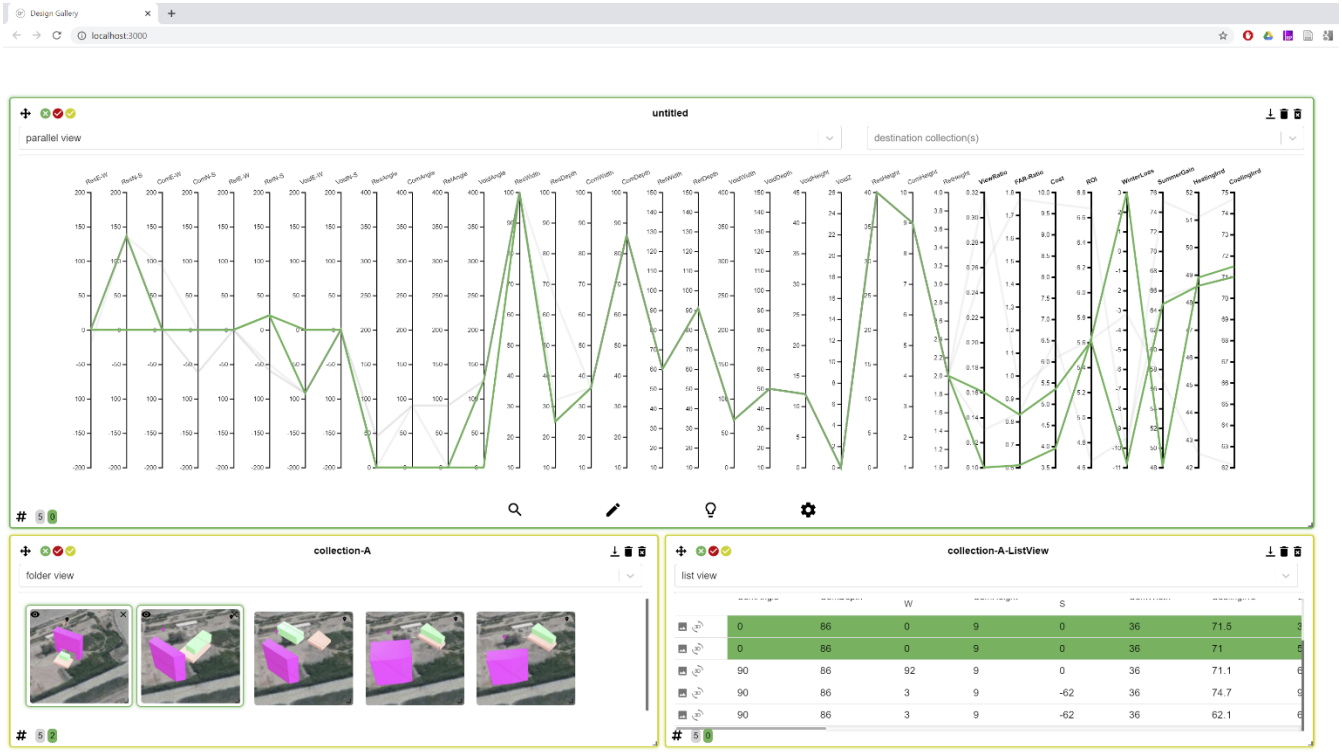

Рисунок 29 – Вигляд моделей у folder, parallel та list view одночасно зі змогою редагування

3.6 Аналіз взаємодії користувача та метрики веб-застосунку

Однією із основних вимог до сучасних веб-додатків є адаптивність. Адаптивний дизайн – це такий дизайн сайту, який здатний забезпечити для користувача зручне і коректне відображення сайту на будь-якому пристрої. Адаптивний сайт буде в автоматичному режимі адаптувати розмір елементів інтерфейсу користувача в залежності від розміру екрану пристрою**.** Користувач не має ніякої необхідності збільшувати чи зменшувати масштаб на екрані пристрою задля зручного використання.

Веб-додаток design gallery розроблений із умовами адаптивності, який може відображатися як і в десктопних, так і в мобільних браузерах.

Також, одними з основних елементів при розробці веб-ресурсу є метрики. Метрика програмного забезпечення – міра, що дозволяє отримати чисельне значення деякої властивості програмного забезпечення або його специфікацій.

Загалом метрики можна розділи за наступними критеріями:

- час завантаження сайту
- час інтерактивності
- час до отримання першого байта
- час початку старту рендера
- загальний розмір файлів контенту
- час до повного завантаженя

Одним із основних критерії оцінювання якості сайту є індекс швидкості. Індекс швидкості - це середній час відображення видимих частин сторінки. Він виражається в мілісекундах і залежить від розміру вікна перегляду.

