КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра інтелектуальних технологій

Кваліфікаційна робота

на здобуття ступеня бакалавра

на тему:

«Розробка веб-інструментарію моделювання повітряного потоку для зниження ризику розповсюдження вірусів в приміщенні»

Галузь знань: **12 «Інформаційні технології»**

Спеціальність: **122 «Комп'ютерні науки»**

Освітня програма: «**Комп'ютерні науки»**

Виконав: студент 4-го курсу, групи КН-41

Овсійчук Петро В'ячеславович _____

Науковий керівник:

Асистент кафедри інтелектуальних технологій

факультету інформаційних технологій

Андрійчук Олег Валентинович _____

Випускна кваліфікаційна робота бакалавра

допущена до захисту рішенням

кафедри інтелектуальних технологій

Протокол № 11 від 06.06.2022 р.

Завідувач кафедри Іларіонов О.Є. _____

Київ - 2022

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра інтелектуальних технологій

Спеціальність 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри
інтелектуальних технологій
Іларіонов О.Є.

_____

"___"_____2022 р.

ЗАВДАННЯ

НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Овсійчуку Петру В'ячеславовичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи)

Розробка веб-інструментарію моделювання повітряного потоку для зниження ризику розповсюдження вірусів в приміщенні

затверджена протоколом засідання кафедри від « 23 » грудня 2021 р. № 4

2. Термін здачі студентом закінченого проекту (роботи) « 29 » травня 2022 року

3. Вихідні дані до проекту (роботи)
Візуалізація повітряних потоків в змодельованому приміщенні

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
Огляд задачі, можливі підходи до вирішення, принцип роботи алгоритму, переваги та недоліки алгоритму, вибір архітектури системи, клієнтська частина, серверна частина, програмна реалізація системи, архітектура системи, тестування

5. Перелік презентаційного матеріалу (з точним зазначенням обов'язкових презентацій)
Презентація (Power Point), коротке відео з демонстрацією візуалізації програми. Презентація повинна містити в собі: короткі відомості про задачу, загальний опис алгоритму, переваги та недоліки алгоритму, розробка системи (серверна, клієнтська частини), архітектура системи, програмна реалізація, рисунки, що демонструють роботу програмної частини.

6. Консультанти з випускної кваліфікаційної роботи із зазначенням розділів випускної кваліфікаційної роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 15 лютого 2022 року

Керівник _____ (підпис) / Андрійчук О. В. (ПІБ) /

Завдання прийняв до виконання _____ (підпис) / Овсійчук П. В. (ПІБ) /

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів випускної кваліфікаційної роботи	Примітка
1	Вибір керівника	25.01.2022 – 29.01.2022	
2	Вибір теми, формулювання, затвердження	30.01.2022 – 03.02.2022	
3	Аналіз предметної області	04.02.2022 – 08.02.2022	
4	Створення програмної частини	09.02.2022 – 13.04.2022	
5	Опис першого розділу (Модель задачі)	14.04.2022 – 30.04.2022	
6	Опис другого розділу (Опис алгоритму)	01.05.2022 – 10.05.2022	
7	Опис третього розділу (Опис системи)	11.05.2022 – 30.05.2022	
8	Передзахист дипломної роботи	04.05.2022	
9	Захист дипломної роботи	24.06.2022	

Студент-дипломник _____ (підпис) / Овсійчук П. В. (ПІБ) /

Керівник випускної кваліфікаційної роботи _____ (підпис) / Андрійчук О. В. (ПІБ) /

Анотація

Овсійчук Петро В'ячеславович виконав випускну кваліфікаційну роботу на тему «Розробка веб-інструментарію моделювання повітряного потоку для зниження ризику розповсюдження вірусів в приміщенні».

У випускній кваліфікаційній роботі визначено основні фактори, які впливають на розповсюдження вірусів у приміщенні; створено веб-інструментарій для симуляції повітряних потоків всередині змодельованих приміщень.

Ключові слова: Повітряні потоки в приміщеннях, обчислювальна гідродинаміка, SARS-CoV-2, CFD, Web Application.

Abstract

Ovsiychuk Petro Vyacheslavovych completed the final bachelor`s project on “Development of web tool for airflow modeling to reduce the risk of spreading viruses indoors”

This bachelor`s work identifies the main factors that affect the spread of viruses in the room; created web tool to simulate airflow inside the created rooms.

Keywords: Indoor airflows, computational hydrodynamics, SARS-CoV-2, CFD, Web Application.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ОС - Операційна Система

CFD (Computational fluid dynamics) - обчислювальна гідродинаміка

CI/CD (Continuous integration, Continuous deployment) - комбінація безперервної інтеграції та безперервного розгортання

UI (User interface) - інтерфейс користувача

Frontend - клієнтська частина веб застосунку

Backend - серверна частина веб застосунку

HTML (HyperText Markup Language) - мова розмітки гіпертексту

DOM (Document Object Model) - об'єктна модель документа

SEO (Search Engine Optimization) - пошукова оптимізація сайту

CSS (Cascading Style Sheets) – мова стилю сторінок

REST (Representational State Transfer) – передача репрезентативного стану

Payload – фактичні дані в повідомленні або запиті

Endpoint – кінцева точка, кінець каналу зв'язку

ВСТУП

На сьогодні, тема забезпечення приміщень хорошою вентиляцією є досить актуальною. У зв'язку з виявленням нових вірусів, таких, як SARS-CoV-2, та його мутаціями: “501.V2 Variant” [19], “Кластер 5” [20], “VOC-202012/01” [21], “Omicron variant” [22, 23], “V. 1.640.2” [24], весь світ був вимушений змінити поточний режим життя, та взаємодію з предметами, що нас оточують в громадських місцях через ризик зараження.

CDC (Центри з контролю та профілактики захворювань у США) рекомендують багатоступеневий підхід для зменшення впливу SARS-CoV-2 на людей, що в першу чергу включає в себе покращення вентиляції будівлі для зменшення ризику поширення захворювання. [8, 9] У більшості випадків, заселення будівель під час пандемії COVID-19 не потребує нових систем вентиляції будівлі. Проте оновлення або вдосконалення систем вентиляції можуть збільшити подачу чистого повітря та розбавити потенційні забруднювачі. [8]

Тому наразі є актуальним створення веб-інструментарію, який слугував би засобом для моделювання приміщень, а також для симуляції повітряного потоку всередині цих приміщень, що допоможе проаналізувати недоліки у системах вентиляції.

РОЗДІЛ 1 МОДЕЛЬ ЗАДАЧІ ПО РОЗРОБЦІ ВЕБ-ІНСТРУМЕНТАРІЮ ДЛЯ МОДЕЛЮВАННЯ ПОВІТРЯНИХ ПОТОКІВ

1.1 Аналіз області застосування веб-інструментарію

Областю застосування розроблюваної системи є сфера будівництва та планування вентиляційних систем. А саме перевірка ефективності роботи, проектування нових або покращення вже готових вентиляційних систем для зменшення ризику поширення вірусів за допомогою візуалізованої симуляції повітряних потоків у приміщенні. Веб-інструментарій дозволить користувачу створити, зберегти та поширити проект для роботи з цільовим приміщенням. За допомогою інтерактивного конструктору користувач зможе побудувати план будівлі на основі фундаментальних конструкцій (стіни, двері, вікна), та конструкцій вентиляційних систем (вентиляційні решітки, вентилятори, джерела чистого повітря). На основі побудованого плану клієнт зможе виконати моделювання повітряного потоку для свого проекту з інтуїтивною візуалізацією та візуально проаналізувати проблеми у вентиляційній системі.

1.2 Аналіз впливу вентиляції на розповсюдження вірусів у приміщенні

Згідно з дослідженнями федеральної агенції міністерства охорони здоров'я США, CDC (центри контролю та профілактики захворювань у США), SARS-CoV-2 (коронавірус) – це оболонковий вірус, що означає, що його генетичний матеріал упакований всередині зовнішнього шару (оболонки) білків і ліпідів. Оболонка для SARS-Cov-2 містить структури (шиповані білки) для прикріплення

до клітин людини під час інфекції (рисунок 1.1). Оболонка для SARS-CoV-2, як і для інших респіраторних вірусів із оболонкою, є лабільною, і може швидко руйнуватися при контакті з поверхнево-активними речовинами, що містяться в засобах для чищення, та в умовах навколишнього середовища. Ризик передачі, опосередкованої такими типами вірусу, залежить в першу чергу від:

- Рівню поширеності інфекції в суспільстві
- Кількості інфікованих вірусом людей
- Осідання викинутих вірусних частинок на поверхні, на які впливає потік повітря та вентиляція
- Взаємодія з факторами навколишнього середовища

Також найвищим серед існуючих способом передачі вірусу є прямий контакт, або повітряно-крапельний шлях [2, 4].

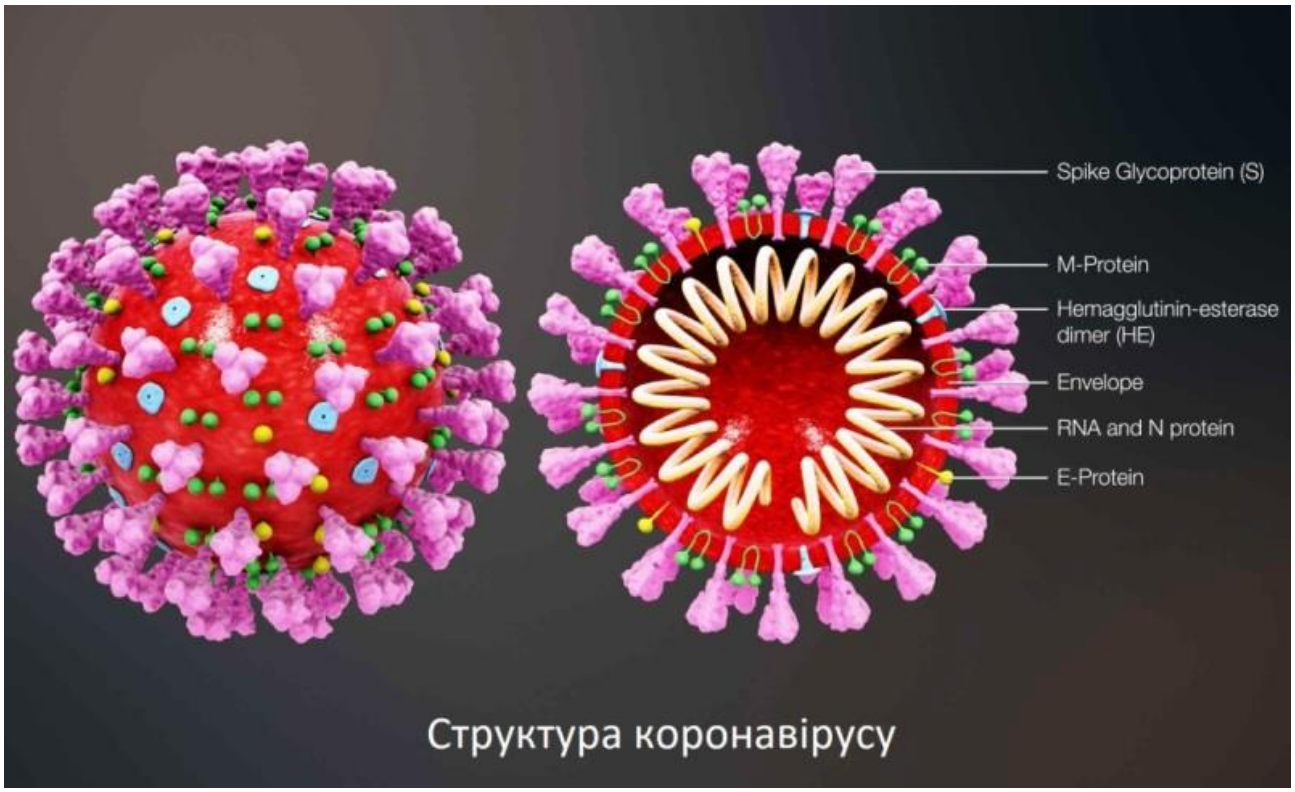

Рисунок 1.1 Структура вірусу SARS-CoV-2 [33]

Якщо людина з підозрою або підтвердженням COVID-19 перебувала у приміщенні, вірус може залишатися у повітрі від кількох хвилин до кількох

годин. Тривалість часу, протягом якого вірус залишається у повітрі та залишається інфекційним, залежить від багатьох факторів, включаючи порушення потоку повітря, вентиляцію, температуру та вологість. [3, 5, 6]

Отже, люди можуть заразитися SARS-CoV-2 при контакті з поверхнями. Однак, виходячи з наявних епідеміологічних даних та досліджень факторів передачі через навколишнє середовище, поверхнева передача не є основним шляхом поширення SARS-CoV-2, і ризик вважається низьким. Основним способом зараження людей SARS-CoV-2 є вплив респіраторних капель, що переносять інфекційний вірус. [7]

На основі цих даних можемо зробити висновок, що якісна вентиляція у приміщеннях є однією з головних аспектів зниження ризику передачі респіраторних захворювань.

1.3 Можливі підходи до вирішення проблеми та конкуренти

Вартість і продуктивність будь-якого фізичного продукту, як правило, визначаються досить рано в процесі проектування — те ж саме стосується проектування вентиляційної системи (рисунок 1.2). На етапі початку дослідження дизайну вентиляційної системи та визначенні концепції свого приміщення, приймаються найбільш ефективні дизайнерські рішення. Після цього темпи реалізації виробничих витрат значно нижчі.

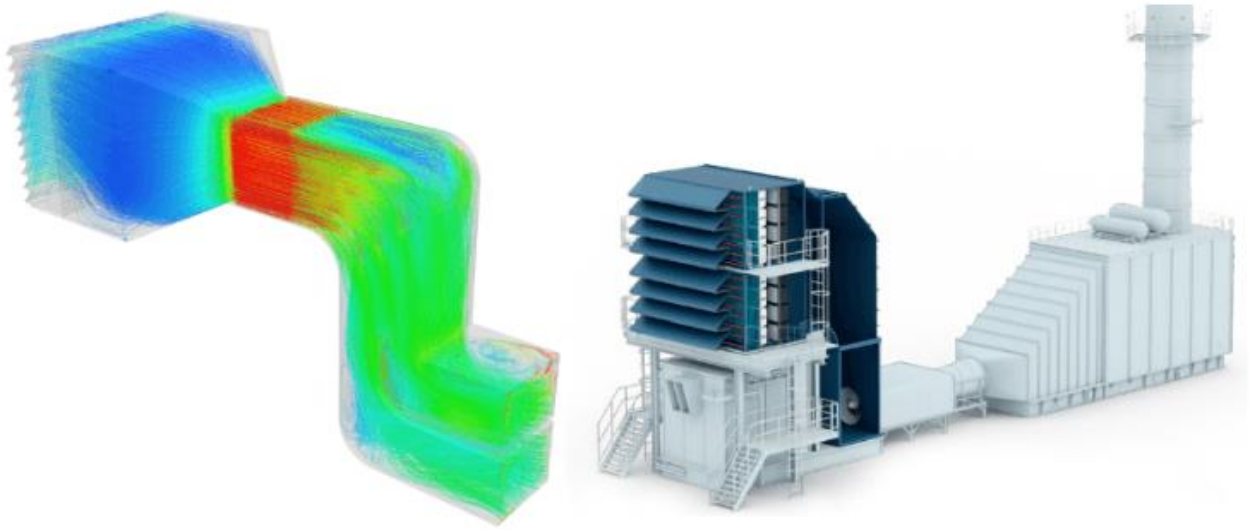

Рисунок 1.2 Проектування вентиляційної системи за допомогою програмного забезпечення з CFD [34]

Зрештою, набагато дешевше мати інженера-конструктора, який працює на комп’ютері, ніж виконувати польові випробування зі створенням фізичних прототипів. Моделювання є одним із інструментів, які відіграють основну роль на цих ранніх етапах розробки продукту, дозволяючи інженерам приймати більш обґрунтовані дизайнерські рішення на початку процесу та знижуючи загальні витрати. Для кінцевого продукту це може означати нижчі виробничі витрати, більш ефективне споживання енергії, менший ризик відмови тощо. [34]

Однією з найпопулярніших пропозицій на ринку є програма SimScale. SimScale – це потужний засіб для моделювання потоків, що стискаються або не стискаються, ламінарних, багатофазних та турбулентних потоків. Також система включає в себе симуляцію механіки твердого тіла (статична, динамічна, модальний аналіз, динаміка декількох тіл тощо), термодинаміку (провідність, конвекція, випромінювання). Інтерфейс програми є досить складним, та не зрозумілим для новачка (рисунок 1.3).

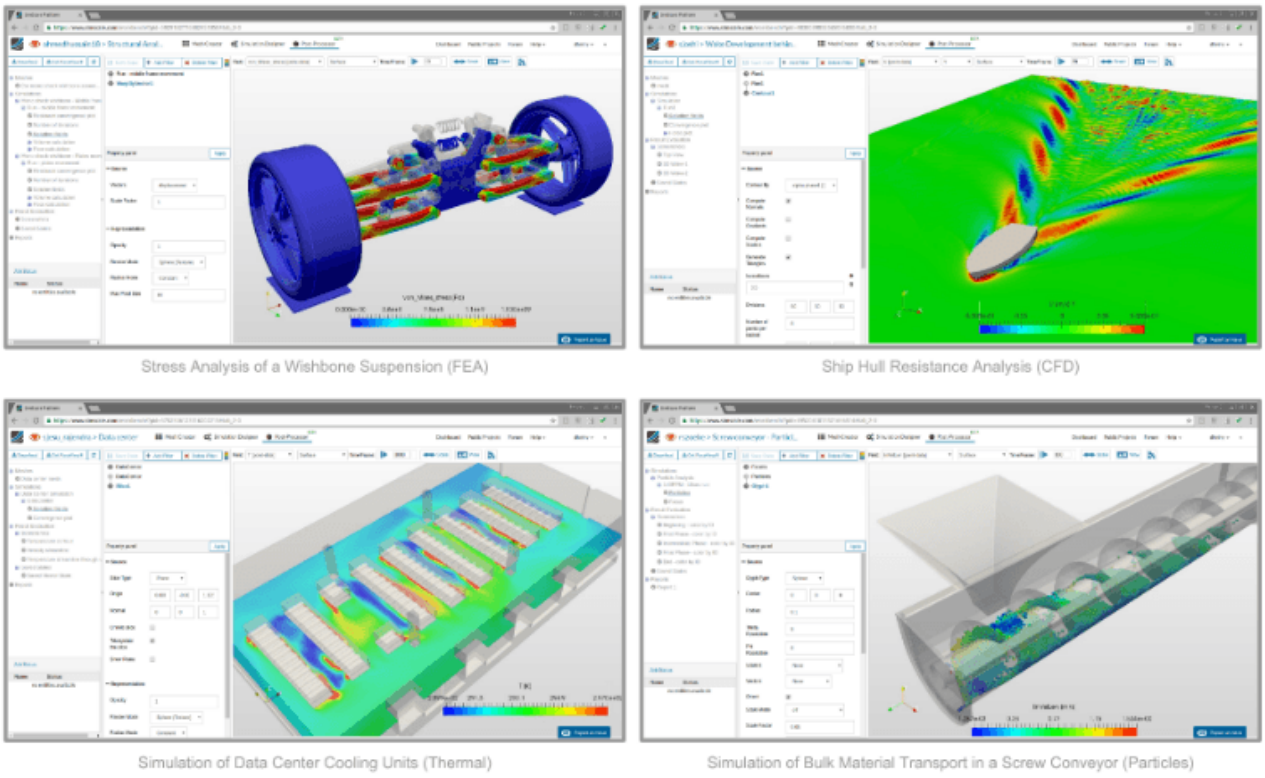

Рисунок 1.3 Інтерфейс програми SimScale

Хоч SimScale і має великі переваги у функціоналі, рядова людина не може собі дозволити ліцензійний пакет цієї системи, а безкоштовна версія є дуже обмеженою у функціоналі. Навіть самий базовий пакет доступу обійдеться клієнту у 2 тисячі доларів (рисунок 1.4). Ціна зумовлюється тим, що система надає можливість проводити хмарні обчислення замість локальних .

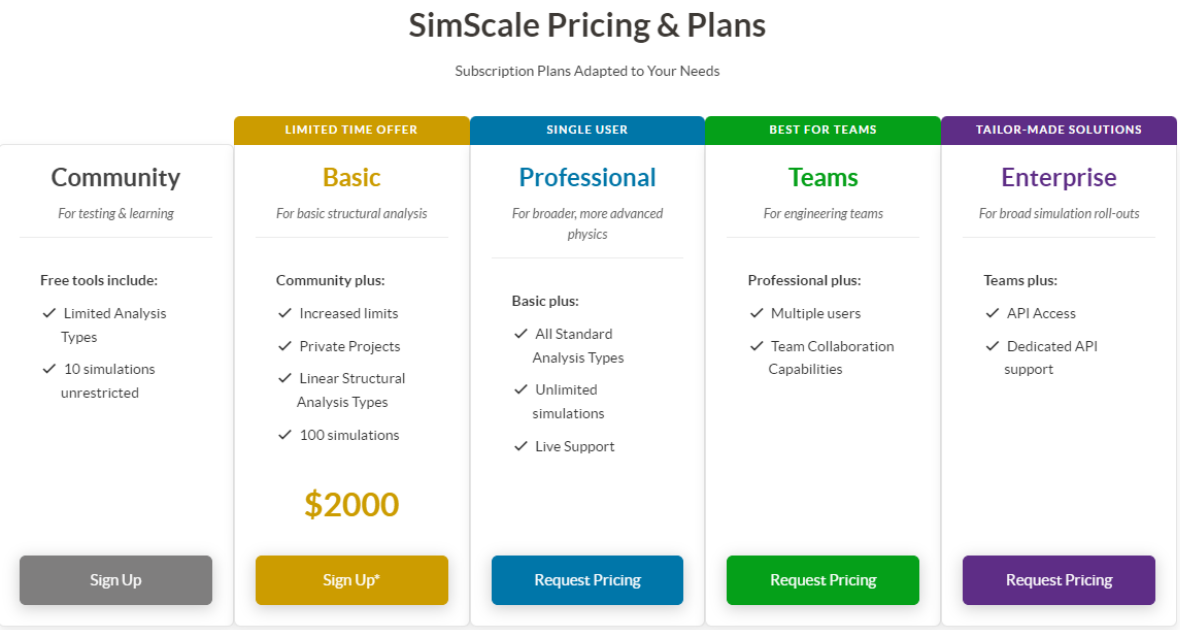

Рисунок 1.4 Вартість пакетів SimScale

Проте у зв'язку з епідеміологічною ситуацією практично кожна людина, котра має своє приміщення, зацікавлена в якісній вентиляційній системі. Тому виникла ідея створити безплатний веб-інструментарій для симулювання повітряних потоків в приміщеннях, що зможе запуститись навіть на телефоні, і не буде потребувати великих обчислень.

Головною проблемою оптимізації в задачах подібного типу є вибір алгоритму обчислювальної гідродинаміки (CFD алгоритму). Проаналізувавши декілька варіантів було вирішено, що задля прискорення розробки та маленької ресурсоємності симуляції, слід зупинитись на алгоритмі “Real-Time Fluid Dynamics for Games” [1], оскільки він використовує не складне розв'язування рівнянь Нав'є – Стокса методом Гауса – Зайделя та використовується для симуляції рідин в іграх. Тобто акцент роботи алгоритму робиться на стабільності та швидкості. Алгоритм з легкістю працює в режимі реального часу в розумних межах роздільної здатності сітки, в двох або трьох площинах (рисунок 1.5).

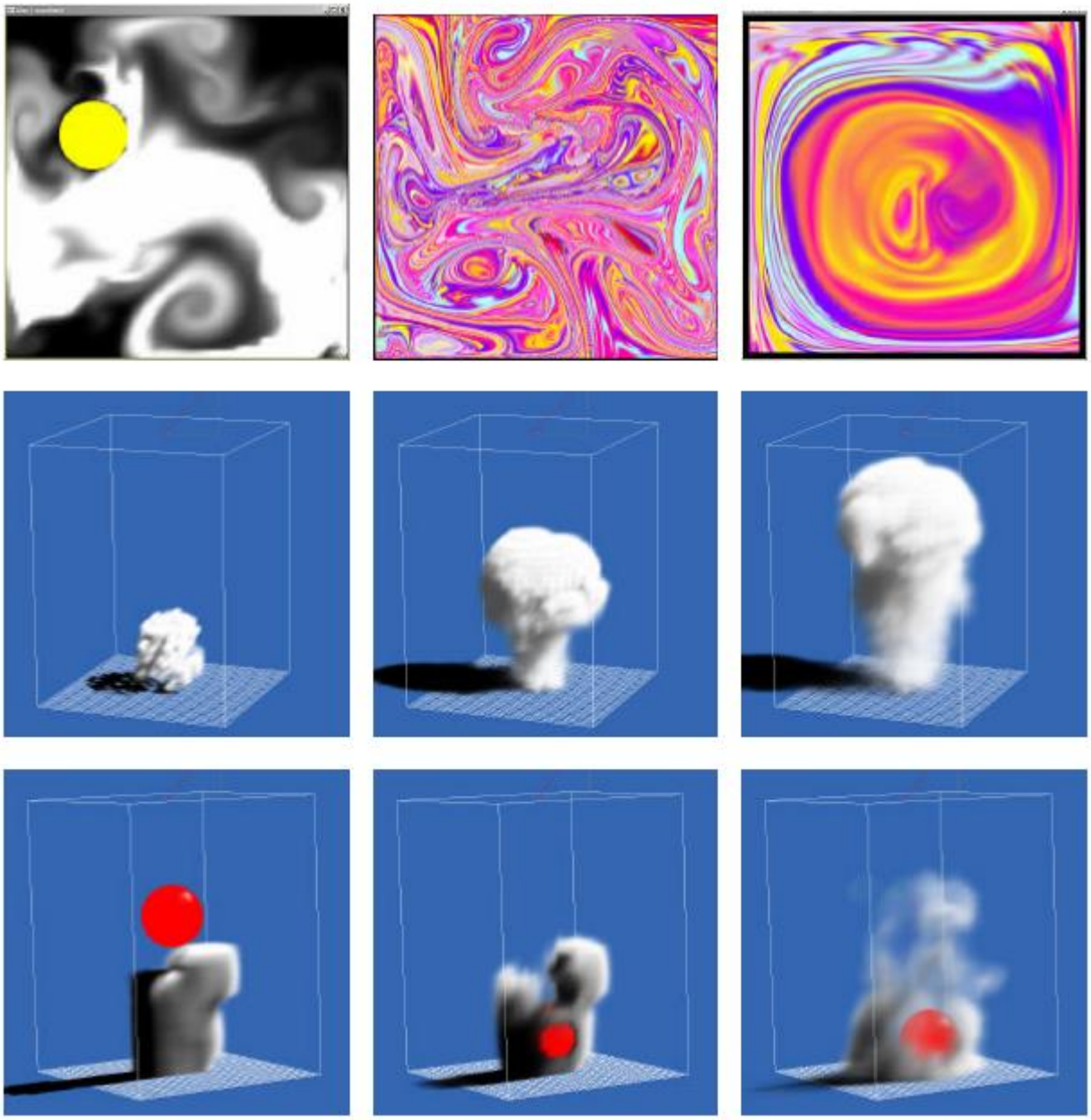

Рисунок 1.5 Приклад роботи алгоритму “Real-Time fluid dynamics for Games” [1]

1.4 Постановка задачі по створенню веб-інструментарію

Об’єктом дослідження є фактори, що впливають на розповсюдження вірусів та небезпечних захворювань в приміщенні.

Предметом дослідження є веб-інструментарій, який буде надавати користувачеві можливість для моделювання приміщень, та симулювання руху повітряних потоків.

Метою дипломної роботи є визначення факторів, які впливають на розповсюдження вірусів у приміщенні; розробка веб-інструментарію для моделювання потоків руху повітря у приміщенні.

Зацікавлені сторони:

- Архітектори лікарень, магазинів та будь-яких громадських місць;
- Люди, що вже мають власний будинок, або хочуть спроектувати чи поновити систему вентиляції у будинку.

В процесі дослідження буде обрано оптимальний алгоритм обчислювальної гідродинаміки. Наступна задача буде полягати у реалізації веб-застосунку, що дозволить користувачу безперешкодно зайти на веб-сайт зі зручним та сучасним інтерфейсом, створити проект свого приміщення за допомогою зручного інтерактивного конструктору та проаналізувати роботу вентиляції, запустивши симуляцію повітряного потоку і переглянувши інтуїтивно зрозумілу візуалізацію цього процесу у режимі реального часу.

Задачі дипломного проектування:

- Дослідити основні причини розповсюдження вірусів у приміщенні;
- Дослідити інформацію про вплив якості вентиляції на розповсюдження вірусів у приміщенні;
- Розробити веб-інструментарій моделювання повітряного потоку для зниження ризику розповсюдження вірусів в приміщенні;
- Розробити серверну частину та базу даних для збереження користувацьких проектів веб-інструментарію у хмарне середовище;
- Розгорнути створений застосунок.

Функціональні вимоги:

- Веб-інструментарій в цілому, та особливо алгоритм симуляції повітряних потоків повинен бути оптимізованим, та працювати швидко навіть в браузерах мобільних телефонів.

– Надійна серверна частина, та база даних, що зможуть швидко зберігати, добавляти, оновлювати, та віддавати проекти.

– Надати можливість користувачам ділитись проектами один з одним.

– Сучасний, швидкий користувацький інтерфейс (розроблений за допомогою новітніх бібліотек та фреймворків)

– Швидке маніпулювання, та обробка даних, що вводить користувач.

Нефункціональні вимоги:

– Надання можливості користувачу швидко створити проект з планом приміщення, та почати роботу по аналізу повітряних потоків.

– Зручний, інтерактивний та інтуїтивний конструктор плану приміщення, що вміщає в себе налаштування проекту, засоби маніпулювання об'єктами та симуляцією, інструменти для створення основних компонентів приміщення (стіни, вікна, двері, вентиляційні решітки, вентилятори та джерела чистого повітря). Також необхідно розробити засоби маніпулювання повітряними потоками вручну (додавання потоку повітря, керування рухом потоку повітря) та кнопки для запуску або зупинки симуляції.

– Можливість зручного створення, збереження, оновлення, та подальшого відкриття проекту для продовження роботи.

– Каталог з існуючими проектами, та зручним пошуком.

Представимо застосунок у вигляді IDEF0 діаграми (рисунок 1.6).

Веб-інструментарій буде мати наступні вхідні дані:

– Налаштування атрибутів повітря (в'язкість, дифузія);

– Масив з фундаментальними елементами будівлі (стіни, двері, вікна);

– Масив з елементами для симуляції вентиляційної системи (вентиляційні решітки, вентилятори, джерела чистого повітря);

На виході у веб-інструментарію будуть результати моделювання повітряного потоку у реальному часі у вигляді візуалізації.

Рисунок 1.6 – IDEF0 діаграма

На вхід системи будуть подаватись налаштування атрибутів повітря, задані користувачем, задля більшої гнучкості в можливостях симуляції. На моделювання повітряного потоку будуть впливати норми вентиляції [35], закон про захист інтелектуальних даних та потужність пристрою користувача.

Фундаментальні елементи будівлі та елементи вентиляційної системи будуть спроектовані користувачем за допомогою влаштованого конструктору проектів.

Отже, під час виконання першого розділу дипломної роботи було проведено аналітичний огляд дипломної роботи, проаналізовано область впливу якості вентиляції на розповсюдження вірусів у приміщенні, розглянуті можливі підходи до вирішення, визначені зацікавлені сторони, виявлено проблемні моменти і визначено мету кваліфікаційної роботи. Проаналізовано існуючі підходи, рішення та результати, що досягнуто на даний момент часу.

Сформульовано об’єкт, предмет та мету дослідження кваліфікаційної роботи. Визначено функціональні та нефункціональні вимоги. Представлено систему у вигляді IDEF0 діаграми.

Були проаналізовані існуючі рішення щодо проблеми моделювання систем вентиляції в приміщеннях. У результаті аналізу алгоритмів для симуляції повітряних потоків у реальному часі самим ефективним виявився алгоритм Jos Stam`a [1], який можна буде використати у подальшій роботі. Було виявлено декілька ефективних рішень при аналізі майбутнього конструктору для проектування приміщень та візуалізації повітряних потоків.

РОЗДІЛ 2 ОПИС АЛГОРИТМУ

2.1 Принцип роботи

2.2.1 Рівняння Нав'є – Стокса

Майже всі методи моделювання рідин містять в собі рівняння Нав'є – Стокса. Ці рівняння описують рух в'язких рідин або газу. Рівняння названі на честь французького інженера та фізика Клода-Луї Нав'є та ірландського фізика і математика Джорджа Габріель Стокса. Вони розроблялися протягом кількох десятиліть поступово: з 1882 (Нав'є) до 1843-1850 (Стокс).

Два найвідоміших рівняння Нав'є - Стокса виглядають наступним чином:

$$\Delta \cdot u = 0, \tag{2.2.1.1}$$

де u - швидкість;

$$p\frac{Du}{Dt} = -\nabla_P + \mu\nabla^2 u + pF, \tag{2.2.1.2}$$

де $p\frac{Du}{Dt}$ - прискорення рідини;

∇_P - градієнт тиску;

$\mu\nabla^2 u$ – в′язкість;

pF – зовнішні сили;

Перше рівняння (2.2.1.1) стверджує, що розбіжність швидкості дорівнює нулю. Це означає, що швидкості в сусідніх областях рідини не можуть текти назустріч одна одній, або віддалятися одна від одної. Для моделювання рідини це б означало, що рідина з'являлась би з пустого місця, або текла в пусте місце. По суті це рівняння гарантує, що рідина зберігає свою початкову масу.

Друге рівняння (2.2.1.2) стверджує, що прискорення рідини залежить від внутрішніх сил (градієнт тиску, в'язкість) та на зовнішніх силах. Градієнт тиску

відповідає за те, що якщо рідина в різних областях має різницю в тиску, вона буде прагнути до руху з боку високого тиску в бік низького. Вища в'язкість означає те, що рідина має більше значення тертя між її частинками, і швидкість рідини в одній області буде гірше поширюватися на навколишні області. Зовнішні сили включають в себе такі речі, як: гравітація, стіни, взаємодія з об'єктами, що знаходяться всередині речовини.

Ці рівняння пояснюють деякі фундаментальні закони фізики рідин, і є дуже важливими в симуляції рідин, проте вони дають тільки специфікацію, а не реалізацію. Іншими словами, рівняння Нав'є – Стокса говорять нам про те, яких правил має дотримуватись симуляція рідини, але вони ніяк не покривають те, як насправді має виглядати процес моделювання.

2.2.2 Представлення

Є кілька різних способів представлення рідини в комп'ютерному моделюванні. А саме: моделювання може використовувати область з частинками, що рухаються (квадрат зліва на рисунку 2.1), або сітку стаціонарних областей з атрибутами, які представляють середнє значення всіх уявних частинок, які повинні бути у кожному квадраті (квадрат справа на рисунку 2.1).

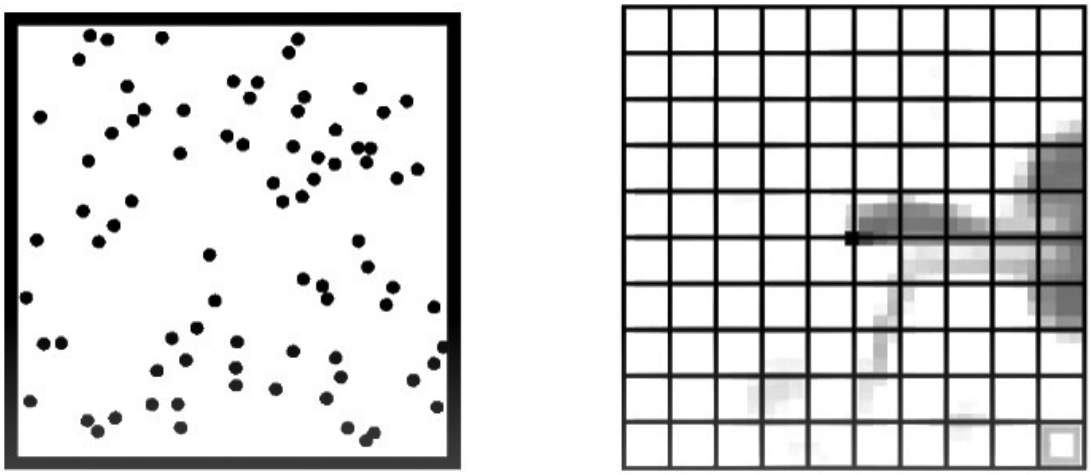

Рисунок 2.1 Моделювання частинками (зліва), моделювання сіткою областей (справа)

Використовуючи метод моделювання сіткою областей, рівні деталізації, які використовуються для представлення кожної області, також можуть змінюватися залежно від щільності кожної області, щоб обчислювальна потужність використовувалася найбільш ефективно там, де вона найбільш потрібна (рисунок 2.2).

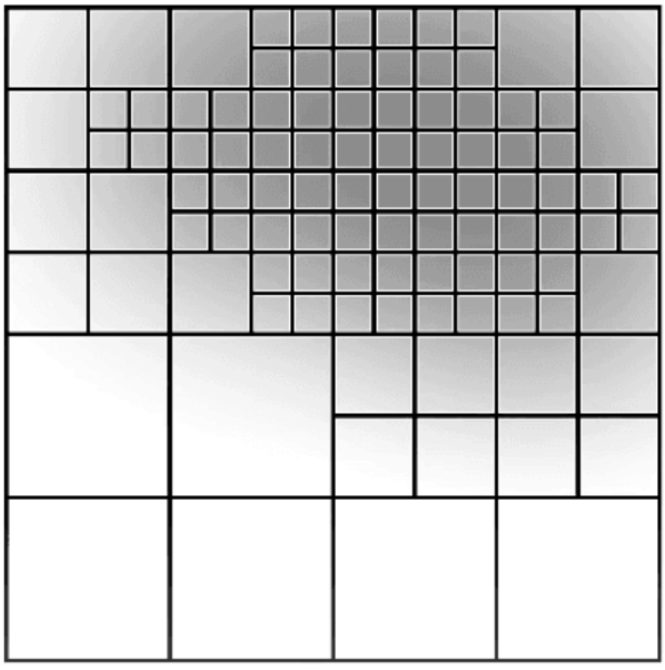

Рисунок 2.2 Рівні деталізації

Задля оптимізації та спрощення проектування симуляції потоків повітря у системі, виберемо варіант реалізації сітки з квадратними областями без динамічного розподілу деталей. Усі квадрати сітки будуть однакового розміру та залишатимуться статичними.

Повернемося до сітки. До кожної клітинки в сітці буде прив'язана інформація про рідину, що знаходиться в цій клітинці. Звичайно, найважливішою характеристикою рідини є те, що вона рухається. Найважливішими атрибутами руху є швидкість, та напрямок, в якому цей рух відбувається. Також кожна клітка сітки може мати інші атрибути, такі як: щільність, температура. Проте те, як усі ці атрибути розподіляються всередині симуляції визначається саме швидкістю.

Мета моделювання полягає в тому, щоб зафіксувати всі атрибути симуляції у певний момент часу, та розрахувати як ці атрибути зміняться протягом наступних ітерацій (рисунок 2.3). Звичайно, ці атрибути також можна буде безперервно змінювати протягом симуляції у вигляді інтерактивної взаємодії.

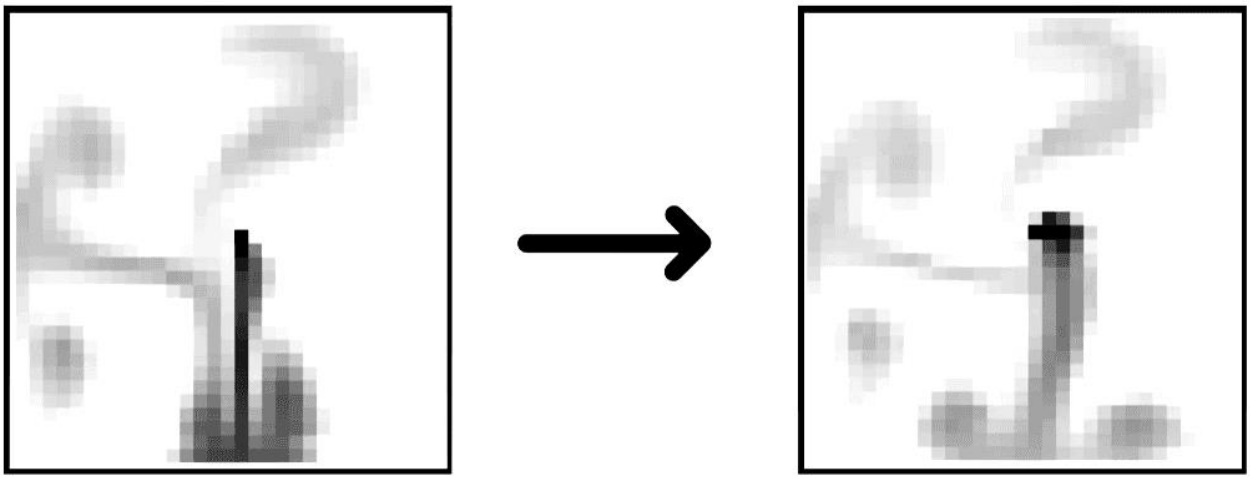

Рисунок 2.3 Зміна атрибутів клітинок протягом декількох ітерацій

2.2.3 Дифузія

Дифузія не залежить від швидкостей речовини. Ця характеристика відповідає за те, що атрибути кожної частини рідини будуть поширюватись на сусідні області. Фактичний розрахунок включає в себе поступове перетворення значення кожної клітки на середнє значення квадратів, що його оточують (рисунок 2.4).

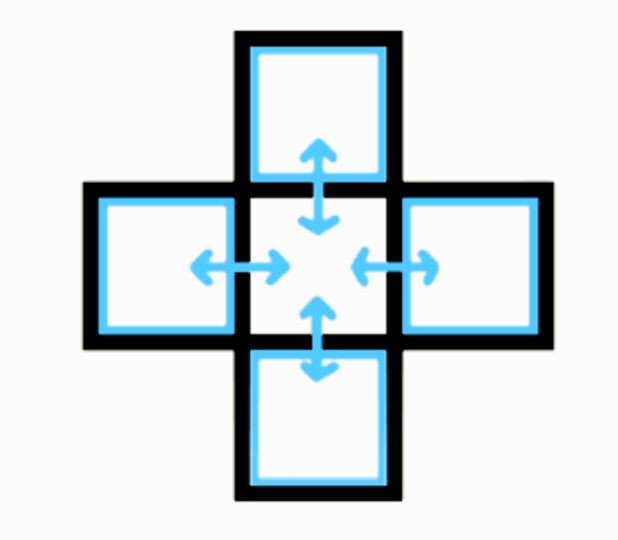

Рисунок 2.4 Приклад дифузії

Розглянемо дифузію щільності. Нехай щільність клітки на позиції (x, y) розраховують за формулою:

$$d(x, y) \qquad (2.2.3.1)$$

Тоді середню щільність сусідніх кліток розраховують за формулою:

$$s(x, y) = \frac{d(x+1,\ y) + d(x-1,\ y) + d(x,\ y+1) + d(x, y-1)}{4} \qquad (2.2.3.2)$$

k – коефіцієнт зміни щільності (буде змінюватись за рахунок часу одного кроку симуляції, та в'язкості);

■$_c$ – індекс для поточної ітерації (відомої);

■$_n$ – індекс для наступної ітерації (невідомої);

Необхідно поступово перетворити щільність клітки на позиції (2.2.3.1) на середню щільність сусідніх кліток (2.2.3.2). Перший спосіб реалізації цього переходу – це використання лінійної інтерполяції. Для цього візьмемо поточну щільність та добавимо до неї різницю між щільністю клітки та середньою щільністю її сусідніх областей, помножену на коефіцієнт.

Отримаємо рівняння:

$$d_n = d_c + k\,(s_c - d_c), \qquad (2.2.3.3)$$

де d_n – нова щільність;

d_c – поточна щільність;

Проте така інтерпретація представляє проблему: при $k > 1$, відбудеться перевищення цільового значення. Це призведе до нестабільних змін: наприклад, значення щільності може стати від'ємним та радикально змінювати своє значення на протязі кількох ітерацій. Неможна просто обмежити коефіцієнт k одиницею, оскільки це обмежить швидкість моделювання.

Вирішенням цієї проблеми є наступний метод: замість використання поточних значень для обрахунку наступних, знайдемо наступні значення, які, в разі переходу на попередню ітерацію, будуть представляти з себе вже відомі поточні значення. Запишемо це у вигляді рівняння (2.2.3.3) з заміною d_n на d_c, та s_c на s_n :

$$d_c = d_n - k\,(s_n - d_n) \qquad (2.2.3.4)$$

Винесемо d_n у ліву частину рівняння, тоді отримаємо:

$$d_n = \frac{d_c + k\, s_n}{1 + k} \qquad (2.2.3.5)$$

Отримуємо гіберболічне відношення замість лінійного. Таким чином, наскільки б великим не був коефіцієнт k – цільове значення ніколи не переповниться.

Це стабільний метод інтерполяції (рівняння 2.2.3.5), але він представляє іншу проблему: потрібно знати значення s_n (середню щільність сусідніх кліток на наступній ітерації) для обчислення, яке, очевидно, ще не є відомим.

Розгорнемо значення s_n в рівнянні (2.2.3.5). Тоді отримаємо рівняння:

$$d_n = \frac{d_c + k\, \frac{d(x+1,\ y) + d(x-1,\ y) + d(x,\ y+1) + d(x,y-1)}{4}}{1 + k} \qquad (2.2.3.6)$$

Оскільки у формулі (2.2.3.6) існує багато значень d, і кожен d - вираз містить в собі інші значення d, прийдемо до висновку, що це по суті система рівнянь.

2.2.4 Метод Гауса – Зайделя

Для створення системи моделювання повітряного потоку потрібно, щоб комп'ютер міг обробляти будь-яку систему рівнянь незалежно від значень, використовуючи однаковий алгоритм на кожній ітерації. Отже, потрібно використати ітераційний метод, щоб розрахувати приблизні розв'язки системи рівнянь.

Ітераційний метод, який буде використано для системи називається “метод Гауса – Зайделя”. Його суть полягає в тому, що для всіх значень, які є невідомими можна призначити щось випадкове (наприклад нуль). Далі відбувається розв'язок рівняння з використанням цих випадкових значень, та оновлюванням значень новими результатами, отриманими з рівнянь. Цей процес повторюється багато разів, кожен раз використовуючи нові значення. Повторюючи цей процес велику кількість ітерацій, отримані значення починають наближатись до правильних рішень для цієї системи рівнянь.

Однак метод Гауса-Зайделя потребує декілька умов для успішної роботи. Зазвичай система рівнянь повинна мати діагонально панівну матрицю. Це означає, що коефіцієнти, які лежать на діагоналі, повинні мати величину, більшу за суму величин усіх інших коефіцієнтів у цьому рядку. Це гарантує, що у вирішеному рівнянні для кожної змінної, коефіцієнти не будуть більшими за знаменник, тому отримані значення є контрольованими і не починають раптово збільшуватися. Однак повертаючись до рівняння щільності (2.2.3.6) можна замітити, що воно вже задовольняє цій умові, оскільки кожен з коефіцієнтів $((x + 1,\ y),\ d(x - 1,\ y),\ d(x,\ y + 1),\ d(x, y - 1))$ ділиться на 4, та сума цих чотирьох коефіцієнтів помножена на k ніколи не буде більше за знаменник $(1 + k)$.