Метрики були обраховані за допомогою сайтів https://gtmetrix.com/ та https://developers.google.com/speed/pagespeed/insights/

Результати зображені на рисунках нижче.

Рисунок 30 – Обрахунок метрик інтерфейсу користувача за допомогою сайту https://gtmetrix.com/

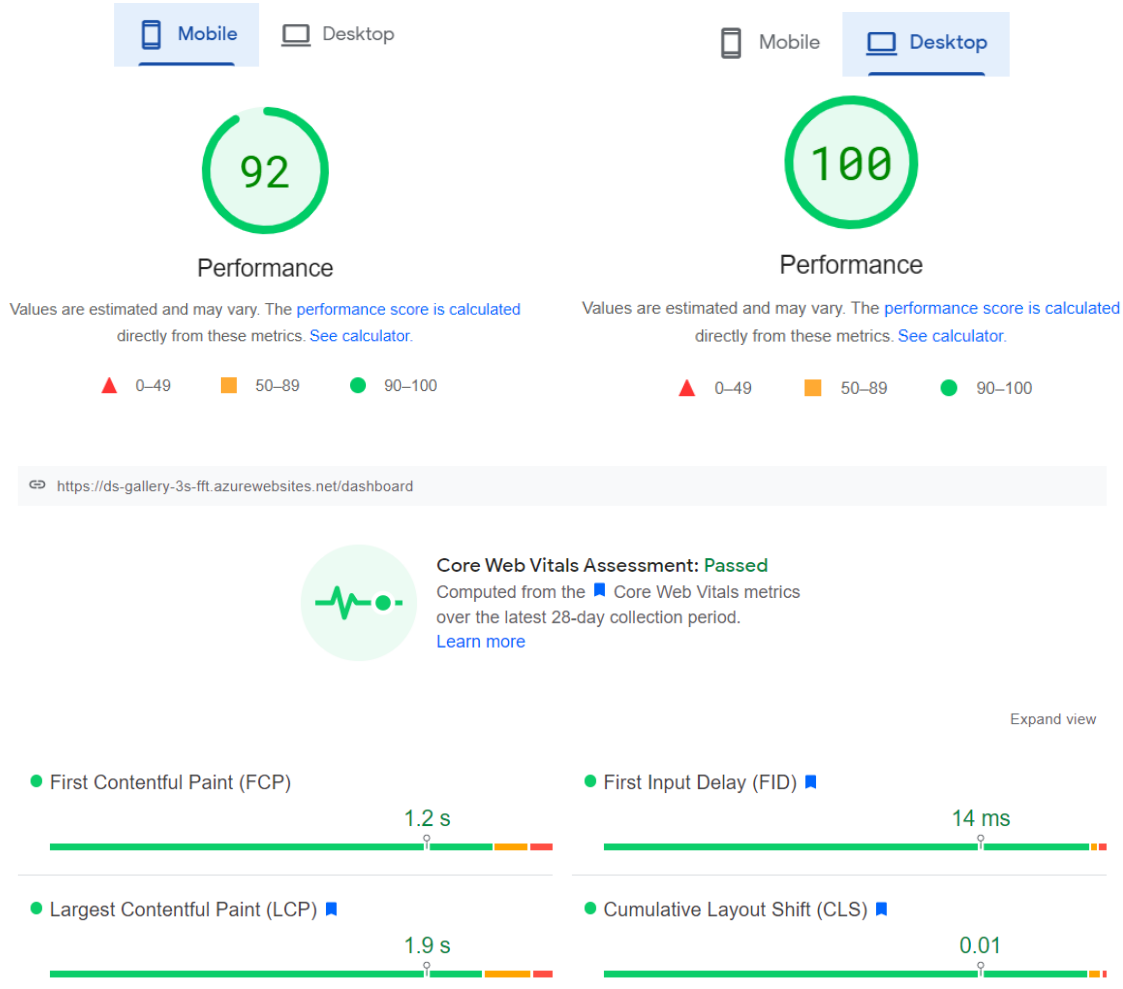

Рисунок 31 – Обрахунок метрик інтерфейсу користувача за допомогою сайту https://developers.google.com/speed/pagespeed/insights/

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено та удосконалено інтегровану систему, до якої входить веб-сервіс для взаємодії з системами для створення 3d-об’єктів та односторінковий веб-інтерфейс, який дозволяє користувачам взаємодіяти з кількома тривимірними об’єктами одночасно у різних представленнях.