Отже, таким чином розповсюджуються атрибути рідини в сітці (рисунок 2.4). Внаслідок багаторазового розв'язку системи рівнянь для кожної клітки, їхня густина зближається до дифузних густин, які вони повинні отримати.

2.2.5 Адвекція

Адвекція – це переміщення атрибутів через рідину відносно її швидкості (рисунок 2.5). Для узгодженості, продовжимо розглядати цей процес на прикладі атрибуту щільності.

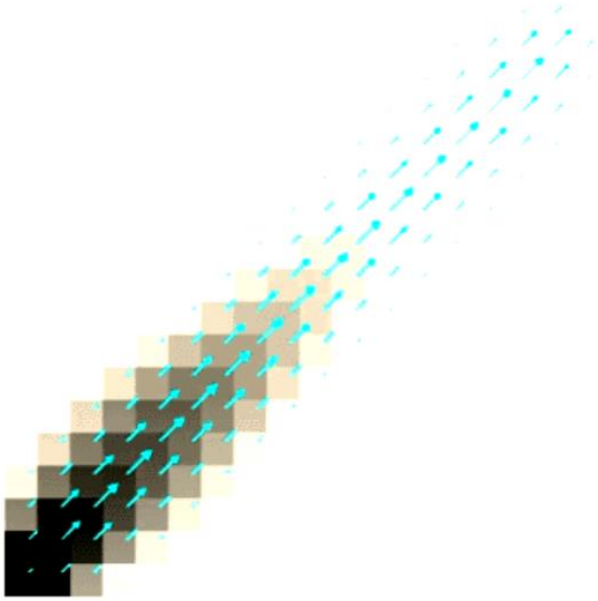

Рисунок 2.5 Приклад адвекції

Це відносно простий крок. Він був би ще простішим, якби вектор швидкості кожної клітки вказував точно на центр іншої клітки (рисунок 2.6). Тому, що тоді можна було б просто перемістити його щільність у новий квадрат. Однак вектори швидкості майже ніколи не вказують точно на центр іншого квадрату (рисунок 2.7). Це означає, що щільність, яка рухається слідом за цим вектором, насправді вплине на чотири квадрати, що оточують точку, куди вказує вектор швидкості (рисунок 2.8).

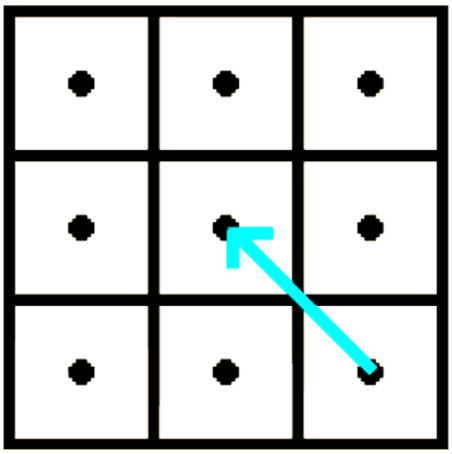

Рисунок 2.6 Вектор швидкості вказує точно на центр сусідньої клітки

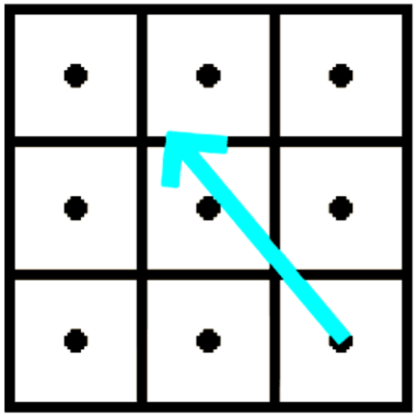

Рисунок 2.7 Вектор швидкості вказує на точку між сусідніми клітками

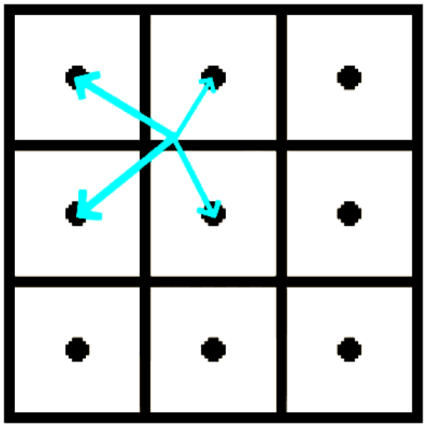

Рисунок 2.8 Чотири квадрати, на яких вплине переміщення щільності

Досить незручно переміщати щільності з їх поточних позицій на нові позиції, та точно розподіляти їх на навколишні квадрати, оскільки може існувати багато векторів, які вказують на те ж саме місце (рисунок 2.9).

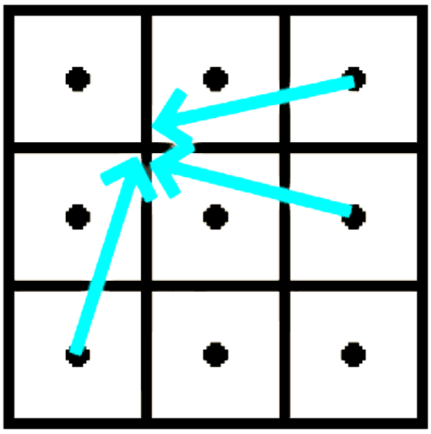

Рисунок 2.9

Отже, зробимо крок назад, щоб з'ясувати звідки саме буде братися наступна щільність клітки. Щоб знайти цю позицію проведемо лінійну інтерполяцію між густинами навколишніх чотирьох квадратів, та знайдемо цільову щільність. Таким чином кожна клітка вимагає лише одного обчислення.

Щоб зайти позицію, звідки буде поступати наступне значення щільності (рисунок 2.10), візьмемо положення квадрату, та віднімемо його швидкість, помножену на зміну часу.

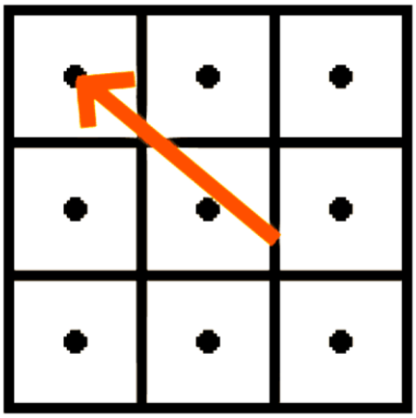

Рисунок 2.10

Нехай f – позиція, звідки буде поступати наступне значення щільності:

$$f = (x, y) - v(x, y)\Delta t, \quad (2.2.5.1)$$

де (x, y) – координати позиції клітки;

$v(x, y)$ – швидкість на позиції (x, y);

Δt - зміна часу;

Далі, щоб дізнатися які центри квадратів оточують це положення, знайдемо цілу частину координат.

Нехай i – ціла частина координат, тоді:

$$i = floor(f), \quad (2.2.5.2)$$

де floor() – функція, що знаходить цілу частину числа;

Щоб знайти точне положення між центрами квадратів (рисунок 2.11), знайдемо дробову частину координат.

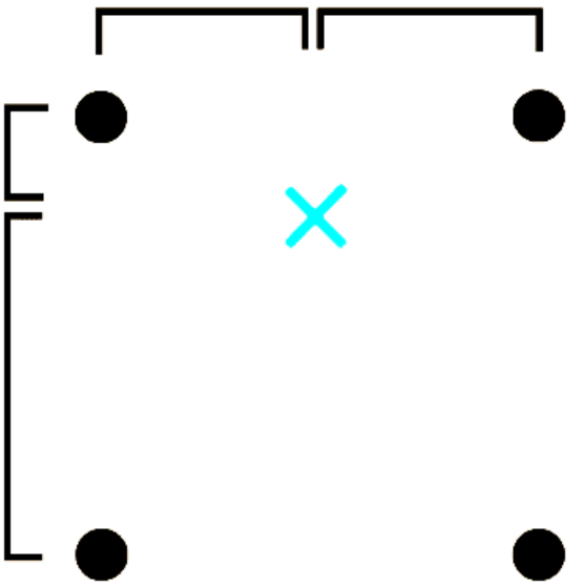

Рисунок 2.11

Нехай j – дробова частина координат, тоді:

$$j = fract(f), \tag{2.2.5.3}$$

де fract() – функція, що знаходить дробову частину числа;

Далі, використовуючи лінійну інтерполяцію, інтерполюємо між чотирма навколишніми значеннями щільності в центрах кліток:

$$lerp(a, b, k) = a + k(b - a), \tag{2.2.5.4}$$

де lerp(a,b,k) – функція, що виконує лінійну інтерполяцію між a та b, відносно k;

На основі функції (2.2.5.4) інтерполюємо значення двох верхніх кліток, та запишемо значення в z_1:

$$z_1 = lerp\,(\,d(i_x,\ \ i_y),\ d(i_x\ +\ 1,\ \ i_y), j_x\,), \qquad (2.2.5.5)$$

Далі інтерполюємо значення двох нижніх кліток, та запишемо значення в z_2:

$$z_2 = lerp\,(\,d(i_x,\ \ i_y\ +\ 1),\ d(i_x\ +\ 1,\ \ i_y\ +\ 1), j_x\,), \qquad (2.2.5.6)$$

Нарешті обчислимо шукане значення щільності для клітки. Для цього проведемо інтерполяцію між z_1 та z_2 за допомогою дробової координати у (j_y):

$$z_2 = lerp\,(z_1, z_2, j_y\,), \qquad (2.2.5.7)$$

2.2.6 Проблема дивергенції

Для початку необхідно з'ясувати поняття скручування та дивергенції. По суті, у векторних полях скручування – це коли вектори обертаються навколо один одного (рисунок 2.12), а дивергенція (розбіжність) – це коли вектори або вказують один на одного, або вказують один від одного (рисунок 2.13).

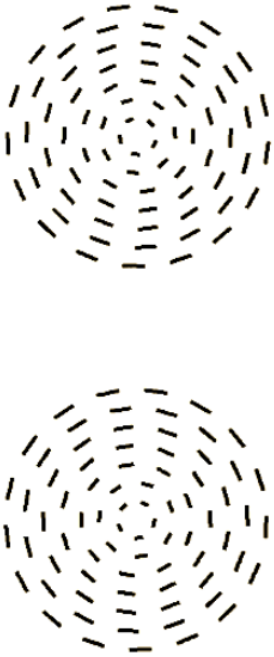

Рисунок 2.12 Демонстрація скручування

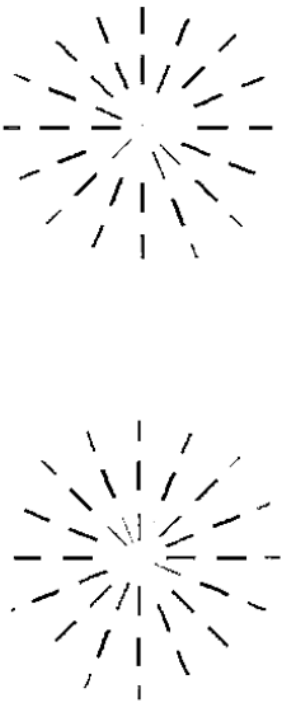

Рисунок 2.13 Демонстрація дивергенції (розбіжностей)

Ці дві властивості є універсальними для кожного векторного поля, з урахуванням будь-якого векторного поля, яке можна виміряти, та з'ясувати, як багато скручувань та розбіжностей воно має.

Рідина часто обертається, отже містить в собі велику кількість скручувань, однак якщо рідина має дивергенцію – це буде означати, що матерія буде зникати в пустоту, або виникати з пустоти. Тому необхідно, щоб поле швидкостей речовини мало скручування і не мало дивергенції (розбіжностей). Однак після того, як відбудеться виконання дифузії та адвекції для імітування зміни поля швидкостей рідини, фактично вийде поле, що буде містити в собі як дивергенцію, так і скручування. Тому необхідно виділити лише частину, вільну від дивергенції.

2.2.7 Теорема декомпозиції Гельмгольца

Відповідно до теореми Гельмгольца, також відомої як “фундаментальна теорема векторного числення”: будь-які векторні поля можуть бути виражені як сума двох векторних полів, одне з яких немає скручування, а інше - не має дивергенції (рисунок 2.14). Немає прямого способу обчислити векторне поле без дивергенції, тому метою буде обчислення частини векторного поля без скручування, і в подальшому знаходження різниці з початковим векторним полем швидкості, що в результаті дасть поле без дивергенції (рисунок 2.15).

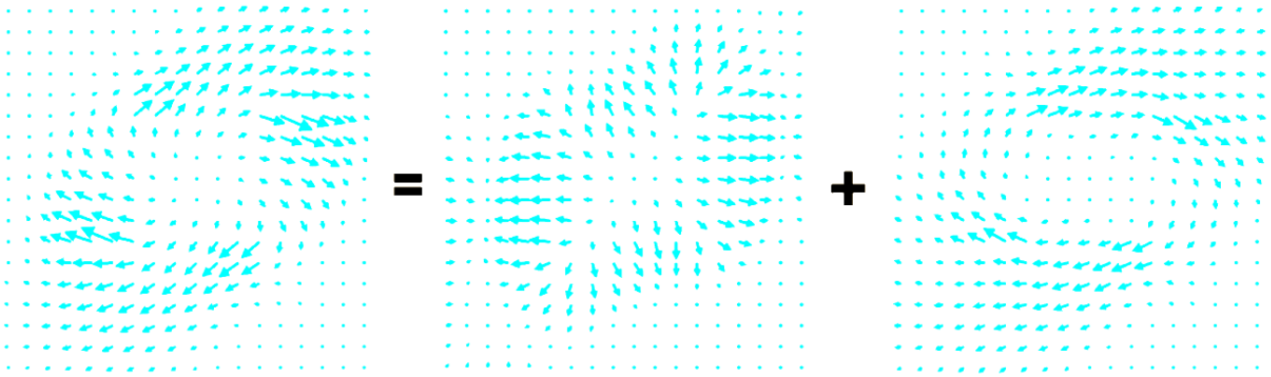

Рисунок 2.14 Демонстрація теореми Гельмгольца (зліва – векторне поле швидкості, посередині – векторне поле без скручування, справа – векторне поле без дивергенції)

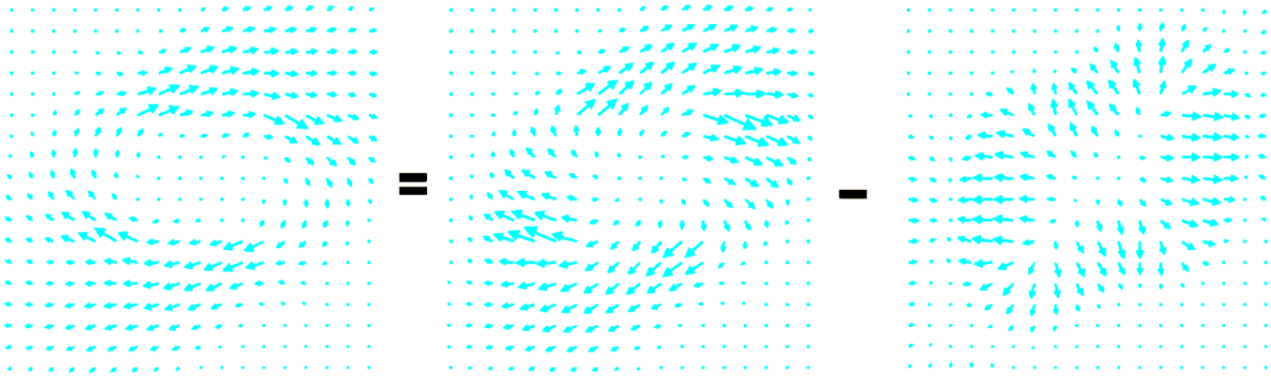

Рисунок 2.15 Демонстрація теореми Гельмгольца (зліва – векторне поле без дивергенції, посередині – векторне поле швидкості, справа – векторне поле без скручування)

Спочатку, розрахуємо дивергенцію у кожній позиції поля за допомогою наступного рівняння:

$$\nabla \cdot v(x,y) = \frac{v_x(x+1,y) - v_x(x-1,y) + v_y(x,y+1) - v_y(x,\ y-1)}{2}, \qquad (2.2.7.1)$$

де $v_x(x\ +\ 1, y)\ -\ v_x(x\ -\ 1, y)$ – різниця в X - швидкостях між двома горизонтальними клітками;

$v_y(x, y\ +\ 1)\ -\ v_y(x,\ \ y-1)$ – різниця в Y - швидкостях між двома вертикальними клітками;

2 – відстань між цими клітками;

Після знаходження дивергенції на всіх клітках поля, необхідно обрахувати наступну систему рівнянь за допомогою методу Гауса – Зайделя (пункт 2.2.4):

$$p(x,y) = \frac{[\,p(x-1,\ \ y) + p(x+1,\ \ y) + p(x,\ \ y-1) - p(x,\ \ y+1)] - \nabla\cdot v(x,\ \ y)}{4}, \quad (2.2.7.2)$$

В результаті вийшло поле зі скалярними значеннями p для кожної клітки. Необхідно знайти векторне поле градієнту цього скалярного поля:

$$\nabla p(x,y) = \left(\frac{p(x+1,\ \ y) - p(x-1,\ \ y)}{2}, \frac{p(x,\ \ +1y) - p(x,\ \ y-1)}{2}\right), \quad (2.2.7.3)$$

Одна з тотожностей векторного числення стверджує, що скручування такого градієнта векторного поля завжди дорівнює нулю. Тому в результаті ми отримали векторне поле без скручування. Тепер ми можемо відняти його від початкового векторного поля швидкостей, щоб очистити рідину від дивергенції (рисунок 2.15).

2.2 Висновок щодо алгоритму

Описані в даному розділі кроки (дифузія, адвекція, усунення дивергенції) є основними процедурами, які може мати симуляція рідини. Також вони є досить геніальними та елегантними методами представлення фізики з матеріального світу в алгоритмічно зручний спосіб.

Даний алгоритм є досить спрощеним варіантом симуляції рідини, оскільки в ньому не враховуються тиск та температура в якості атрибутів. Проте головною його перевагою є те, що з його допомогою сучасні персональні комп'ютери з легкістю можуть симулювати рідину у режимі реального часу.

РОЗДІЛ 3 ОПИС СИСТЕМИ

3.1 Засоби розробки

3.1.1 Вибір мови програмування

Під час розробки проекту було проаналізовано досить багато мов програмування та технологій. Виходячи з власного досвіду, та порад від компетентних розробників було виявлено низку потрібних технологій для вирішення поставленої задачі. Ці технології є простими у використанні, досить гнучкими, та швидкими. Також вони мають хороші перспективи в майбутньому, оскільки мають високі рейтинги симпатії та довіри у користувачів.

В якості основної мови програмування було обрано JavaScript [25]. Це мова програмування високого рівня, яка лежить в основі технологій всесвітньої мережі. Сьогодні майже всі веб-сайти використовують JavaScript для написання потрібних скриптів, проте вона використовується також в багатьох середовищах, окрім браузерів, наприклад, для написання серверів.

Клієнтський JavaScript працює на front-end частині застосунку, та може бути використаний для розробки поведінки веб-сторінок. JavaScript підтримує об'єктно орієнтовний, імперативний та функціональний стилі програмування. Динамічні можливості JavaScript включають побудову об'єктів під час виконання, змінні функцій, інспекцію об'єктів (через for … in), декомпіляцію тіл функцій у вихідний код, і так далі. Найвідомішим рушієм JavaScript – є рушій V8. Він був розроблений Данським відділенням компанії Google, та є проектом з відкритим кодом. V8 використовується для виконання JavaScript та Node.js.

Серед переваг JavaScript можна виділити наступне: в наслідок того, що це інтерпретована мова програмування, вона зменшує кількість часу, необхідного для компіляції. Також він може виконуватись на front-end (клієнтській) частині веб-застосунку, що значно прискорює виконання програми, оскільки його

виконання не потребує підключення до сервера. JavaScript легко зрозуміти та вивчити, за рахунок С-подібного синтаксису.

Також в JavaScript реалізований механізм, що називається Event Loop. Він дозволяє мові легко виконувати асинхронні операції (рисунок 3.1).

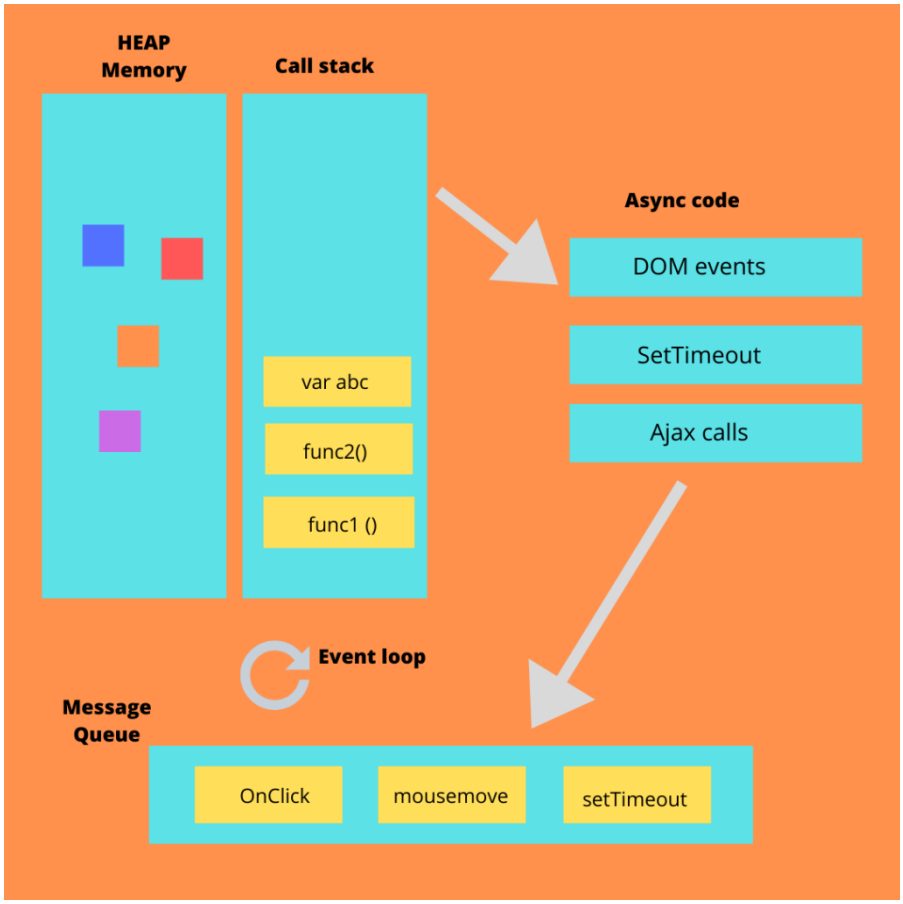

Рисунок 3.1 Схема роботи Event Loop в рушії V8 для JavaScript [26]

3.1.2 Клієнтська частина

3.1.2.1 Користувацький інтерфейс

Для створення користувацького інтерфейсу було обрано JavaScript-бібліотеку ReactJs [18]. Вона була створена так, щоб її впровадження у проект могло відбуватись поступово. Тобто компоненти ReactJs на front-end частині застосунку можна суміщати з привичними компонентами. Великою перевагою

ReactJs є те, що розробники створили чудову документацію, яка підійде для людей із різним досвідом та стилем вивчання. Будь-який розробник, який має досвід з JavaScript, за кілька днів, зможе усвідомити філософію ReactJs, та почати створювати веб-додатки за допомогою цієї бібліотеки.

Створення динамічних веб-додатків за допомогою простого HTML - це складна задача, проте ReactJs вирішив цю проблему, та полегшив процес розробки веб-додатків для розробників. В порівнянні зі створенням додатку без допоміжних бібліотек, ReactJs зменшує кількість необхідного коду, та надає більше функціональних можливостей. Він використовує JSX (JavaScript Extension), який є вдалим синтаксисом для подання дочірніх компонентів HTML тегів.

Веб-додаток ReactJs може складатись з кількох компонентів, і кожен компонент має власні елементи керування та логіку. Ці компоненти відповідають за подання, зазвичай, невеликого за розміром фрагменту HTML-коду, який можна повторно використовувати там, де він буде потрібен у вашому інтерфейсі. Компоненти багаторазового використання полегшують розробку та підтримку програм. Ці компоненти можуть бути вкладеними в інші компоненти, що дозволяє будувати складні користувацькі інтерфейси з простих будівельних блоків.

ReactJs використовує віртуальний механізм на основі DOM для заповнення даних в HTML DOM (рисунок 3.2). Віртуальний DOM працює швидше, ніж звичайний, оскільки змінює лише окремі елементи DOM замість того, щоб кожен раз оновлювати повний DOM. ReactJS покращує продуктивність завдяки віртуальному DOM. Більшість розробників стикалися зі сповільненням роботи програми під час оновлення DOM. ReactJS вирішив цю проблему, представивши віртуальний DOM. Віртуальний DOM React міститься в оперативній пам'яті програми і є репрезентацією DOM-у веб-браузера. Коли front-end програміст пише компонент React, він не влаштовується безпосередньо в DOM. Замість цього, описані розробником віртуальні компоненти ReactJs, взаємодіють з HTML

DOM тільки тоді, коли це потрібно, що призводить до більш гладкої та швидкої роботи.

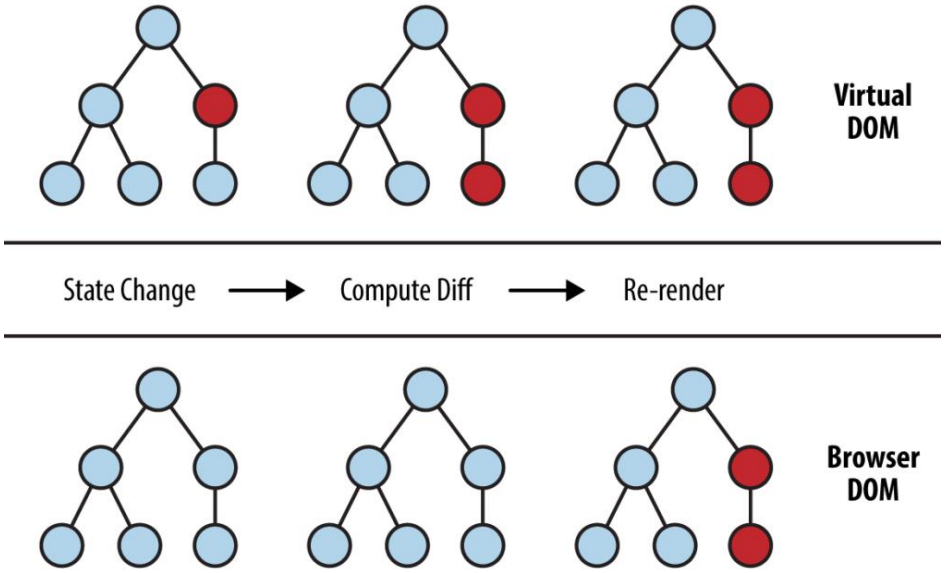

Рисунок 3.2 Віртуальний DOM бібліотеки ReactJS [27]

Більшість фреймворків JavaScript мають проблеми з SEO. Пошукові системи зазвичай мають проблеми з читанням важких JavaScript додатків. Багато веб-розробників часто скаржилися на цю проблему. ReactJS долає цю проблему: застосунки, написані за допомогою ReactJS можуть працювати на сервері, а віртуальна DOM буде перетворена на звичайну DOM, та повернеться до браузера як звичайна веб-сторінка.

Серед недоліків ReactJS можна відмітити високі темпи розвитку бібліотеки, та постійні радикальні зміни в документації. Середовище змінюється так швидко, що деякі розробники не виявляють бажання адаптуватись, та регулярно вивчати нові способи роботи.

3.1.2.2 Бібліотека компонентів

Оскільки унікальність стилізації компонентів для веб-сторінок не є першочерговою задачею, було використано бібліотеку компонентів, яка допоможе нам створити сучасний користувацький інтерфейс без потреби прописування більшості стилів вручну. AntDesign є другою у світі за популярністю бібліотекою ReactJS компонентів [16].

AntD має широкий вибір готових та візуально красивих компонентів, необхідних для створення веб-застосунку. Це включає в себе основні компоненти, такі як таблиці та форми, та спеціалізовані: сповіщення, календарі та складні меню (рисунок 3.3). Також є можливість придбати професійну ліцензію та отримати доступ до графіків, діаграм, інформаційних панелей та багато іншого. AntD також надає кілька шаблонів разом із онлайн-редактором для швидкого створення прототипу дизайну вашого сайту.

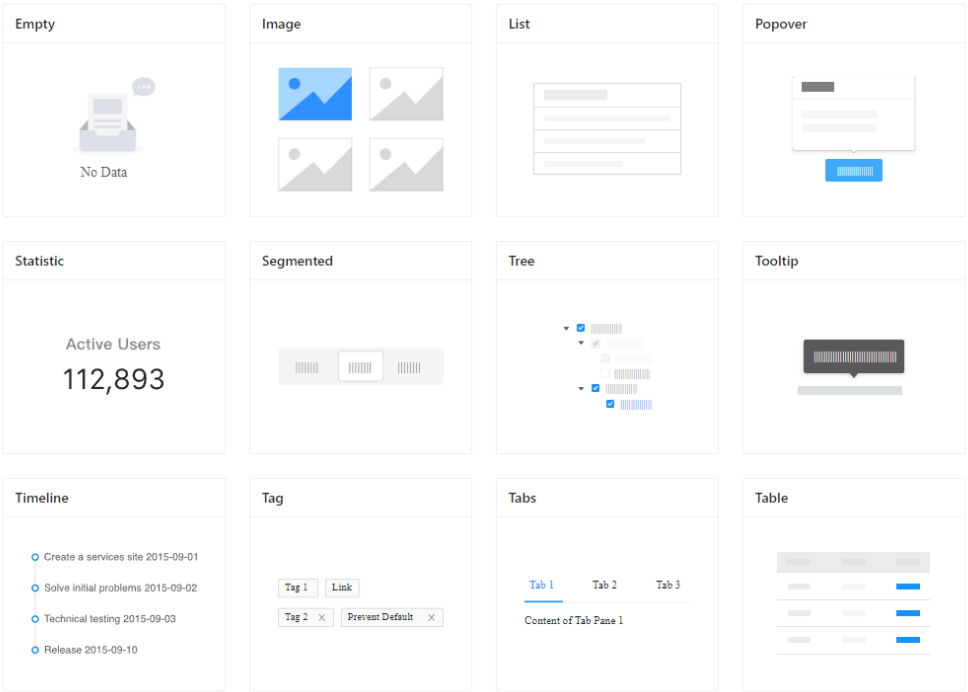

Рисунок 3.3 Приклад стилізованих компонентів AntDesign

3.1.2.3 Стилізація компонентів

Під час розробки великих веб-застосунків за допомогою таких бібліотек, як React, виникає певний дискомфорт при стилізації великої кількості компонентів на чистому CSS. Тому для спрощення реалізації цієї частини проекту на React – хорошим рішенням буде використання бібліотеки styled-components [11]. Вона покриває весь синтаксис CSS, та добавляє додаткові оператори, які спрощують динамічну передачу змінних для зміни стилів (рисунок 3.4).

```
// import styled from 'styled-components'

const padding = '3em'

const Section = styled.section`
  color: white;

  /* Pass variables as inputs */
  padding: ${padding};

  /* Adjust the background from the properties */
  background: ${props => props.background};
`

render(
  <Section background="cornflowerblue">
    ✨ Magic
  </Section>
)
```

Рисунок 3.4 Приклад передачі змінних для динамічної зміни стилів з сайту документації styled-components [11]

3.1.2.4 Інтерактивна графіка

Для створення візуалізації повітряних потоків у проекті необхідно обрати бібліотеку, яка дозволила б відобразити обрахунки та числа з алгоритму обчислювальної гідродинаміки на щось візуально зрозуміле. Також необхідно забезпечити інтерактивність користувача з графічним інтерфейсом для подальшого створення моделей приміщень та вентиляційних систем. Для вирішення цього завдання було обрано один з найпопулярніших 2D-рушіїв для веб-додатків – Pixi.js

Pixi.js - це безкоштовний 2D-рушій з відкритим вихідним кодом, який використовується розробниками для створення анімованих веб-застосунків та ігор [10]. Він сумісний з усіма сучасними браузерами - як на настільних комп’ютерах, так і на мобільних пристроях. Він був запущений у лютому 2013 року Меттом Гроувсом і є безкоштовним за ліцензією MIT.

Найбільшою перевагою Pixi.js є його швидкість. Pixi.js - це спеціалізований механізм візуалізації (рисунок 3.5). Він використовує WebGL для більш швидкої роботи, що робить 2D-рендерінг дуже швидким. Проте якщо WebGL на пристрої не підтримується, механізм повертається до стандартного HTML Canvas.

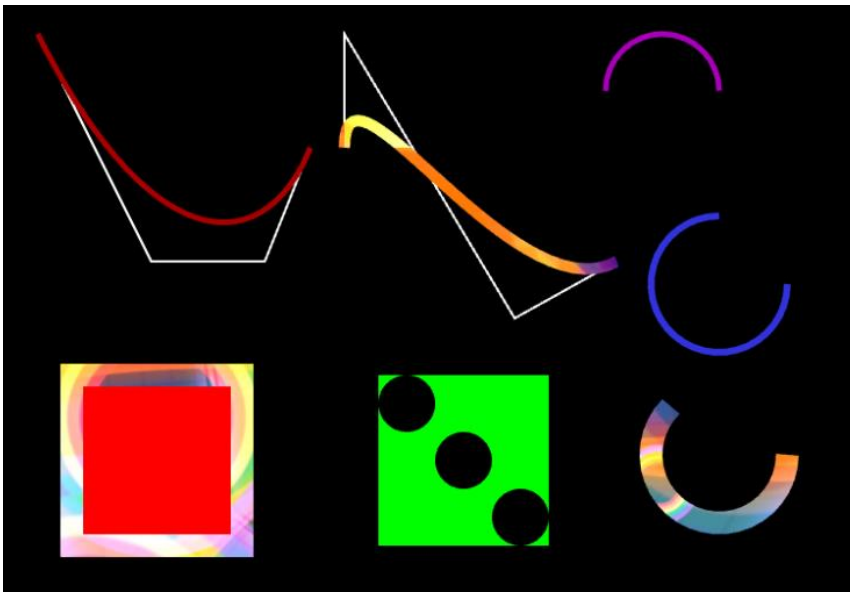

Рисунок 3.5 Приклад 2D графіки, виконаної за допомогою PixiJS

3.1.2.5 HTTP клієнт

Оскільки архітектура веб-застосунку буде передбачувати серверну частину для збереження, завантаження та оновлення користувацьких проектів, необхідно реалізувати взаємодію front-end частини застосунку з сервером. Для запитів з боку клієнтської частини необхідно використати HTTP клієнт.

Axios – популярна бібліотека, яка спеціалізується на створенні асинхронних запитів HTTP на кінцеві точки REST [15]. Ця бібліотека використовується для зв'язку з back-end (рисунок 3.6). Використовуючи Axios, ми робимо запити до API нашої серверної частини, та, після того, як запит оброблено і надіслано відповідь, ми отримуємо дані, а потім використовуємо їх в нашому проекті. Ця бібліотека дуже популярна серед розробників, оскільки навіть на GitHub в неї близько 94 тисяч зірок.

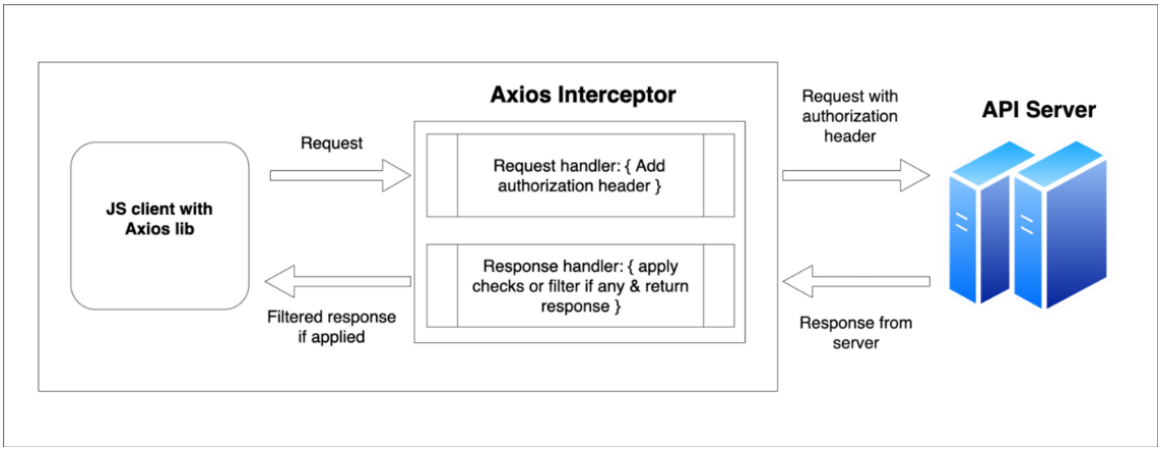

Рисунок 3.6 Приклад роботи бібліотеки Axios [28]

3.1.2.6 Комплектування програми

Для розробки тяжких веб-додатків існує багато різних комплектувальників, наприклад: Vite, Parcel, Webpack тощо. Колись для проектів на React використовували виключно Webpack, проте сьогодні світ веб-інструментів поповнюється новими альтернативами, що стрімко піднімаються в чартах швидкості комплектування проекту (рисунок 3.7).

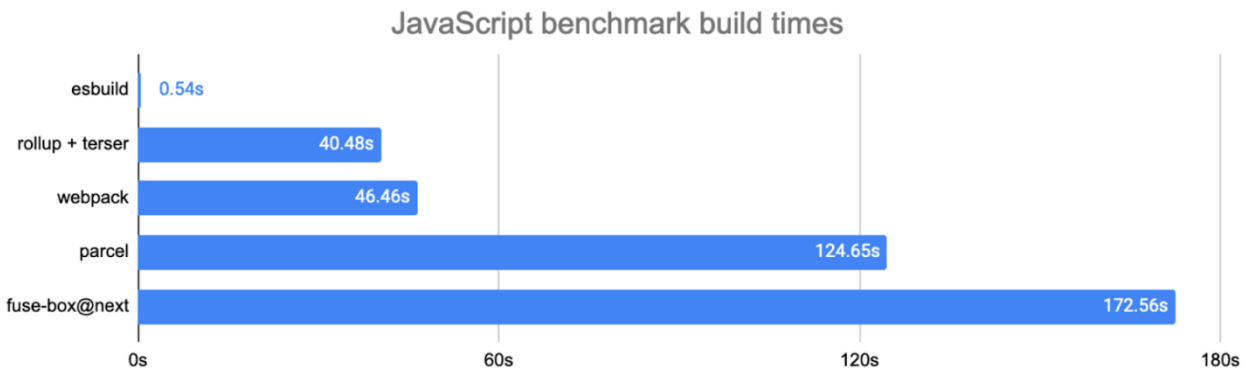

Рисунок 3.7 Порівняння швидкостей різних комплектувальників [29]

У проекті буде використано комплектувальник Vite. Причиною цього є швидкість. Він використовує сучасні ESM та EsBuild та досягає від 10 до 100 кратного підвищення швидкості зборки. Також Vite вирішує проблему, що має назву “Hot Module Replacement” (швидка заміна модулів), та без повного перезавантаження замінює, додає або видаляє модулі під час роботи програми без повного перезавантаження.

3.1.3 Серверна частина

3.1.3.1 Мова програмування

Під час вибору мови програмування для написання серверної частини застосунку було обрано Node.js.

Node.js дозволяє використовувати JavaScript для розробки, і, в наслідок цього, стає легко зіставляти внутрішні функції серверу із зовнішніми функціями клієнтської частини додатку. Однією з переваг використання Node.js є вбудована підтримка JSON. JSON є основою більшості HTTP-запитів. Тоді як для більшості інших серверних мов потрібні спеціальні бібліотеки для аналізу JSON, Node.js робить це з коробки. [14]

Node.js робить процес розробки швидким. Сервери Node.js можна запустити з невеликими зусиллями, а базовий API «Hello World» можна запустити менше, ніж за хвилину. Завдяки тому, що в Node легко створити API, Node.js також є чудовим вибором для розробників, які створюють середовища мікросервісів, оскільки вони пов’язують багато API разом. Node.js є відмінним вибором для прототипів рішень та архітектур, оскільки він робить експерименти швидкими та простими.

Однією з головних причин вибору Node.js є NPM, менеджер пакетів для Node.js. NPM дозволяє завантажувати пакети коду, створені іншими

розробниками, і використовувати їх у своїх власних проектах. Для розробника це означає, що йому не прийдеться писати код, який вже був написаний до нього. NPM полегшує керування залежностями програми, встановлюючи не лише код бібліотеки, а й усі залежності для цієї бібліотеки.

На базі NPM було створено інший менеджер пакетів - yarn, який має деякі переваги над NPM, а саме: паралельне та мультипоточне завантаження пакетів яке є значно швидшим за звичайне однопоточне, та спрощений синтаксис який дозволяє легше використовувати yarn у командному рядку.

3.1.3.2 Веб-фреймворк

Для створення серверу на Node.js необхідно використати веб-фреймворк. Сьогодні існує два основних конкуренти: Express та Fastify. В результаті аналізу відгуків розробників було обрано Fastify, оскільки він є швидшим, та легшим для використання.

Fastify — це веб-фреймворк, орієнтований на забезпечення найкращого досвіду розробника з найменшими витратами та потужною архітектурою плагінів. Fastify можна використовувати для швидкого налаштування сервера на Node.js. Цей фреймворк може обробляти великі запити, використовуючи при цьому дуже малі ресурси тому, що запускає мінімальну кількість функцій для обробки запиту. За замовчуванням Fastify має безпечний аналізатор формату JSON для читання інформації з запиту.

3.1.3.3 База даних

Для зберігання користувацьких проектів необхідно було обрати базу даних. Взаємодія з базою даних повинна відбуватись через серверну частину застосунку, написану за допомогою Node.js. Оскільки схема об'єктів, що зберігаються в базі даних не буде складною, та включати в себе зв'язки між таблицями, було вирішено обрати MongoDB.

MongoDB була створена в 2009 році як надійна безкоштовна база даних NoSQL з відкритим вихідним кодом. Він також має комерційну версію. Вихідний код MongoDB можна знайти на GitHub. MongoDB здобула репутацію універсальної, гнучкої бази даних і сьогодні використовується як серверне сховище даних багатьох високопоставлених компаній і організацій, таких як Forbes, Facebook, Google, IBM, Twitter та багатьох інших.

MongoDB — це нереляційна система баз даних. Існує два основних типи баз даних: SQL (реляційна) і NoSQL (нереляційна). Реляційні бази даних зберігають дані в стовпцях і рядках. З іншого боку, бази даних NoSQL зберігають неструктуровані дані без схем у кількох колекціях і вузлах. Нереляційні бази даних не потребують фіксованих таблиць.

У MongoDB записи зберігаються як документи у стиснутих файлах BSON. Документи можна отримати безпосередньо у форматі JSON (рисунок 3.8), який має багато переваг. Більшості розробників легко працювати з JSON, оскільки це простий і потужний спосіб описувати та зберігати дані. MongoDB створив двійковий формат JSON (BSON) , щоб підтримувати більше типів даних, ніж JSON. Цей новий формат дозволяє швидше аналізувати дані. Дані, що зберігаються в BSON, можна шукати та індексувати, що значно підвищує продуктивність. MongoDB підтримує широкий спектр методів індексування, включаючи текстові, десяткові та часткові. [30]

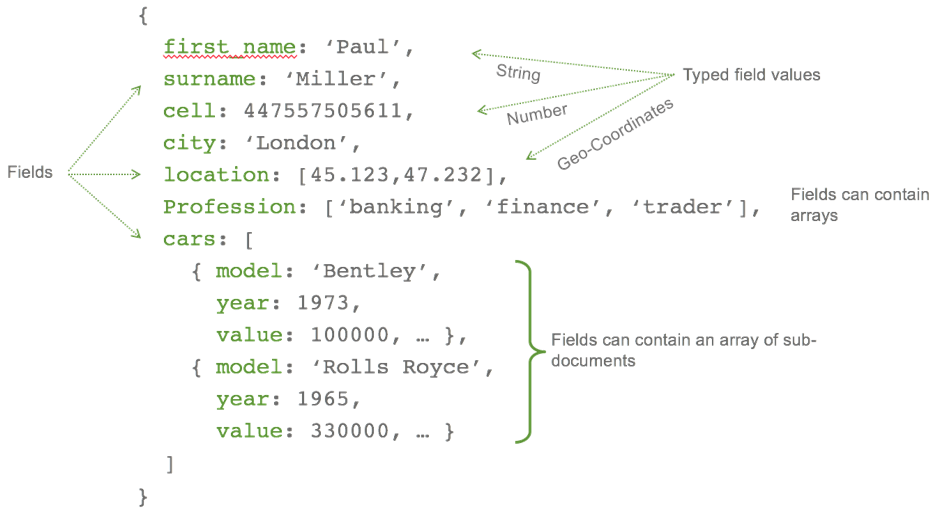

Рисунок 3.8 Приклад запису в базі даних MongoDB [30]

3.1.3.4 Взаємодія з базою даних

MongoDB - це сучасна база даних загального призначення, яка широко поєднується з Node.js у популярних технологічних стеках, таких як MEAN стек (MongoDB, Express.js, AngularJS і Node.js) і стек MERN (MongoDB, Express.js, React.js і Node.js).

Yarn, менеджер пакунків Node.js, - це інструмент, який дозволяє встановити драйвер MongoDB Node.js, що значно полегшує розробникам роботу з MongoDB з програми в Node.js.

3.1.4 Система контролю версій

Для оптимізації розробки проекту було вирішено застосувати систему контролю версій Git, а саме, онлайн-хостинг Git системи - GitHub.

Git є найбільш популярною системою контролю версій. Git відстежує зміни, внесені до файлів, і надає можливість повернутись до минулих версій файлів за потребності. Git також спрощує співпрацю, дозволяючи всі зміни, внесені кількома людьми, об’єднувати в одне джерело. Git - це програмне забезпечення, що працює локально. Файли та їхня історія зберігаються на комп’ютері. [31]

GitHub може бути використаний для збережень копій файлів, та їх історії редагування (рисунок 3.9).

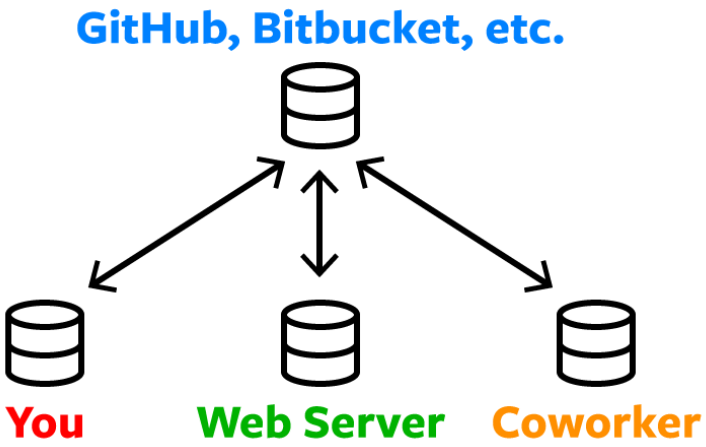

Рисунок 3.9 Архітектура роботи GitHub [31]

3.1.5 Docker-контейнери

Для ізоляції процесів, відкритих на одному хості, запуску додатків, призначених для різних платформ, можна використовувати віртуальні машини, що ділять між собою ресурси хоста. Недоліком такого підходу є те, що на обслуговування кожної з віртуальних машин йде значна частина ресурсів. Ці ресурси могли б використовувати самі додатки.