Проведено аналіз представлення об’єктів та взаємодії користувачів у веб-додатках, опановано методику й етапи створення каталогів та секцій систем, базуючись на критеріях якості побудови інтерфейсів користувача, що дозволило впровадити нову систему розділів односторінкового веб-застосунку.

Також було реалізовано аутентифікацію для забезпечення надійності даних і покращено алгоритм створення тривимірних моделей на основі вихідних даних, вершин та граней (vertices and faces) з додатків та систем для створення різних 3d-об’єктів, що прискорило швидкодію ресурсу та позитивно вплинуло на метрики веб-застосунку.

За результатами кваліфікаційної роботи було проаналізовано технології 3d-моделювання, враховано знайдені недоліки при проектуванні, розглянуто різні інструменти для реалізації веб-застосування, на основі чого було створено прототип веб-додатку для відображення моделей, що дає змогу користувачу динамічно їх змінювати.

Даний веб-додаток може бути використаний користувачами у галузі архітектури та дизайну для візуального редагування тривимірних моделей засобами веб-інтерфейсу, що дає змогу швидко та ефективно вирішувати поставлені завдання, затрачуючи при цьому менше часу та коштів на реалізацію.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Lunzer, A. and Hornbæk, K. (2008). Subjunctive interfaces: Extending applications to support parallel setup, viewing and control of alternative scenarios. ACM Transactions on Computer-Human Interaction (TOCHI), 14(4):17
2. Zaman, L., Stuerzlinger, W., Neugebauer, C., Woodbury, R., Elkhaldi, M., Shireen, N., and Terry, M. (2015). GEM-NI: A system for creating and managing alternatives in generative design. In Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, pages 1201–1210, New York, NY, USA. ACM.
3. Фридман Л. М. Наглядность и моделирование в обучении. – М.: Знание, 1984. – 80 с.
4. Woodbury, R., Datta, S., and Burrow, A. (2000). Erasure in design space exploration. In Artificial Intelligence in Design 2000, pages 520–545, Worcester, Massachusetts. Key Centre for Design Computing, Kluwer Academic Publishers.
5. Mohiuddin, A.,Woodbury, R., Cichy, M., Mueller, V., and Ashtari, N. (2017). A Design Gallery System: Prototype and Evaluation. In ACADIA 2017: Disciplines & Disruption, Boston, MA. ACADIA. 414-425.
6. Terry, M. and Mynatt, E. D. (2002b). Side views: Persistent, on-demand previews for open-ended tasks. In Proceedings of the 15th Annual ACMSymposium on User Interface Software and Technology, UIST ’02, pages 71–80, New York, NY, USA. ACM.
7. Authentication General Guidelines, [Електронний ресурс] – Режим доступу до ресурсу: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

8. What is Human-Computer Interaction (HCI), [Електронний ресурс] – Режим доступу до ресурсу: https://www.interaction-design.org/literature/topics/human-computer-interaction
9. User Modeling in Human–Computer Interaction, [Електронний ресурс] – Режим доступу до ресурсу: https://link.springer.com/article/10.1023/A:1011145532042
10. Інформація з сайту офіційної документації по MongoDB, [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/
11. Інформація з сайту офіційної документації по Node.js, [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/uk/
12. Інформація з сайту офіційної документації по three.js, [Електронний ресурс] – Режим доступу до ресурсу: https://threejs.org/
13. Інформація з сайту офіційної документації по glTF, [Електронний ресурс] – Режим доступу до ресурсу: https://khronos.org/gltf/
14. Інформація з сайту офіційної документації по Passport.js, [Електронний ресурс] – Режим доступу до ресурсу: https://www.passportjs.org/
15. Інформація з сайту офіційної документації по React.js, [Електронний ресурс] – Режим доступу до ресурсу: https://reactjs.org/
16. Інформація з сайту офіційної документації по Redux, [Електронний ресурс] – Режим доступу до ресурсу: https://redux.js.org/
17. 3D Rendering Market Report Coverage [Електронний ресурс] – Режим доступу до ресурсу: https://www.gminsights.com/industry-analysis/3d-rendering-market