Альтернативним підходом до ізоляції додатків є контейнери. Він полягає у виділенні ізольованої області для запуску додатків в межах самої операційної системи.

Контейнери Docker використовують цю ідею і надають зручний інтерфейс для створення та управління контейнерами. Docker пропонує поняття образу (image) – шаблона для створення контейнера, що містить операційну систему, сам додаток і бібліотеки, потрібні для його запуску. Для створення образу зручно писати Dockerfile, який описує які додати бібліотеки, який обрати базовий образ та інше.

3.1.6 Розгортання програмного забезпечення та CI/CD

Після того, як додаток буде розроблений, необхідно зберегти його у веб-мережі, та зробити загальнодоступним незалежно від ПК розробника.

Безперервне розгортання (CI/CD) — це автоматизований випуск програмного забезпечення, що забезпечує розгортання коду в робочому стані (рисунок 3.10). Щоб мінімізувати ризик розгортання проблемного коду, потрібен надійний і повний набір автоматизованих тестів. Це гарантує, що новий код, який об’єднується в головну гілку (і згодом розгортається), буде готовий до розгортання та без помилок.

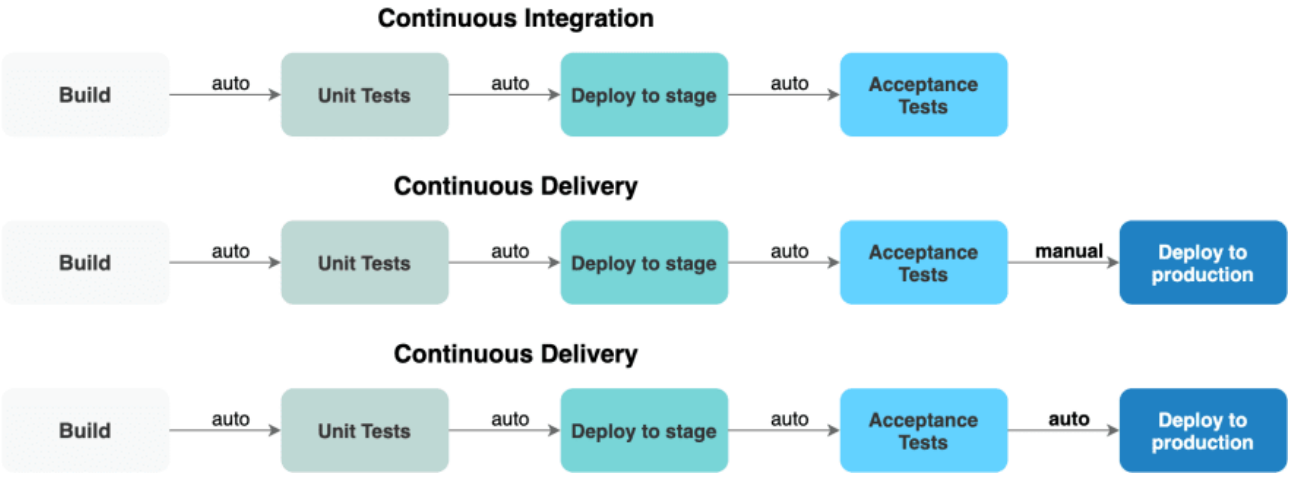

Рисунок 3.10 Схема CI/CD системи [32]

Vercel – це платформа для статичних сайтів та інтерфейсних фреймворків. Vercel реалізує безперервне розгортання застосунку (CI/CD) в довільному середовищі. Він також надає інтерфейс керування змінними середовища, який є досить стандартним. Також перевагою Vercel є те, що він легко інтегрується з GitHub, який вже використовується в проекті. Платформа дозволяє розробникам легко розробляти, переглядати та передавати найновіші оновлення клієнтам застосунку.

3.1.7 Статистичний аналіз коду

JavaScript, будучи динамічною мовою з нечіткими типами, особливо схильний до помилок розробника. ESLint дозволяє створити контроль над заданим стилем чи стандартом програмування і допоможе мінімізувати ці помилки. Основна причина введення цих стандартів полягає в тому, що кожен розробник має свій стиль письма (наприклад, умовні назви/табуляції/одинарні чи подвійні лапки для рядка). І з різними методами стилізації кодова база

програміста може бути більш вразливою і схильною до помилок. Особливо, під час роботи з Javascript, це може призвести до підводних каменів, з якими ніхто ніколи не захоче мати справу.

ESLint — це утиліта для Javascript з відкритим вихідним кодом. Вона часто використовується для пошуку проблемних шаблонів або коду, який не відповідає певним рекомендаціям щодо стилю. ESLint написаний за допомогою Node.js для забезпечення швидкого середовища виконання та легкої установки через NPM або Yarn. За допомогою ESLint ви можете встановити стандарт написання коду, використовуючи певний набір окремих правил. Eslint також надає можливість вмикати та вимикати ці правила. [13]

3.2 Реалізація веб-інструментарію

3.2.1 Архітектура системи

Веб-інструментарій для моделювання повітряних потоків у приміщенні має включати такі функції:

- Робота користувача з системою
- Моделювання повітряних потоків
- Взаємодія з базою даних

Робота користувача з системою включатиме такі функції:

- Відкриття проекту (створення нового або пошук існуючого)
- Редагування налаштувань проекту
- Побудова плану приміщення
- Побудова системи вентиляції
- Збереження проекту
- Запуск симуляції

- Перегляд візуалізації моделювання
- Зупинка симуляції

Моделювання повітряних потоків з системою включатиме наступні функції:

- Отримання даних про атрибути проекту та сітку об'єктів
- Перетворення сітки в одновимірний масив
- Ініціалізація алгоритму
- Створення життєвого циклу для алгоритму та візуалізації
- Запуск алгоритму
- Обробка атрибутів кліток сітки та об'єктів системою відображення
- Відображення елементів та атрибутів кліток сітки

Взаємодія з базою даних включатиме наступні функції:

- Підключення до бази даних
- Збереження нового проекту
- Оновлення існуючого проекту
- Видача інформації про всі проекти
- Видача інформації про конкретний проект за ключем

На основі описаного функціонального аналізу побудовано дерево функцій, що можна побачити на рисунку 3.11.

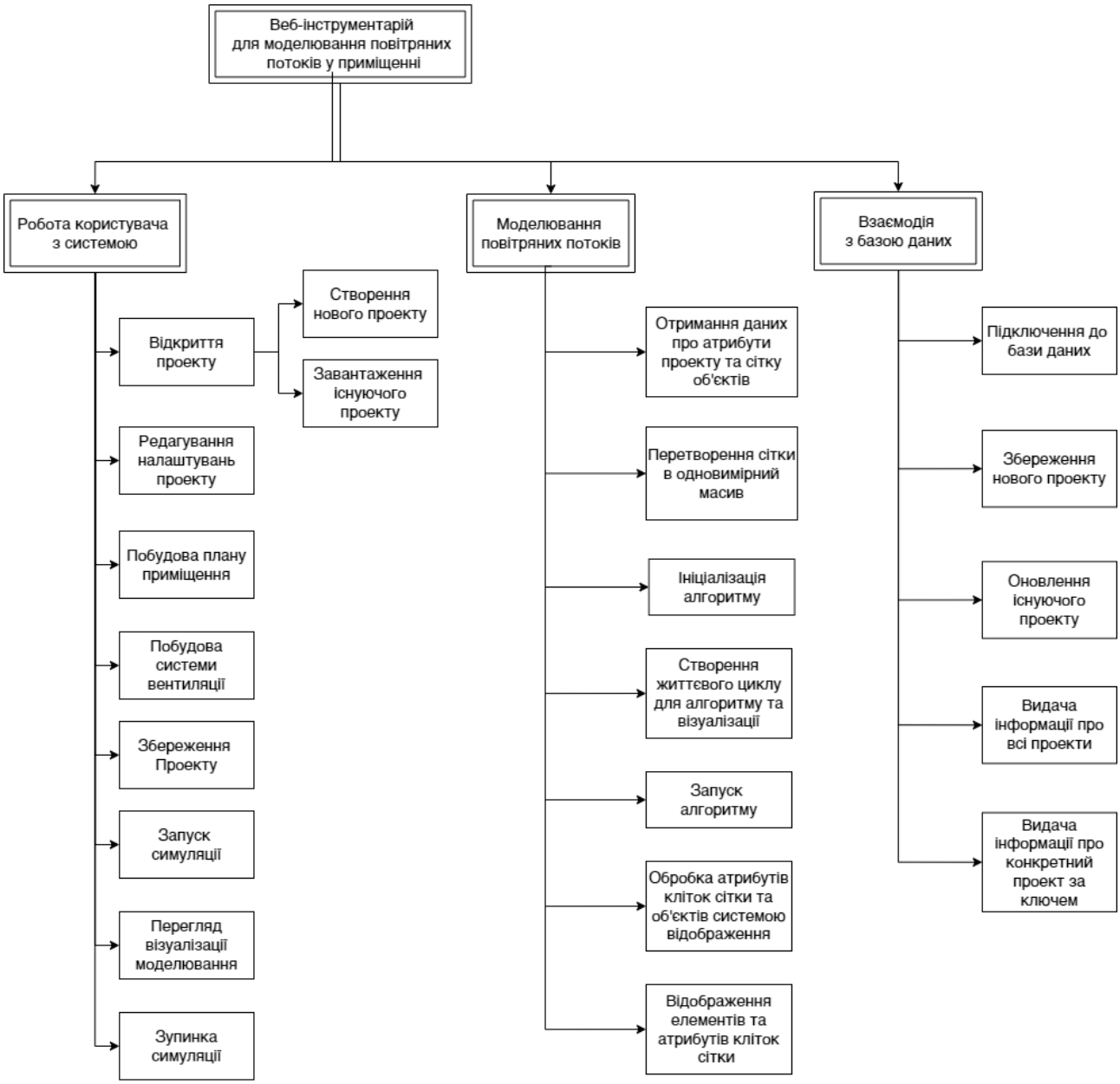

Рисунок 3.11 Дерево функцій веб-інструментарію

На основі дерева функцій було спроектовано діаграму прецедентів (рисунок 3.12) і дерево програмних модулів back-end (рисунок 3.13) та front-end (рисунок 3.14) частин.

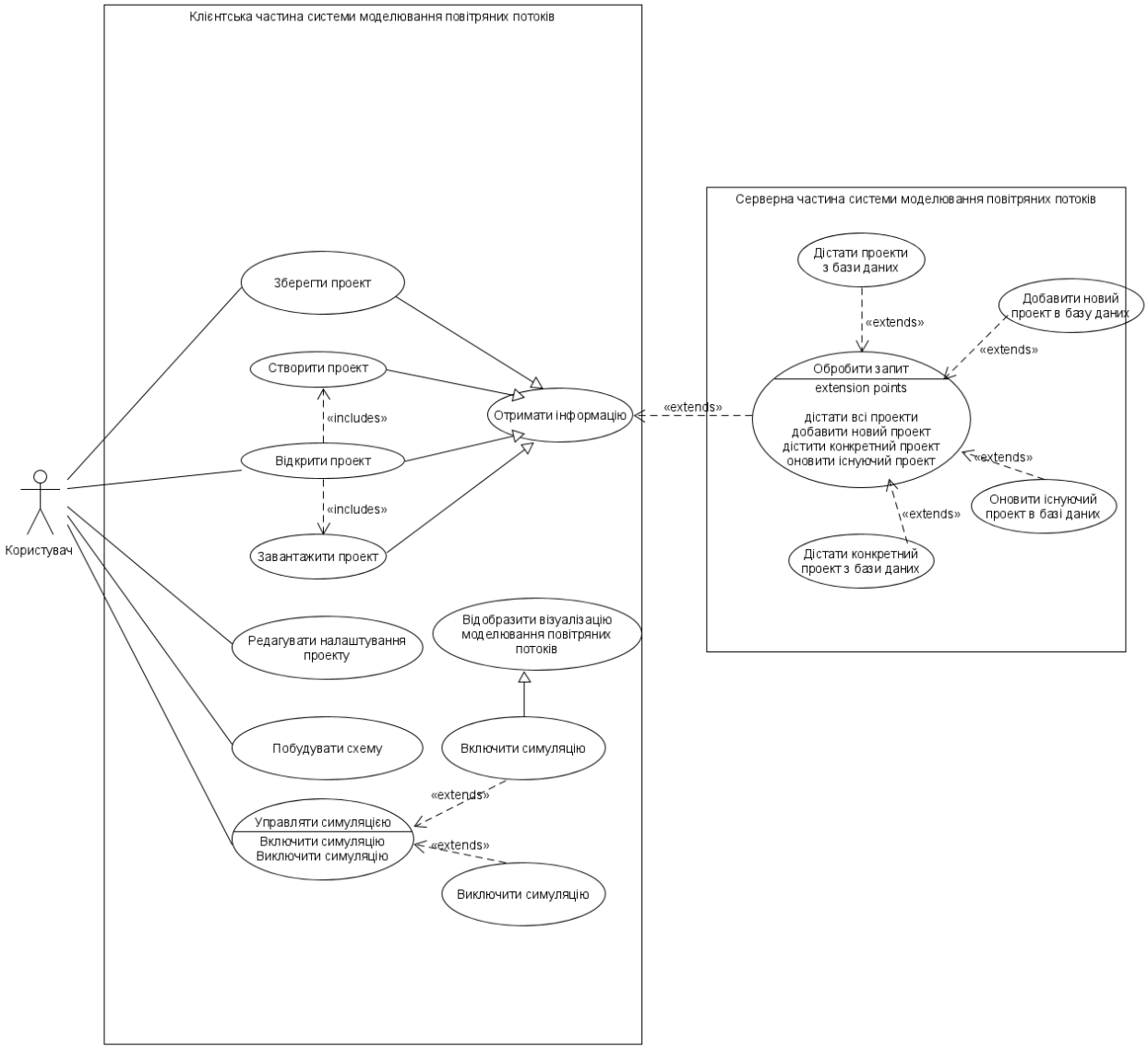

Рисунок 3.12 Діаграма прецедентів системи

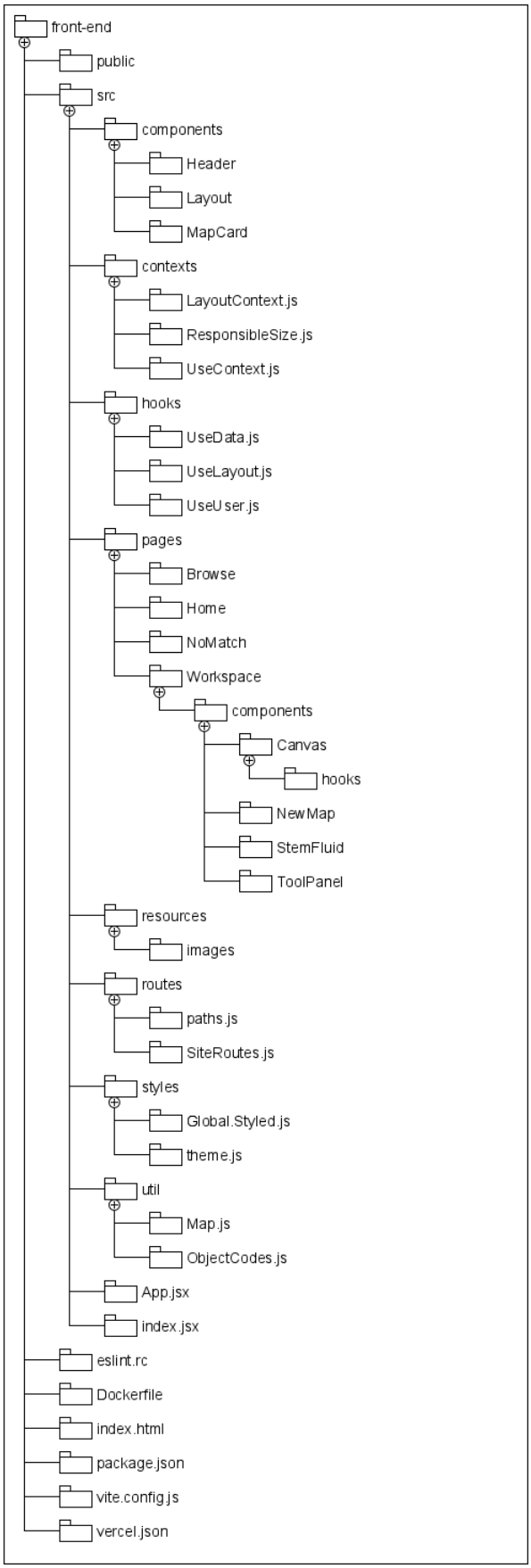

Рисунок 3.13 Дерево програмних модулів front-end частини

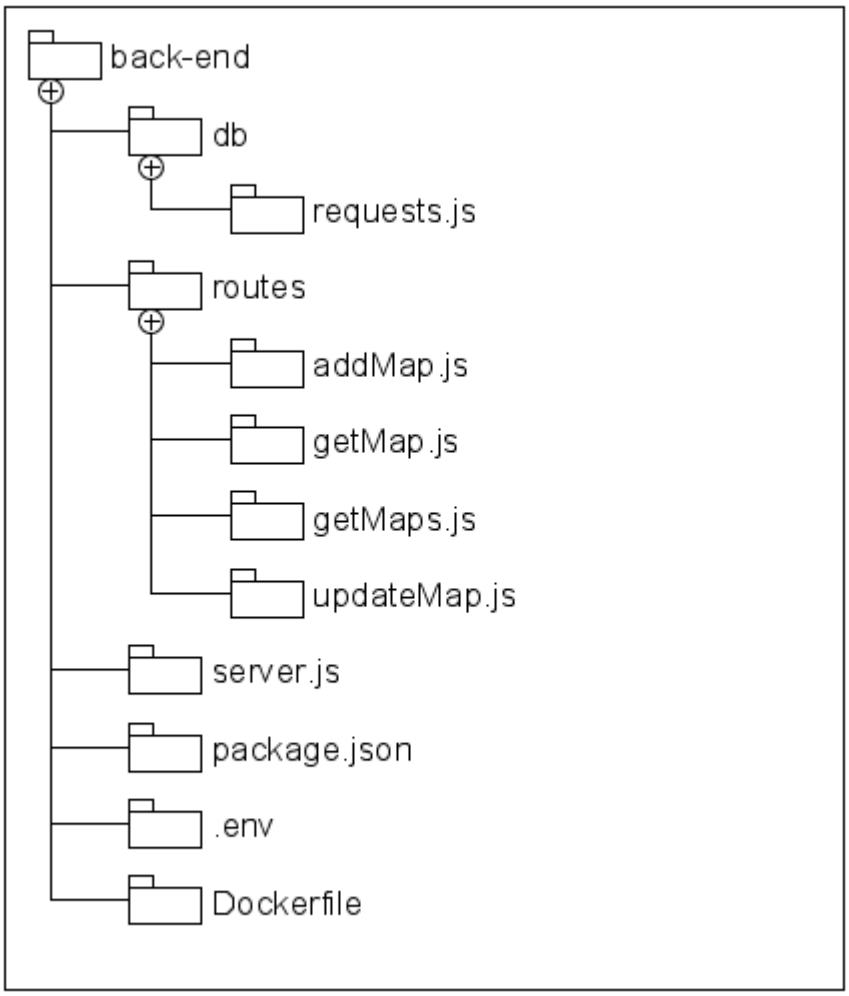

Рисунок 3.14 Дерево програмних модулів back-end частини

3.2.2 Програмна реалізація веб-інструментарію

Розробка застосунку почалась з ініціалізації front-end частини проекту, та створення GitHub репозиторію за допомогою командного рядка.

Спершу необхідно було ініціалізувати NPM пакет за допомогою команди: “yarn init”. Ця команда створює файл package.json, куди в подальшому будуть записуватись залежності front-end частини проекту.

Наступним кроком буде ініціалізація React-проекту. Вона відбувалась за допомогою командного рядка та модуля CRA (Create-React-App). Щоб автоматично створити необхідні для React-проекту файли необхідно прописати команду: “npx create-react-app stam-cfd”, де “stam-cfd” – це назва проекту. Оскільки в якості комплектувальника програми раніше був обраний Vite, необхідно мігрувати з CRA. Для цього створюємо файл “vite.config.js” (рисунок 3.15)

```
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';

import { themeColors } from './src/styles/theme';

export default defineConfig( config: {
  build: {
    outDir: 'build',
  },
  plugins: [react()],
  css: {
    preprocessorOptions: {
      less: {
        modifyVars: {
          'primary-color': themeColors.accent,
          'layout-body-background': themeColors.light,
          'layout-header-background': themeColors.default,
          'body-background': themeColors.light,
          'component-background': themeColors.default,
          'background-color-light': themeColors.defaultLight,
          'menu-item-active-bg': themeColors.defaultLight,
        },
        javascriptEnabled: true,
      },
    },
  },
});
```

Рисунок 3.15 Файл конфігурації бібліотеки Vite, vite.config.js

У файлі vite.config.js серед всього іншого присутня властивість “modifyVars”, куди записані змінні кольорів з попередньо створеного файлу з підібраними кольорами стилю веб-застосунку (рисунок 3.16).

```
export const themeColors = {
  header: '#5CDB95',
  mainContent: '#f0f2f5',
  accent: '#08539e',
  accentLight: 'rgba(5,56,107,0.71)',
  defaultDarker: '#379683',
  default: '#5CDB95',
  defaultLight: '#8EE4AF',
  light: '#EDF5E1',
  red: '#ff5c5c',
  brick: '#AA4A44',
  metal: '#909497',
  window: '#2874A6',
  air: '#2874A6',
  border: '#d9d9d9',
};

export const sizes = {
  maxWidthContent: 900,
  maxWidthLogin: 500,
};
```

Рисунок 3.16 Файл конфігурації кольорів theme.js

Наступним кроком є підключення необхідних бібліотек та модулів. За допомогою команди “yarn install” встановлюємо бібліотеку компонентів “AntDesign”, систему стилізації компонентів “styled-components”, рушій для створення інтерактивної графіки “pixi.js”, HTTP клієнт “Axios”, статистичний аналізатор коду “Eslint”. Повний список залежностей проекту тепер можна переглянути у файлі package.json (додаток А) .

Перед початком написання коду необхідно конфігурувати Eslint для правильного статистичного аналізу коду. В якості бази використовується один з

найпопулярніших конфігураторів: “Airbnb”. Також допишемо налаштування помилок, та попереджень і добавимо необхідні плагіни (Додаток Б).

Для коректної роботи кінцевих точок (сторінок програм), запишемо їх у влаштованому в React роутері (Додаток В). Веб-застосунок буде мати основних 6 кінцевих точок: “workspace” без ключа проекту всередині адреси (створення нового проекту), “workspace” з ключем проекту всередині адреси (робота з проектом), “home” (домашня сторінка), browse (сторінка для пошуку проектів), “no_match” (сторінка, що відповідає за неіснуючу). Всі інші адреси будуть також переадресовані на кінцеву точку “no_match”.

Для комунікації з сервером за допомогою бібліотеки Axios напишемо кастомний хук “UseData”, що дозволить робити потрібні запити з будь-якої частини застосунку. Всередині хука створимо функції для запитів наступних типів: GET-запит на “/maps” – отримати усі проекти, GET-запит на “/map” з вказаним кодом– отримати проект по його коду, POST-запит на “/map” та об'єктом проекту в якості payload – створити новий проект, PUT-запит на “/map” та об'єктом проекту в якості payload – оновити проект. Повна версія реалізації викладена у додатку Г.

Розробка візуальних компонентів для сторінок застосунку буде виконуватись з допомогою бібліотеки AntDesign. Для початку опишемо елементи, загальні для усіх сторінок сайту. Це header (заголовок, верхня частина сайту) та footer (підпис, нижня частина сайту).

Header повинен містити в собі меню, яке дасть змогу користувачу переходити з однієї вкладки на іншу. У лівій частині елементу помістимо логотип проекту, а для обраних вкладок зробимо підсвічування (рисунок 3.17). Також Header був оптимізований для розмірів мобільних девайсів (рисунок 3.18).

Рисунок 3.17 Заголовок сайту (Header)

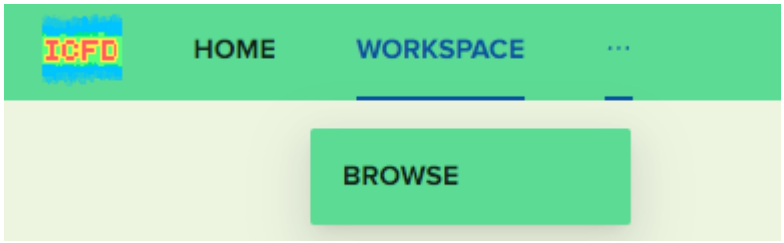

Рисунок 3.18 Мобільна версія заголовку сайту (Header)

Наступним кроком є створення нижньої частини сайту (Footer). В ньому просто вкажемо назву програми, дату, та автора (рисунок 3.19).

Рисунок 3.19

Оскільки з розробкою спільних компонентів покінчено, можна перейти до створення самих сторінок. Для початку розробимо зовнішній вигляд сторінки Home (домашньої, основної). В центр сторінки помістимо назву проекту - ICFD, та розшифровану абревіатуру – indoor computational fluid dynamics. Під назвою розмістимо кнопку з призивом до дії – створенням першого проекту, а під кнопкою залишимо посилання на автора алгоритму обчислювальної гідродинаміки, який був використаний у проекті (рисунок 3.20).

Рисунок 3.20 Головна сторінка застосунку (Home)

Наступним кроком буде розробка сторінки зі створенням нового проекту (Workspace без коду в кінцевій точці). Заголовок з вкладками та нижня частина сторінки залишаються незмінними, а в якості наповнення добавимо надпис “Create new map” (створити новий проект), поле для вводу назви проекту та кнопку “Create” (створити). Результат можна побачити на рисунку 3.21. При натисканні на кнопку “Create” за допомогою кастомного хука UseData, відправляємо запит на створення нового проекту. При успішній спробі створення проекту, сервер повинен повернути згенерований ключ проекту, і застосунок робить автоматичну переадресацію на кінцеву точку: “/map/<ключ проекту>”.

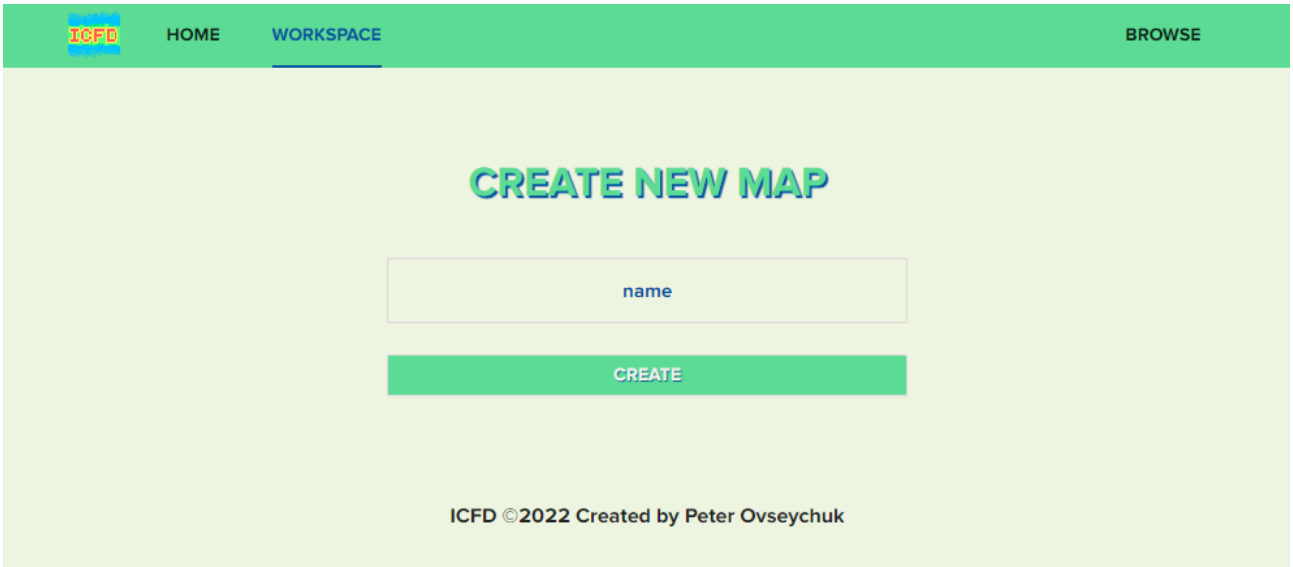

Рисунок 3.21

Для поля з вводом назви проекту задамо обмеження по довжині. Нехай мінімальна довжина буде 3 символи, а максимальна - 20. При введенні некоректних даних виведемо повідомлення про помилку (рисунок 3.22). Також при виникненні помилки зі створенням виведемо повідомлення (рисунок 3.23) (додаток Е).

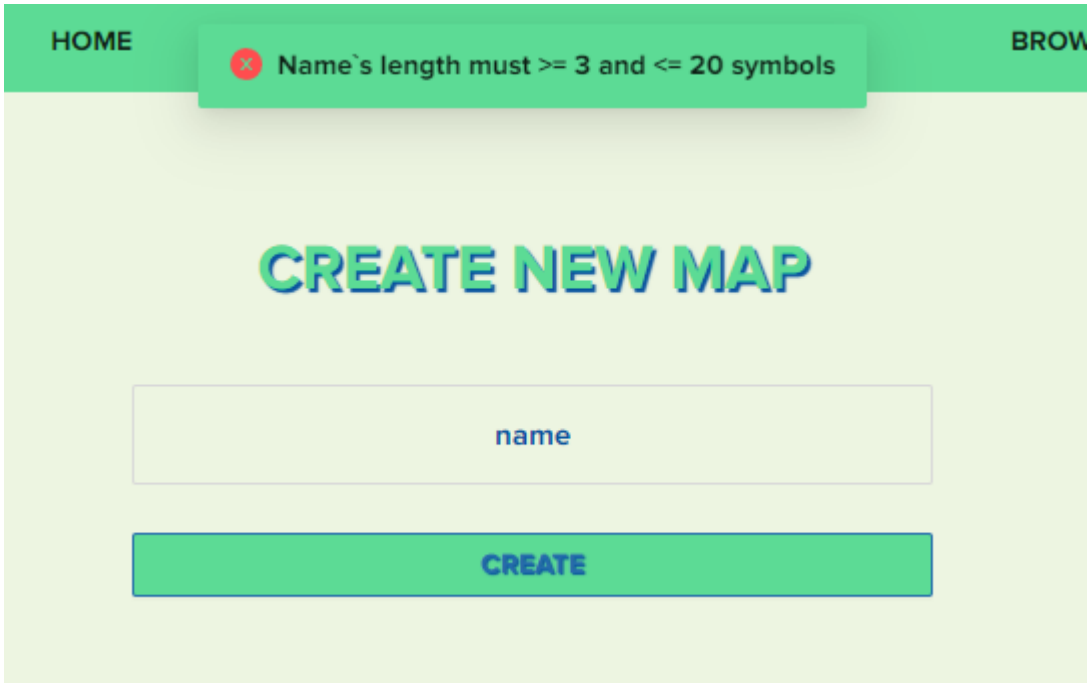

Рисунок 3.22 Повідомлення про некоректно введені дані

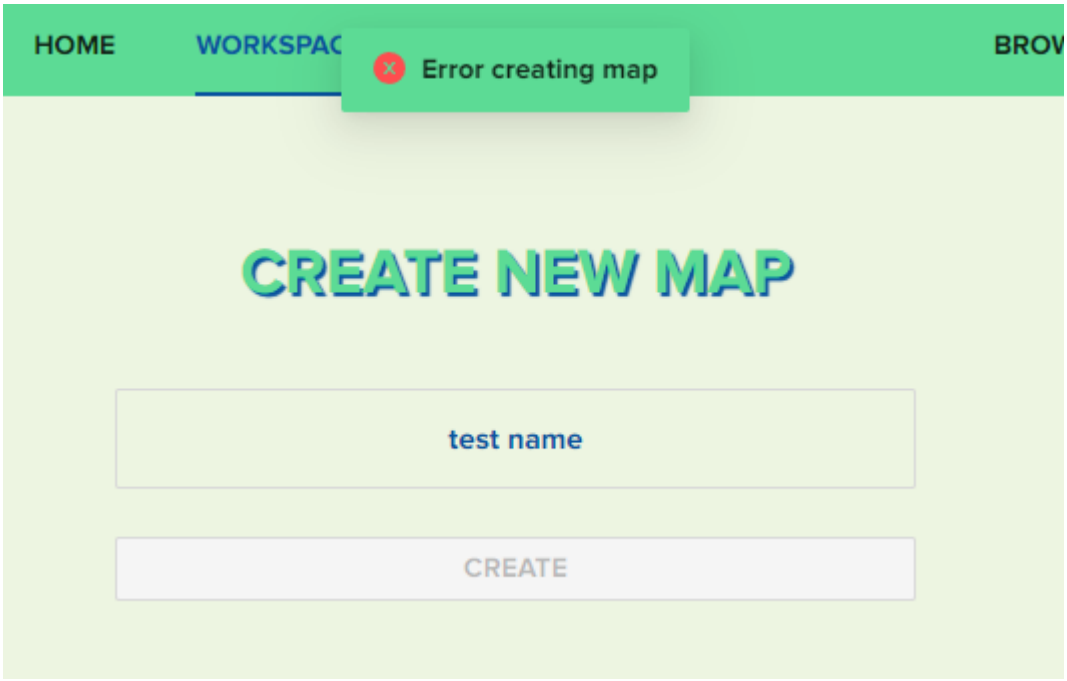

Рисунок 3.23

Після отримання ключа, та переходу до кінцевої точки “/map/<ключ проекту>”, за допомогою кастомного хуку робиться запит на сервер (GET-запит на кінцеву точку “/map” з ключем проекту у якості payload), та back-end частина додатку повертає на front-end частину об'єкт з атрибутами проекту.

Наступним та самим трудомістким завданням є програмна реалізація алгоритму. Для цього створимо клас “Fluid” (додаток Ґ), та обернемо його в кастомний хук “useStemFluid” (додаток Д) для легкої взаємодії з об'єктом речовини ззовні. У класі “Fluid” представимо сітку з атрибутами кліток у вигляді одновимірного масиву. Далі реалізуємо дифузію, адвекцію, вирішення проблеми дивергенції за допомогою вирішення систем рівнянь методом Гауса – Зайделя згідно з оригіналом опису алгоритму “Real-Time Fluid Dynamics for Games” [1].

Потім створимо клас “ICFDMAP” для зручної роботи з проектами користувачів (додаток Є). Цей клас буде в собі містити налаштування рідини в

проекті (в'язкість, дифузія), ключ проекту, розмір сітки, назву, екземпляр класу “Fluid”, двовимірний масив елементів приміщення, та допоміжні функції. До цих функцій можна віднести:

- updateViscosity (оновлення значення в'язкості рідини);
- updateDiffuse (оновлення значення дифузії рідини);
- objectsToStemBound (двовимірного перетворення масиву об'єктів приміщення на одновимірний масив статичних об'єктів для подальшого використання у класі “Fluid”);
- initStemFluid (ініціалізація алгоритму, та створення екземпляру класу “Fluid”);
- addObject (добавити заданий об'єкт в план будівлі на задані координати);
- removeObject (видалити об'єкт з плану будівлі на заданих координатах);
- updateStemBoundRef (оновити одновимірний масив статичних об'єктів у класі “Fluid”);
- generateClearMap (згенерувати пустий масив об'єктів для проекту. Використовується при створенні нового проекту, або при очищенні уже існуючого), IX (приймає координати двовимірного масиву, та вертає координату одновимірного);

Наступним кроком буде створення сторінки з інтерактивним та інтуїтивним конструктором приміщень, за допомогою якого користувач буде змогу створювати плани приміщень за допомогою стін, дверей та вікон, можливість конструювання вентиляційних систем за допомогою вентиляційних решіток, вентиляторів та джерел чистого повітря. На основі побудованого плану користувач зможе виконати моделювання повітряного потоку для свого проекту з інтуїтивною візуалізацією та візуально проаналізувати проблеми у вентиляційній системі. Для цього за допомогою бібліотеки pixi.js створимо компонент Canvas, що буде відповідати за відображення схеми приміщення та візуалізацію повітряних потоків. Дочірніми компонентами Canvas будуть кастомні хуки:

- useCanvas.js – відповідає за створення “життєвого циклу” візуалізації (setInterval), обробку подій, пов'язаних з клієнтськими вхідними даними, обробку функціоналу деяких кнопок з панелі управління симуляцією, зупинку та початок симуляції, реалізацію функціоналу вентиляторів та джерел чистого повітря.
- 2DContextRender.js – відповідає за створення об'єктів та повітряних потоків в якості візуальних елементів на полотні. Градація швидкостей потоків для візуалізації зображена на рисунку 3.24
- renderObjects.js – оптимізує відображення складних елементів.
- UseResponsibleSize – слідкує за необхідним розміром Canvas відносно розміру екрану

Результат розробки компоненту Canvas разом з алгоритмом симуляції повітряних потоків зображено на рисунку 3.25.

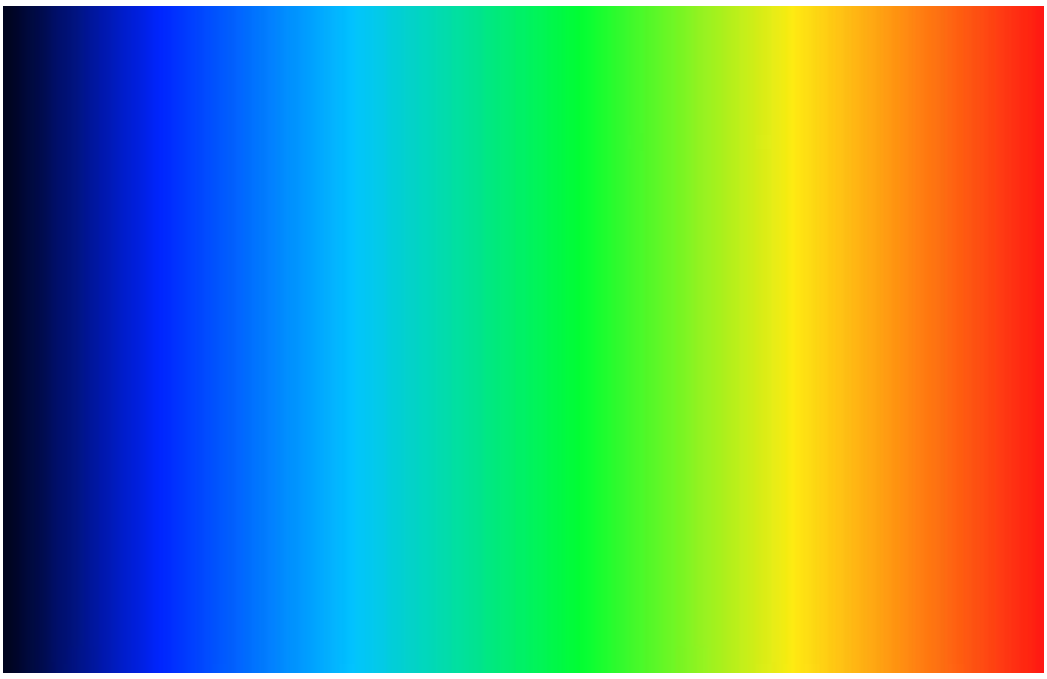

Рисунок 3.24 Градієнт відображення швидкостей повітряних потоків

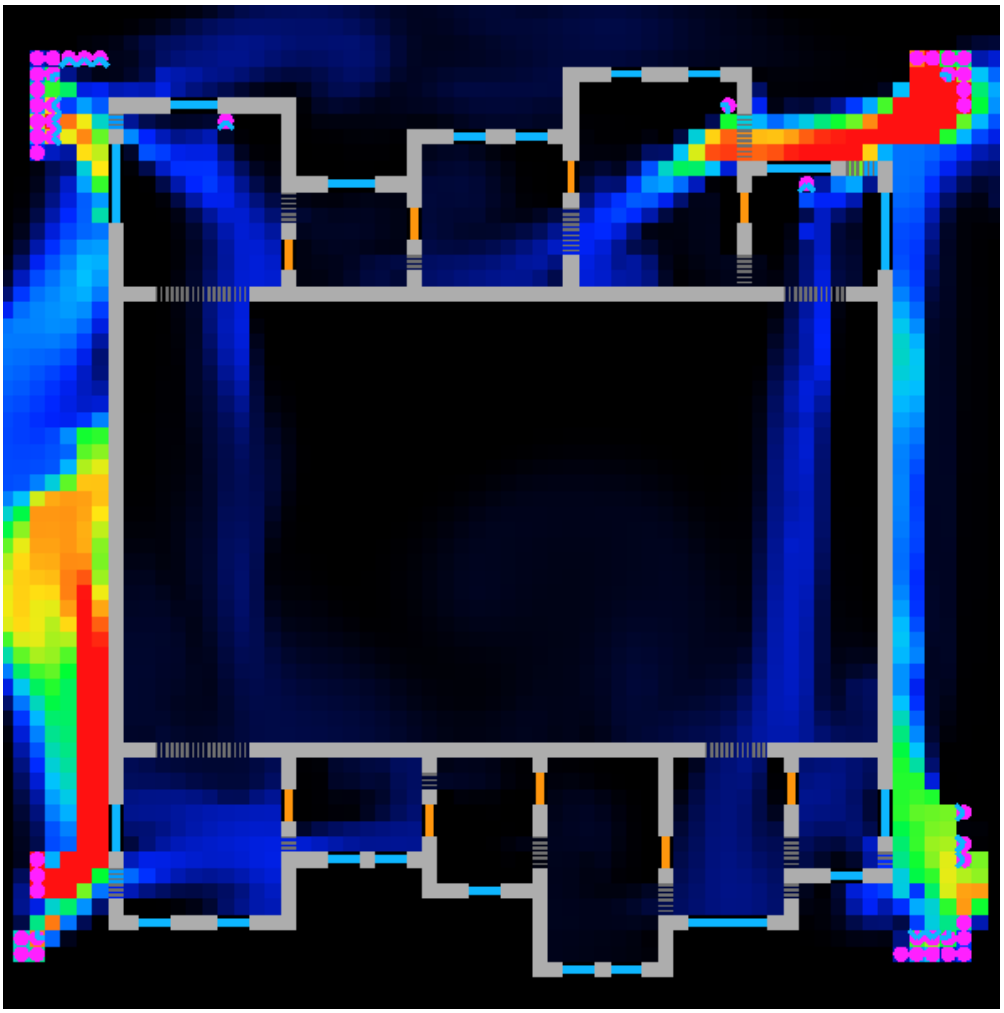

Рисунок 3.25 Візуалізація проекту за допомогою Canvas

Наступною задачею буде створення компоненту ToolBar (панель інструментів, рисунок 3.26). Вона повинна містити в собі усі елементи управління конструктором, розбиті на 5 категорій: MANAGE (управління проектом), CURSOR (вибір курсору), BUILD (основні елементи для побудови схеми приміщення та вентиляційної системи), CONTROL (елементи для ручного контролю потоками повітря), SIMULATION (управління симуляцією). Перелік реалізованих елементів:

– SETTINGS – відкриває налаштування атрибутів рідини в проекті (рисунок 3.27);

- SAVE – зберігає проект, відправляючи його на сервер за допомогою запиту PUT на кінцеву точку “/map” з об'єктом проекту в якості payload;
- CLEAR AIR – очищає щільність повітря на сітці симуляції;
- CLEAR OBJECTS – видаляє всі об'єкти з плану приміщення;
- DEFAULT - звичайний курсор без функціоналу;
- ERASER – видалення об'єктів курсором миші. Також для виконання цієї функції є можливість використовувати ПКМ, при будь-яких обраних об'єктах;
- WALL – стіна, не пропускає повітря;
- WINDOW – вікно, не пропускає повітря;
- DOOR – двері, не пропускають повітря;
- VENT – вентиляційні решітки, пропускають повітря;
- AIR SOURCE – джерело чистого повітря;
- FAN – вентилятор з можливістю вибору напряму застосування сили потоку (рисунок 3.28).
- POUR – добавити щільність повітря у вказану точку.
- MOVE – перемістити щільність повітря за рухом миші.
- STOP – зупинити симуляцію.
- START – запустити симуляцію.

Рисунок 3.26 Зовнішній вигляд компоненту ToolBar

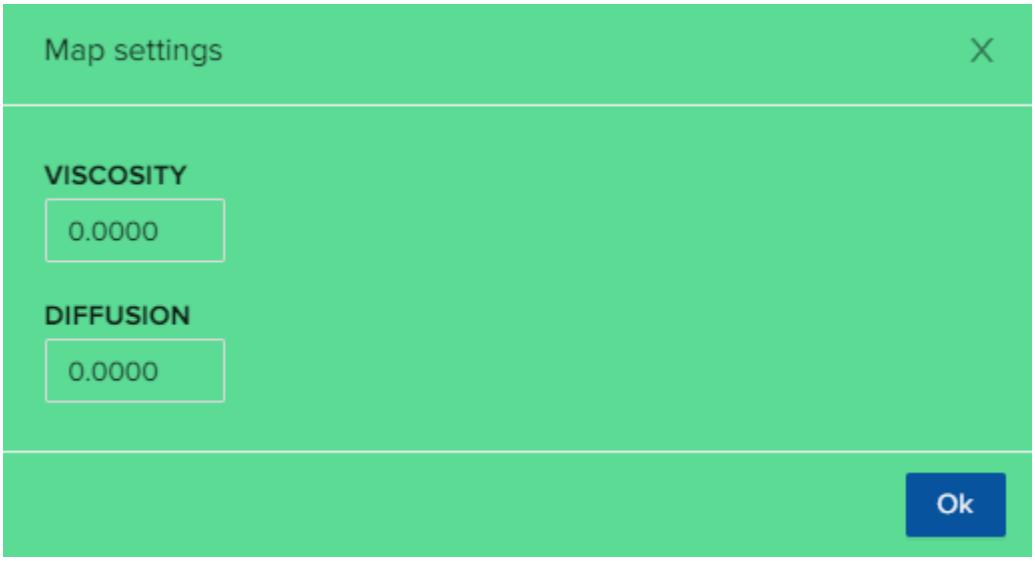

Рисунок 3.27 Панель Settings

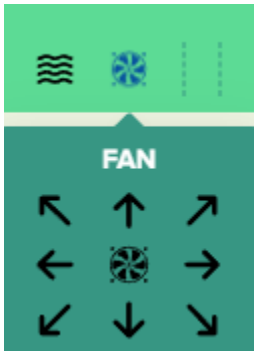

Рисунок 3.28 Вибір напряму вентилятора

Далі розмістимо компоненти Canvas та ToolBar на сторінці Workspace з ключем проекту всередині адреси. Також добавимо BreadCrumb компонент для відображення назви проекту. Вихідний зовнішній вигляд сторінки Workspace зображено на рисунку 3.29.