ДОДАТКИ

Додаток А

Приклад даних тривімірної моделі у форматі JSON


```
{
  "hysteresis_mode": "recorded",
  "filepath": "C:\\Users\\Ungfyrto\\server\\GH\\hyst-contained.ghx",
  "params": {
    "v1-posX": 43,
    "v1-posZ": 0,
    "s1-sclX": 50,
    "s1-sclY": 45,
    "s1-sclZ": 21,
    "v1-sclX": 26,
    "v1-sclZ": 57,
    "s1-rotXY": 0.41,
    "s2-posX": 34,
    "s2-posY": 22,
    "s2-posZ": 0,
    "s3-sclX": 23,
    "v3-rotXY": 0,
    "s3-sclY": 42,
    "v1-rotXY": -0.79,
    "v3-posY": 18,
    "v3-posX": 4,
    "s3-rotXY": -0.32,
    "v3-posZ": 31,
    "s2-rotXY": 0.26,
    "v1-posY": 27
```

```
  },
  "provenance": [
    "e3773e62-42d2-4c1e-a857-8c09daec9639",
    "e3773e62-42d2-4c1e-a857-8c09daec9639",
    "e3773e62-42d2-4c1e-a857-8c09daec9639",
    "e3773e62-42d2-4c1e-a857-8c09daec9639"
  ],
  "Output": {
    "SurfaceArea": "2757"
  },
  "partName": "D",
  "cad_file": "f73c0eb8-75f3-47f2-92e8-ee339060c0e2.obj",
  "image": "f73c0eb8-75f3-47f2-92e8-ee339060c0e2.bmp",
  "uid": "f73c0eb8-75f3-47f2-92e8-ee339060c0e2",
  "model_name": "massing",
  "RhinoFile": " C:\\Users\\Ungfyrto\\server\\GH\\siteA.3dm",
  "position": {
    "x": 3342,
    "y": 230
  }
}
```

Додаток Б

Програмний код створення 3d-моделі у форматі .glTF

```
var gltfTable = {};
gltfTable.createGltf = async(geometry) => {
    global.btoa = require("btoa");

    const GLTFUtils = require("gltf-js-utils");
    const fs = require("fs");

    const asset = new GLTFUtils.GLTFAsset();
    const scene = new GLTFUtils.Scene();
    asset.addScene(scene);

    const node = new GLTFUtils.Node();
    scene.addNode(node);

    const faces = geometry.faces;
    const vertices = geometry.vertices;

    const vertexArray = [];
    for (let i = 0; i < vertices.length; i += 3) {
      const vertex = new GLTFUtils.Vertex();
      vertex.x = vertices[i];
      vertex.y = vertices[i + 1];
      vertex.z = vertices[i + 2];
      vertexArray.push(vertex);
    }
```

```
  const mesh = new GLTFUtils.Mesh();
  mesh.material = [new GLTFUtils.Material()];

  for (let i = 0; i < faces.length; i += 3) {
    const v1 = vertexArray[faces[i]];
    const v2 = vertexArray[faces[i + 1]];
    const v3 = vertexArray[faces[i + 2]];
    mesh.addFace(v1, v2, v3, {r: 220, g: 220, b: 220}, 0);
  }
  node.mesh = mesh;

  const gltfFiles = await GLTFUtils.exportGLTF(asset, {
    bufferOutputType: GLTFUtils.BufferOutputType.DataURI,
    imageOutputType: GLTFUtils.BufferOutputType.DataURI,
  });
};

module.exports = gltfTable;
```

Додаток В

Частина програмного коду використовуючи локальну стратегію

```
const router = require('express').Router(),
  passport = require('passport'),
  LocalStrategy = require('passport-local').Strategy,
  { isEqual } = require('lodash')

const localUser = {
  username: 'vladyslav.sosnov',
  password: 'testPassword1*'
}

passport.use(new LocalStrategy((username, password, done) => {
    return isEqual(localUser, { username, password })
      ? done(null, localUser) // success
      : done(null, false) // failure
  }
))

// 'local' strategy
router.get('/', passport.authenticate('local', {
  session: true,
  successRedirect: '/dashboard',
  failureRedirect: "/login?=incorrect username/password"
}))

module.exports = router
```

Додаток Г

Частина програмного коду використовуючи стратегію Google OAuth2

```
const router = require('express').Router(),
  passport = require('passport'),
  GoogleStrategy = require('passport-google-oauth').OAuth2Strategy,
  env = require('../env')

passport.use(new GoogleStrategy(
  {
    clientID: env.GOOGLE_CLIENT_ID,
    clientSecret: env.GOOGLE_CLIENT_SECRET,
    callbackURL: 'http://localhost:8000/auth/google/verify'
  },
  (accessToken, refreshToken, profile, done) => {
      return profile && profile.emails && profile.emails[0].value
        ? done(null, { email: profile.emails[0].value }) // success
        : done(null, false) // failure
  }
))

// scope request for what info is needed
router.get('/', passport.authenticate('google', { scope:
'https://www.googleapis.com/auth/userinfo.email' }))
router.get('/verify', passport.authenticate('google', { session: false
}),
  (req, res, next) => {
    // the information sent from passport sent in the request as user
    const { user } = req
    if (!user) {
```

```
      return res.redirect('/?google_authorization_unsuccessful')
    }
    res.redirect(`/?authorized_as_${user.email}`)
  })

module.exports = router
```