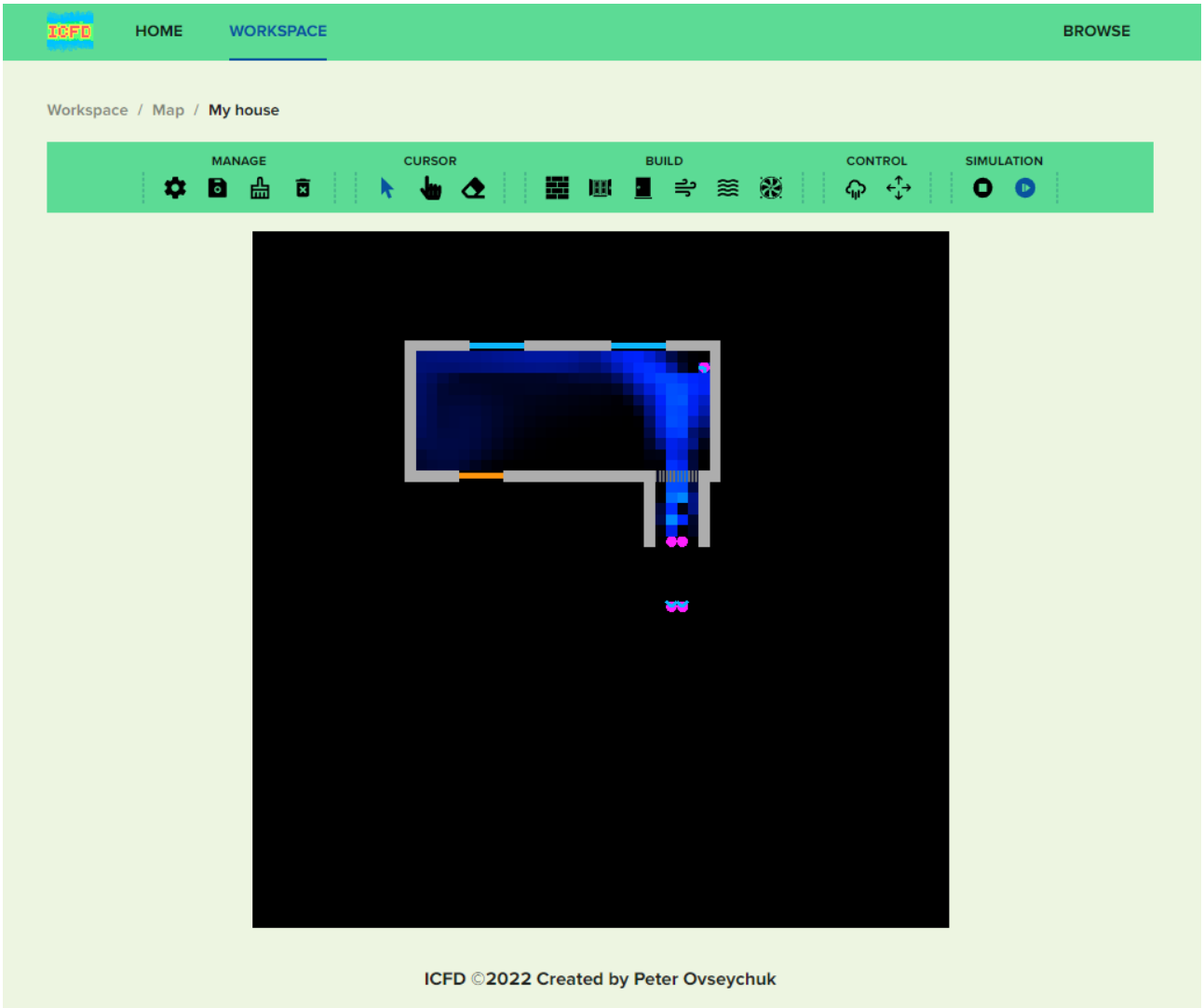

Рисунок 3.29 Зовнішній вигляд сторінки Workspace

Залишилось розробити сторінку Browse для пошуку проектів. Для цього при заході на сторінку, за допомогою кастомного хуку UseDate відбудеться GET-запит на кінцеву точку “/maps”, що поверне клієнту список всіх проектів. Після цього візуалізуємо кожен проект у вигляді карточки з основною інформацією про нього, наприклад, ім'я, ключ, розмір поля. Для швидкого доступу до потрібного проекту реалізовано поле з пошуковим запитом (рисунок 3.30).

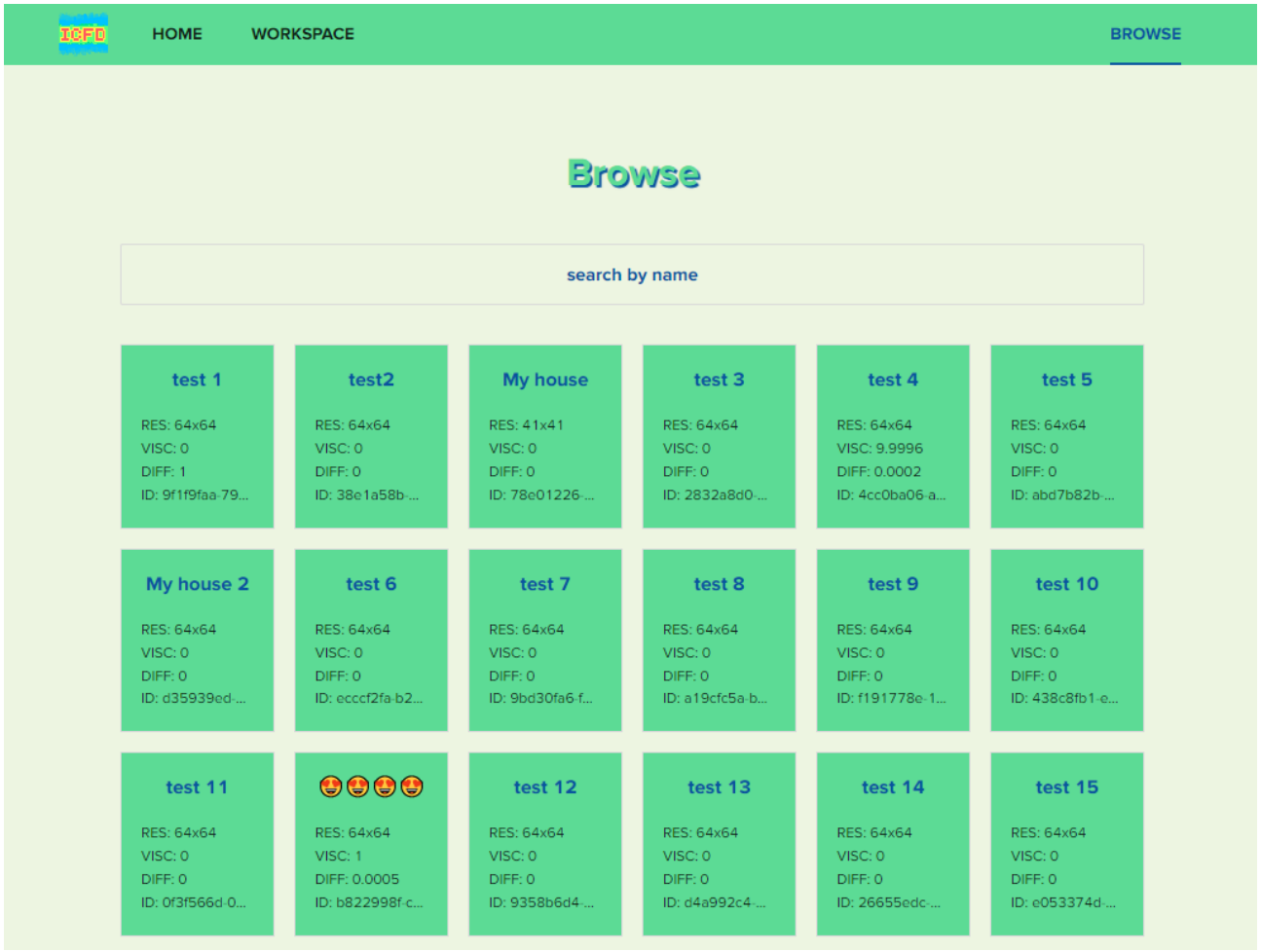

Рисунок 3.30

Базу даних MongoDB було ініціалізовано за попередньою схемою. За допомогою адмін-панелі було створено користувача – сервер, який міг відправляти запити на читання та редагування записів (рисунок 3.31).

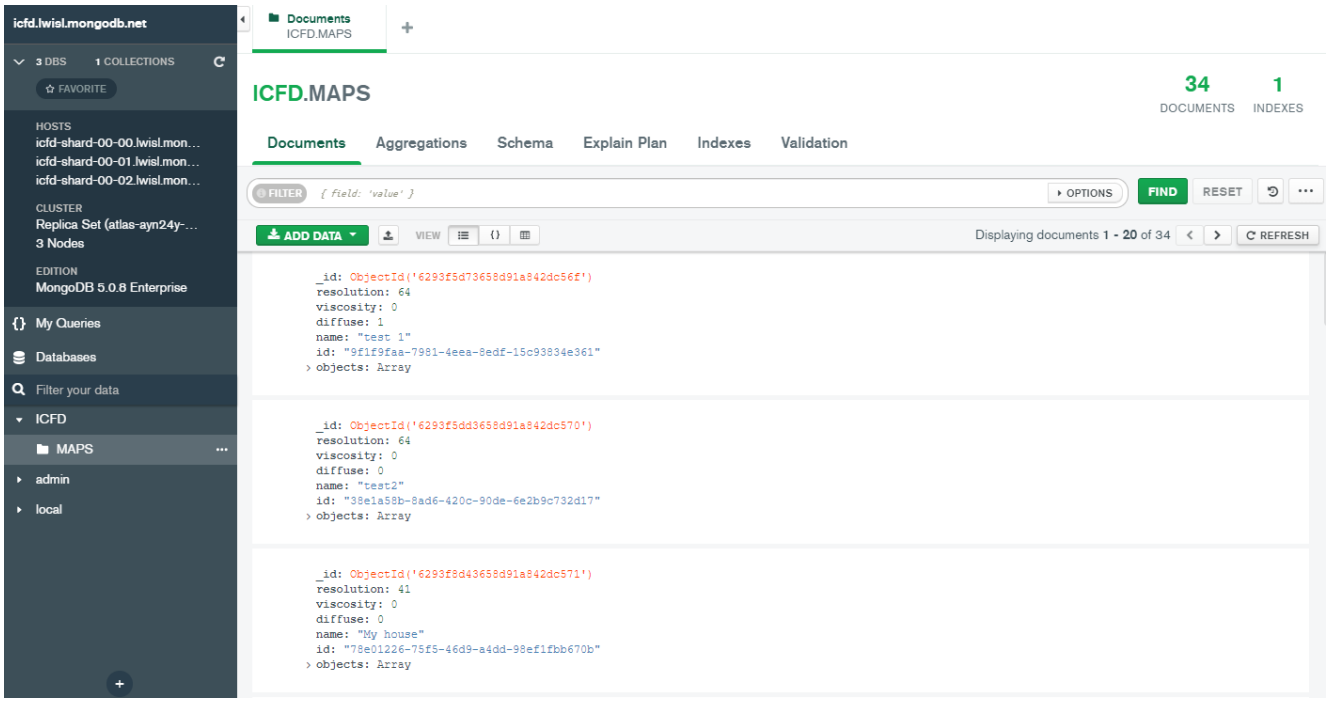

Рисунок 3.31 Зовнішній вигляд записів в базі даних MongoDB

Для створення серверної частини застосунку було ініціалізовано NPM - модуль, подібно до того, як це було зроблено для front-end частини. Далі було створено файл server.js, в якому ініціалізуються плагіни для веб-фреймворка Fastify, підключаються кінцеві точки для запитів з боку front-end, та запускається сервер. Для обробки запитів було створено 4 окремих модулі, що обробляють GET – запити на “/map” та “/maps”, POST – запит на “/map”, PUT – запит на “map”. Під час обробки кожного запиту back-end частина застосунку звертається через NPM модуль “MongoDB” до бази даних, та зберігає, добавляє необхідну інформацію (додаток З).

Для полегшення подальшого розгортання програмного забезпечення було вирішено обернути front-end та back-end в Docker контейнер. Конфігурація docker-compose зображена на рисунку 3.32.

```yaml
version: "3"
services:
  backend:
    build: ./back-end
    ports:
      - "5001:5000"
  frontend:
    build: ./front-end
    ports:
      - "3001:3000"
    environment:
      - NODE_OPTIONS=--openssl-legacy-provider
```

Рисунок 3.32 Схема docker-compose.yml

Розгортання програмного забезпечення відбувалось за допомогою Vercel, шляхом синхронізації проекту у Vercel та репозиторія на GitHub (рисунок 3.33). Під час налаштування було конфігуровано кореневу папку проекту.

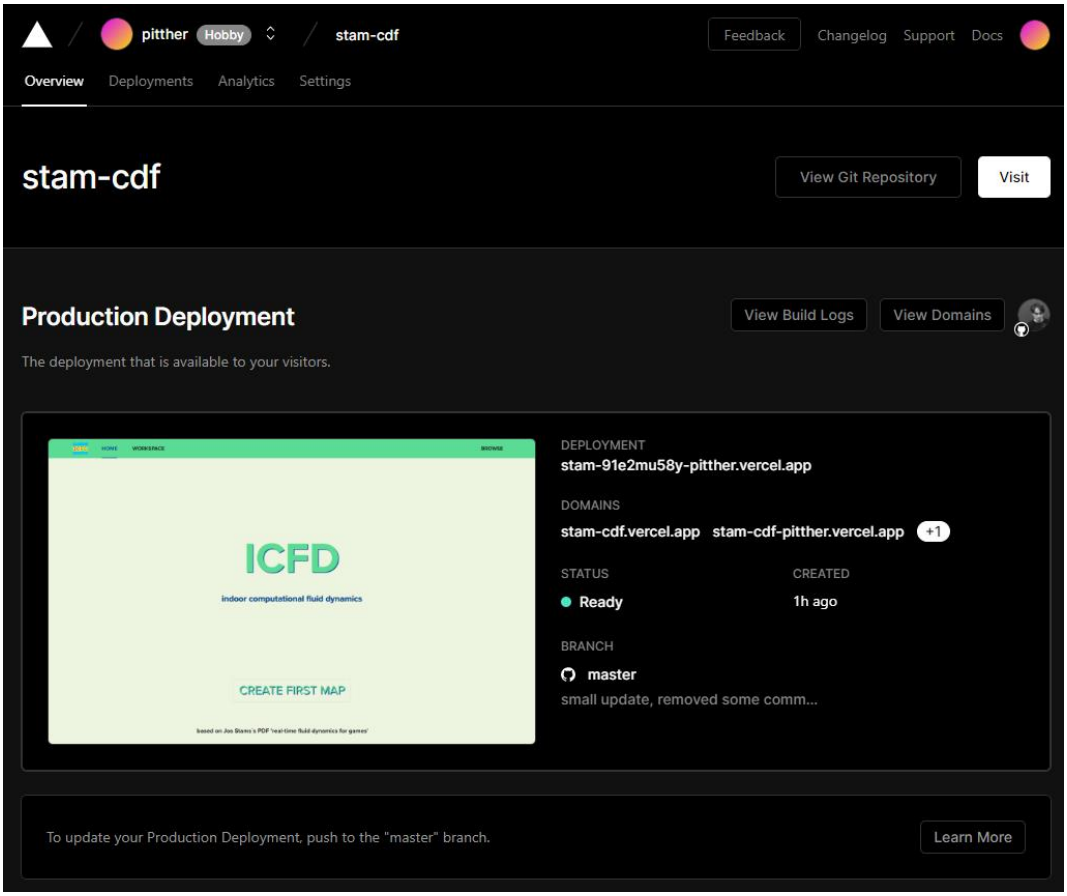

Рисунок 3.33 Сторінка з проектом на Vercel

Вихідний код можна переглянути за посиланням на GitHub репозиторій: https://github.com/pitther/stam-cfd-course-work/

3.2.3 Тестування системи

Для тестування веб-інструментарію моделювання повітряного потоку для зниження ризику розповсюдження вірусів в приміщенні побудуємо проекти для двох систем вентиляції однієї будівлі. Перший план буде прикладом поганої вентиляції з високим ризиком поширення вірусів всередині приміщення

(рисунок 3.34), а другий план буде прикладом покращеної вентиляції (рисунок 3.35).

Для розробки проектів використаємо усі елементи, розроблені для побудови приміщень з компоненту ToolBar.

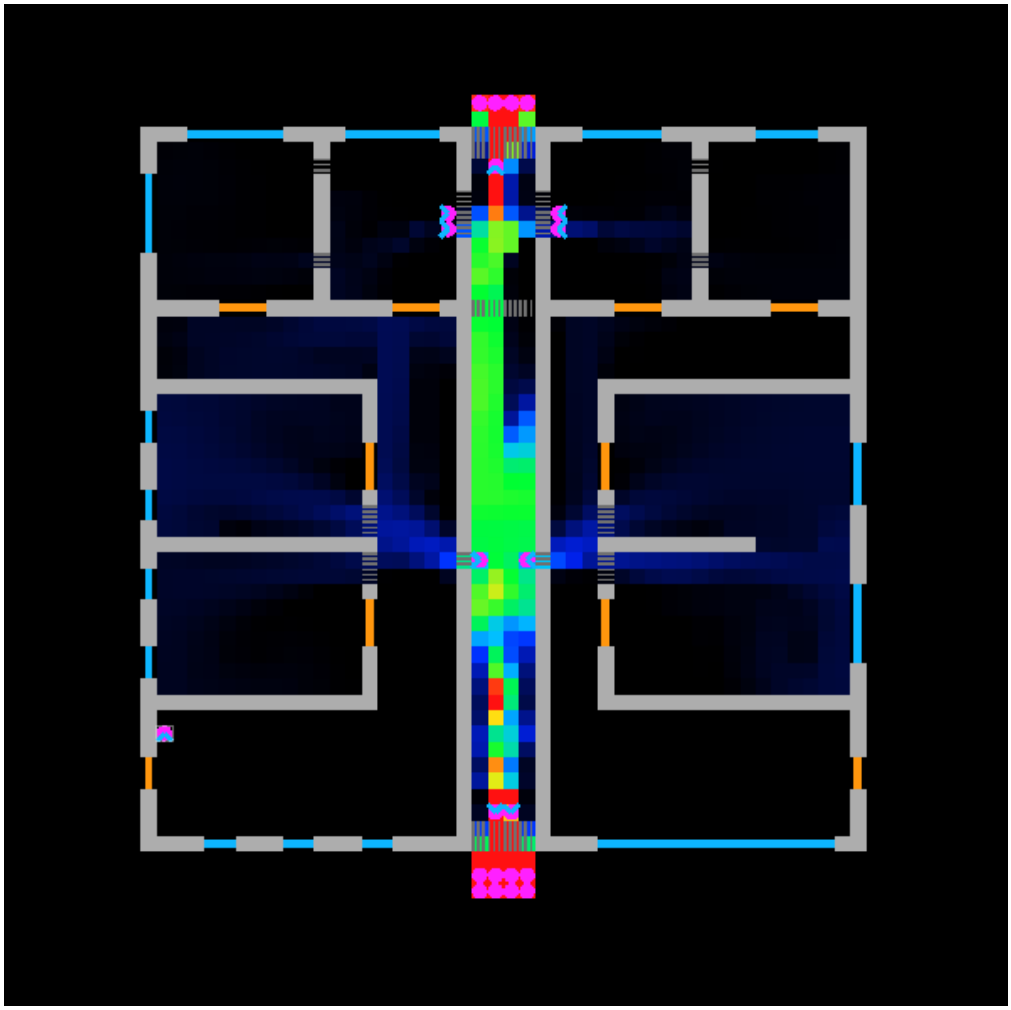

Рисунок 3.34 Приклад поганої вентиляції

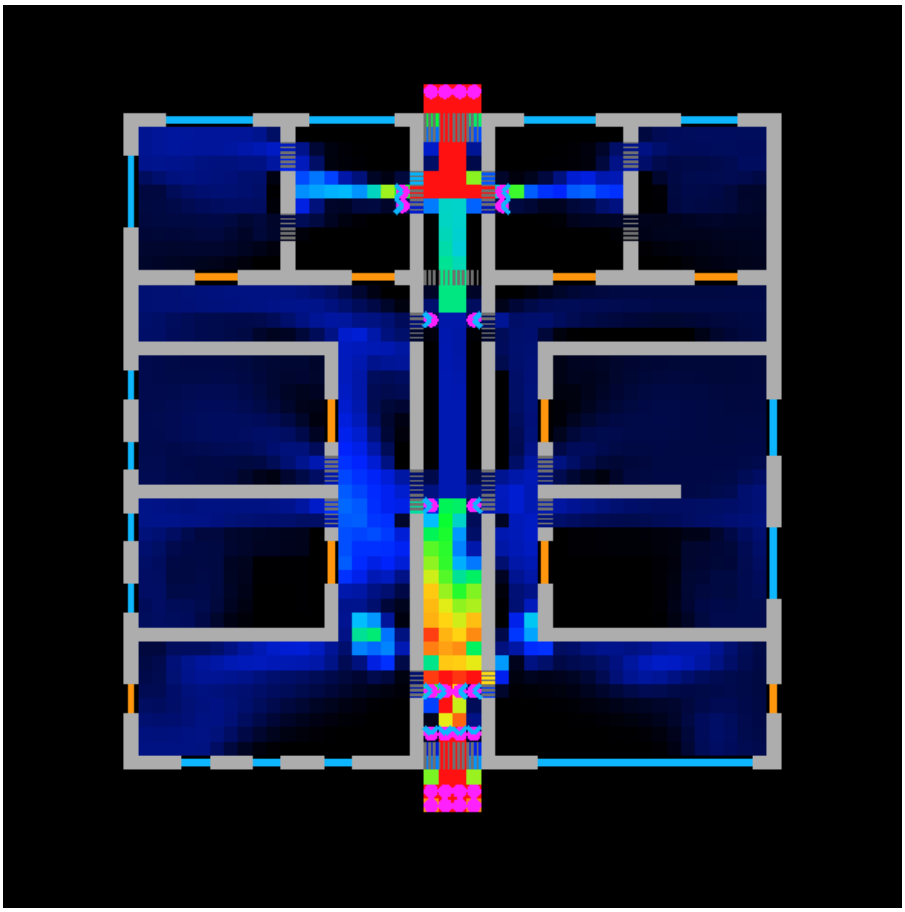

Рисунок 3.35 Приклад хорошої вентиляції

Градація кольору відображає швидкість потоку повітря. Чорний колір – нульова швидкість, синій – мала, зелений – середня, червоний – висока. Чим більша швидкість потоку в окремо взятій кімнаті – тим менше шансів у вірусу заразити людину у цьому приміщенні.

Майже вся площа кімнат в другому проекті (рисунок 3.35) покрита синім кольором рівномірно. Це означає, що в них присутній задовільний рух повітря, і відповідно ризик заразитись вірусом в таких приміщеннях – менший. Чого не можна сказати про перший проект (рисунок 3.34). На ньому замітно менша площа покрита потоками повітря задовільної швидкості, а в деякі кімнати навіть не поступає чисте повітря.

Можемо зробити висновок, що розроблена система працює вірно, та видає достовірні результати моделювання повітряних потоків. Згідно наведеного вище тестового прикладу можна покращити якість вентиляції будь-якої будівлі, попередньо проаналізувавши потоки повітря в розробленому веб-інструментарії.

3.2.3 Інструкції щодо встановлення

Для встановлення локальної версії програмного продукту, користувачу необхідно мати встановлений Docker.