Додаток Д

Програмний код рендерингу тривимірної моделі

```
import React, {Component} from 'react';
import * as THREE from 'three';
import * as GLTFLoader from 'three-gltf-loader';
import styles from './scene.module.scss';

const OrbitControls = require('three-orbit-controls')(THREE);

class Scene extends Component {
  constructor(props) {
    super(props);
    this.animate = this.animate.bind(this);
    this.initializeCamera = this.initializeCamera.bind(this);
    this.initializeOrbits = this.initializeOrbits.bind(this);
  }

  componentDidMount() {
    this.scene = new THREE.Scene();
    this.scene.background = new THREE.Color(0xeded ed);

    this.camera = new THREE.PerspectiveCamera(80, 1, 0.001, 1000);
    this.scene.add( this.camera );

    this.renderer = new THREE.WebGLRenderer({antialias: true});
    this.controls = new OrbitControls(this.camera,
this.renderer.domElement);
```

```
    this.renderer.setSize(100, 100);
    this.renderer.toneMapping = THREE.ReinhardToneMapping;

    this.mount.appendChild(this.renderer.domElement);
    this.initializeOrbits();

    this.loader = new GLTFLoader();
    this.loader.load( 'data/structureBlack.gltf', (gltf) => {

      this.scene.add(gltf.scene);
      this.scene.add( new THREE.AmbientLight( 0xd60000) );

      this.scene.traverse( ( child ) => {
        if ( child instanceof THREE.Mesh ) {
          child.geometry.center();
          child.material = new THREE.MeshPhongMaterial( {
            color: 0x4f0303,
            emissive: 0x000000,
            shading: THREE.FlatShading,
            polygonOffset: true,
            polygonOffsetFactor: 1,
            polygonOffsetUnits: 1,
          } );

          let geo = new THREE.WireframeGeometry( child.geometry );
          let mat = new THREE.LineBasicMaterial( {color: 0x000000,
linewidth: 1} );
          let wireframe = new THREE.LineSegments( geo, mat );
          child.children.push(wireframe);
```

```
        child.castShadow = true;
        child.receiveShadow = true;
      }
      this.initializeCamera();
    });
  });

  this.animate();

  window.onresize = () => {
    const width = window.innerWidth;
    const height = window.innerHeight;
    this.camera.aspect = width / height;
    this.camera.updateProjectionMatrix();
    this.renderer.setSize(width, height);
  };
}

componentWillUnmount() {
  cancelAnimationFrame(this.frameId);
  this.mount.removeChild(this.renderer.domElement);
}

initializeOrbits() {
  this.controls.rotateSpeed = 1.0;
  this.controls.zoomSpeed = 1.2;
  this.controls.panSpeed = 0.8;
}
```

```
  initializeCamera() {
    const object = this.scene.children[1].children[0];
    const offset = 20;
    const boundingBox = new THREE.Box3();
    boundingBox.setFromObject( object );

    const size = boundingBox.getSize();

    const maxDim = Math.max( size.x, size.y, size.z );
    const fov = this.camera.fov * ( Math.PI / 180 );

    let cameraZ = Math.abs( maxDim / 4 * Math.tan( fov * 2 ) );
    cameraZ *= offset;
     this.camera.position.z = cameraZ;

    const minZ = boundingBox.min.z;
    const cameraToFarEdge = ( minZ < 0 ) ? -minZ + cameraZ : cameraZ -
minZ;

    this.camera.far = cameraToFarEdge * 3;
    this.camera.updateProjectionMatrix();
  }

  animate() {
    this.frameId = window.requestAnimationFrame(this.animate);

    this.renderer.setPixelRatio( window.devicePixelRatio );
    this.renderer.gammaOutput = true;
    this.renderer.gammaFactor = 2.2;
```

```
    this.renderer.shadowMap.enabled = true;
    this.renderer.shadowMap.type = THREE.PCFSoftShadowMap;

    this.renderer.render(this.scene, this.camera);
 }

  handleControls(e) {
    e.stopPropagation();
  }

  render() {
    return (
      <div className={styles['render-box']}
        onMouseDownCapture={this.handleControls}
        id="boardCanvas"
        ref={(mount) => {
          this.mount = mount;
        }}
      />
    );
  }
}

export default Scene;
```