Відкриваючи проект необхідно виконати наступні команди:

1. docker-compose -f docker-compose.yml up – інсталює усі залежності та бібліотеки у проекті, прокладає необхідні порти, та запускає front-end та back-end частини застосунку;
2. docker ps – для перевірки наявності запущеного процесу застосунку;

Після чого користувачеві необхідно перейти за посиланням http://localhost:3000/.

Іншим варіантом використання програмного продукту є перехід за іншим посиланням: https://stam-cdf.vercel.app. Це розгорнутий додаток на платформі Vercel.

3.2.4 Експлуатація системи користувачем

Для того, щоб почати роботу з веб-інструментарієм, необхідно зайти на головну сторінку сайту та натиснути на кнопку “Create first map” (рисунок 3.36).

Рисунок 3.36 Головна сторінка системи

Після цього користувача переключить на іншу сторінку зі створенням проекту. Другим кроком буде введення імені проекту в поле “name” (рисунок 3.37, пункт 1), та натискання на кнопку “Create” (рисунок 3.37, пункт 2).

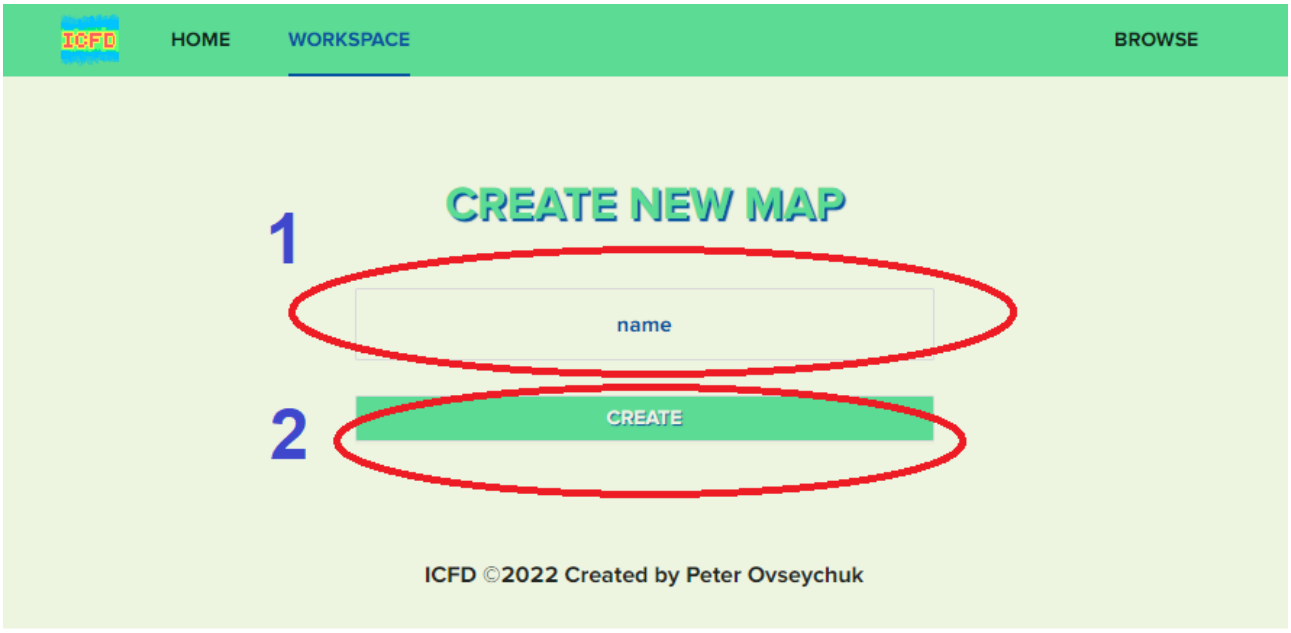

Рисунок 3.37

Після переходу до редагування проекту перед користувачем буде відображено конструктор для створення плану будівлі. У верхній частині конструктору знаходиться панель управління, у нижній – робоче поле. На панелі управління знаходяться елементи, за допомогою яких користувач буде створювати план будівлі та схему вентиляції на робочому полі. На ньому також буде відбуватись симуляція потоків повітря після запуску моделювання. Елементи панелі управління (рисунок 3.38):

1. SETTINGS – відкриває налаштування атрибутів рідини в проекті;
2. SAVE – зберігає проект;
3. CLEAR AIR – очищає щільність повітря на сітці симуляції;
4. CLEAR OBJECTS – видаляє усі об'єкти з плану приміщення;
5. DEFAULT - звичайний курсор без функціоналу;
6. ERASER – видалення об'єктів курсором миші. Також для виконання цієї функції є можливість використовувати ПКМ, при будь-яких обраних об'єктах;
7. WALL – стіна, не пропускає повітря;
8. WINDOW – вікно, не пропускає повітря;

9. DOOR – двері, не пропускають повітря;
10. VENT – вентиляційні решітки, пропускають повітря;
11. AIR SOURCE – джерело чистого повітря;
12. FAN – вентилятор з можливістю вибору напряму застосування сили потоку.
13. POUR – добавити щільність повітря у вказану точку.
14. MOVE – перемістити щільність повітря за рухом миші.
15. STOP – зупинити симуляцію.
16. START – запустити симуляцію.

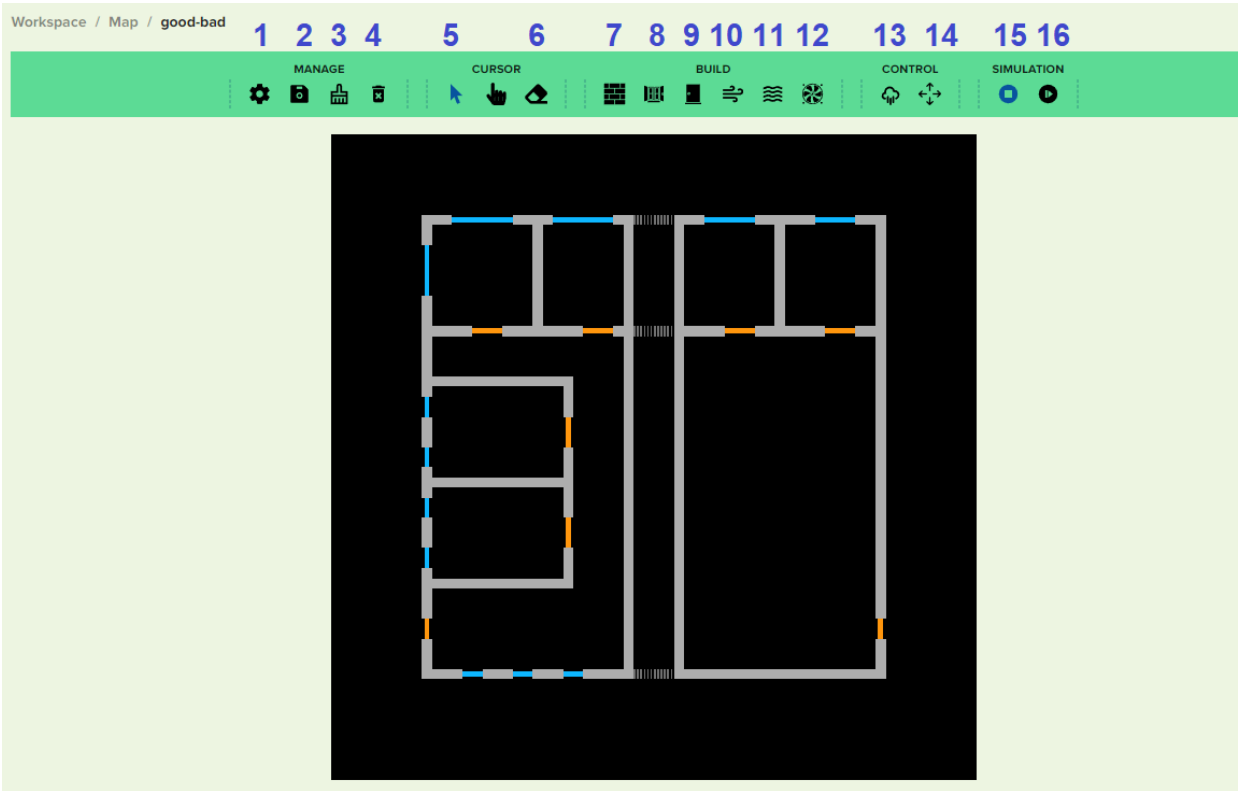

Рисунок 3.38 Робочий простір з пронумерованими елементами панелі управління

Для запуску симуляції необхідно натиснути на кнопку Start, після чого запуститься моделювання повітряних потоків у створеній схемі приміщення, та користувач може почати аналіз повітряної системи (рисунок 3.39, 3.40). Градація

кольору відображає швидкість потоку повітря. Чорний колір – нульова швидкість, синій – мала, зелений – середня, червоний – висока. Чим більша швидкість потоку в окремо взятій кімнаті – тим менше шансів у вірусу заразити людину у цьому приміщенні.

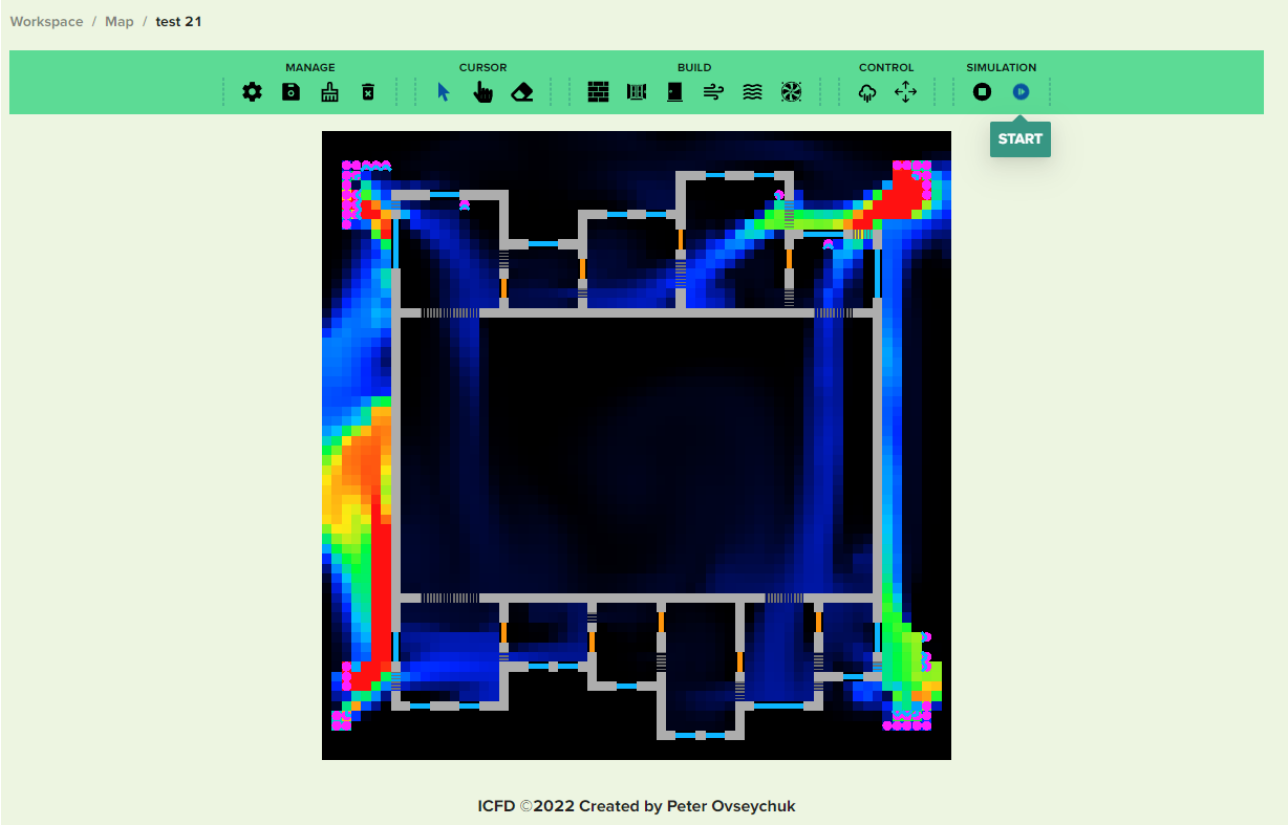

Рисунок 3.39 Запуск симуляції

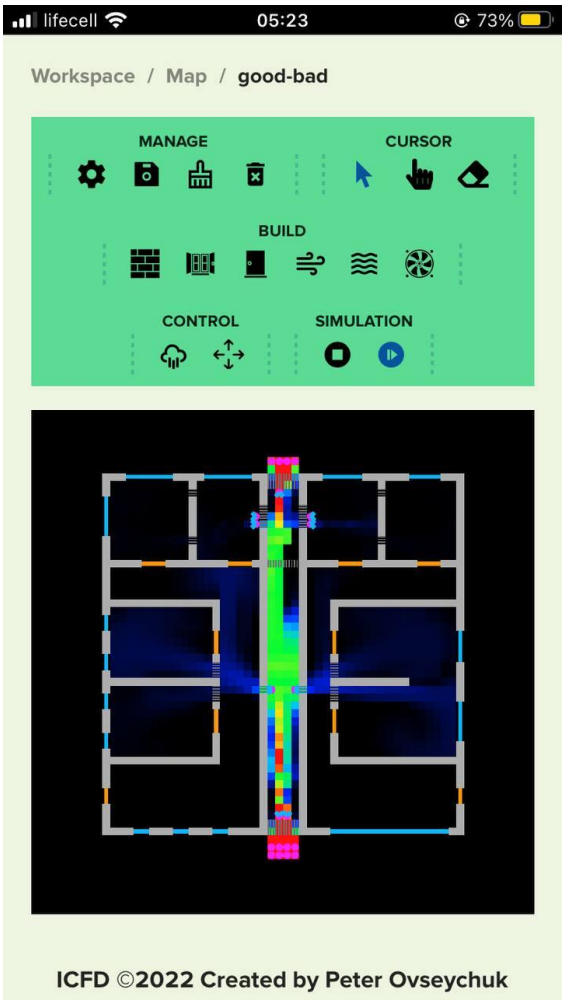

Рисунок 3.40 Запуск симуляції на мобільній версії застосунку

Також для пошуку проектів по імені створена вкладка Browse. Достатньо вписати назву проекту в поле пошуку, і система знайде потрібні вам результати. Щоб почати роботу зі знайденими результатами – просто натисніть на відповідну карточку з назвою та короткою інформацією про атрибути проекту (рисунок 3.41).

Рисунок 3.41 Пошук проектів за допомогою вкладки Browse

ВИСНОВОК

У рамках роботи було проведено аналіз впливу вентиляції на розповсюдження вірусів у приміщеннях на прикладі вірусу SARS-CoV-2. Також було зроблено аналітику можливих підходів до вирішення проблеми та аналіз програм-конкурентів, які вміщають в собі функціонал моделювання повітряних потоків. Було визначено об'єкт, предмет, мету роботу, а також зацікавлені сторони. Також були висунуті функціональні та нефункціональні вимоги до застосунку.

Під час роботи над дипломною роботою було створено веб-інструментарій для моделювання повітряного потоку у приміщенні задля зниження ризику зараження вірусом. Для реалізації веб-інструментарію в якості CFD алгоритму був обраний варіант методу обчислення гідродинаміки у реальному часі під назвою “Real-Time Fluid Dynamics for Games”, акцент роботи якого робиться на стабільності та швидкості. [1] Було детально розібрано принцип роботи алгоритму, та основні методи і закони, що використовуються під час обчислення гідродинаміки.

Під час розробки веб-інструментарію було проведено аналітику, та досліджено багато мов програмування та технологій розробки веб-застосунків. Виходячи з особистого досвіду, та порад від досвідчених розробників було виявлено декілька потрібних технологій для вирішення поставленої проблеми. В якості мови основної мови програмування було обрано JavaScript, а для створення користувацького інтерфейсу – бібліотеку React. Для серверної частини було обрану мову NodeJs, що дозволяє легко працювати з асинхронними запитами.

В результаті було отримано веб-інструментарій з зручним, мінімалістичним та сучасним інтерфейсом користувача, що, за допомогою розробленого конструктора приміщень дозволяє навіть новому користувачу

створити, зберегти, та поділитись проектом з іншими людьми, а також може працювати на мобільних пристроях. Сам процес створення будівлі відбувається за допомогою розробленого інтуїтивного конструктору, який містить в собі фундаментальні елементи будівлі, такі, як стіни, вікна та двері, а також елементи для створення систем вентиляції. В результаті користувач отримає візуалізацію роботи вентиляції в його цільовому приміщенні, та зможе проаналізувати її якість.

Звичайно, точність отриманої симуляції в 2D просторі не є дуже великою в порівнянні з важкими хмарними обчисленнями програм конкурентів, проте вона надає можливість безкоштовно та швидко створити необхідну вам модель приміщення, та дослідити будівлю на якість вентиляції.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Real-Time Fluid Dynamics for Games Jos Stam Alias | wavefront 210 King Street East Toronto, Ontario, Canada M5A 1J7
2. E. A. Meyerowitz, A. Richterman, R. T. Gandhi and P. E. Sax, “Transmission of SARS-CoV-2: a review of viral, host, and environmental factors,” Annals of internal medicine, 2020.
3. RM Jones та LM Brosseau, “Аерозольна передача інфекційних захворювань”, J Occup Environ Med., vol. 57, № 5, С. 501-508, 2015.
4. G. Kampf, Y. Brüggemann, H. Kaba, J. Steinmann, S. Pfaender, S. Scheithauer and E. Steinmann, “Potential sources, modes of transmission and effectiveness of prevention measures against SARS-CoV-2,” Journal of Hospital Infection, 2020.
5. E. P. Vejerano and L. C. Marr, “Physico-chemical characteristics of evaporating respiratory fluid droplets.,” J. R. Soc. Interface , vol. 15, p. 20170939, 2018.
6. LM Casanova, S. Jeon, WA Rutala, DJ Weber and MD Sobsey, «Вплив температури повітря та відносної вологості на виживання коронавірусу на поверхнях», Appl Environ Microbiol, vol. 76, № 9, с. 2712-2717, 2010.
7. SARS-CoV-2 and Surface (Fomite) Transmission for Indoor Community Environments [Електронний ресурс]. Режим доступу: https://www.cdc.gov/coronavirus/2019-ncov/more/science-and-research/surface-transmission.html
8. Ventilation in Buildings [Електронний ресурс]. Режим доступу: https://www.cdc.gov/coronavirus/2019-ncov/community/ventilation.html
9. Filtration and air cleaning summary [Електронний ресурс]. Режим доступу: https://www.ashrae.org/technical-resources/filtration-disinfection#mechanical
10. PixiJS [Електронний ресурс]. Режим доступу: https://pixijs.download/release/docs/index.html
11. Styled-components adapting based on props [Електронний ресурс]. Режим доступу: https://styled-components.com/docs/basics#adapting-based-on-props

12. Quick Start with React router [Електронний ресурс]. Режим доступу: https://v5.reactrouter.com/web/guides/quick-start
13. Eslint project configuration [Електронний ресурс]. Режим доступу: https://eslint.org/
14. NodeJS [Електронний ресурс]. Режим доступу: https://nodejs.org/uk/docs/
15. Axios HTTP client [Електронний ресурс]. Режим доступу: https://github.com/axios/axios
16. AntDesign component styles [Електронний ресурс]. Режим доступу: https:/ant.design/components/
17. Vite Config [Електронний ресурс]. Режим доступу: https://vitejs.dev/config/
18. ReactJS [Електронний ресурс]. Режим доступу: https://uk.reactjs.org/docs/
19. South Africa announces a new coronavirus variant. [Електронний ресурс]. Режим доступу: https://www.nytimes.com/2020/12/19/world/south-africa-announces-a-new-coronavirus-variant.html
20. Danish government halts plans to kill more than 15 million minks over coronavirus scare [Електронний ресурс]. Режим доступу: https://www.washingtonpost.com/world/2020/11/10/denmark-halts-mink-cull/
21. Investigation of novel SARS-COV-2 Variant [Електронний ресурс]. Режим доступу: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/959438/Technical_Briefing_VOC_SH_NJL2_SH2.pdf
22. Heavily mutated Omicron variant puts scientists on alert [Електронний ресурс]. Режим доступу: https://www.nature.com/articles/d41586-021-03552-w
23. New Botswana variant with 32 'horrific' mutations is the most evolved Covid strain EVER [Електронний ресурс]. Режим доступу: https://www.dailymail.co.uk/news/article-10238113/New-Botswana-variant-32-horrific-mutations-evolved-Covid-strain-EVER.html

24. Emergence in Southern France of a new SARS-CoV-2 variant. [Електронний ресурс]. Режим доступу: https://www.medrxiv.org/content/10.1101/2021.12.24.21268174v1
25. About JavaScript [Електронний ресурс]. Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript
26. How Javascript Event Loop Works? [Електронний ресурс]. Режим доступу: https://www.techboxweb.com/javasript-event-loop/
27. React Virtual DOM [Електронний ресурс]. Режим доступу: https://programmingwithmosh.com/react/react-virtual-dom-explained/
28. Axios Interceptors [Електронний ресурс]. Режим доступу: https://blog.clairvoyantsoft.com/intercepting-requests-responses-using-axios-df498b6cab62
29. Storybook for vite [Електронний ресурс]. Режим доступу: https://storybook.js.org/blog/storybook-for-vite/
30. MongoDB. Getting started [Електронний ресурс]. Режим доступу: https://www.mongodb.com/blog/post/getting-started-with-python-and-mongodb
31. What is git? [Електронний ресурс]. Режим доступу: https://www.nobledesktop.com/blog/what-is-git-and-why-should-you-use-it
32. Continuous deployment for nodejs [Електронний ресурс]. Режим доступу: https://dev.to/kevinc/continuous-deployment-for-nodejs-projects-using-vercel-3bhd
33. Особливості діагностики та лікування COVID-19 [Електронний ресурс]. Режим доступу: https://nuft.edu.ua/news/podiyi/z-osoblivostyami-diagnostiki-ta-likuvannya-covid-19-znajomly
34. Optimize Your Ventilation System design with CFD[Електронний ресурс]. Режим доступу: https://www.simscale.com/blog/2017/07/optimizing-ventilation-system-design/
35. Ventilation Rate [Електронний ресурс]. Режим доступу: https://www.sciencedirect.com/topics/engineering/ventilation-rate

ДОДАТКИ

Додаток А

Лістинг файлу package.json front-end частини програми

```
{
  "name": "app",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@ant-design/icons": "^4.7.0",
    "@craco/craco": "^6.4.3",
    "@inlet/react-pixi": "^6.6.5",
    "@testing-library/jest-dom": "^5.11.4",
    "@testing-library/react": "^11.1.0",
    "@testing-library/user-event": "^12.1.10",
    "@vitejs/plugin-react": "^1.3.1",
    "antd": "^4.18.5",
    "axios": "^0.24.0",
    "color-scales": "^3.0.2",
    "craco-less": "^2.0.0",
    "font-proxima-nova": "^1.0.1",
    "howler": "2.2.1",
    "i18next": "^21.5.3",
    "pixi-stats": "^1.0.7",
    "pixi.js": "^6.2.2",
    "prop-types": "^15.7.2",
    "react": "^17.0.2",
    "react-dom": "^17.0.2",
    "react-i18next": "^11.14.3",
    "react-icons": "^4.3.1",
    "react-query": "^3.33.5",
    "react-router-dom": "^6.3.0",
```

```
  "react-swipeable-views": "^0.14.0",
  "react-tiny-popover": "^7.0.1",
  "react-transition-group": "^4.4.2",
  "styled-components": "^5.3.3",
  "uuid": "^8.3.2",
  "vite": "^2.9.5",
  "web-vitals": "^1.0.1"
},
"devDependencies": {
  "babel-eslint": "10.1.0",
  "eslint": "^7.26.0",
  "eslint-config-airbnb": "^18.2.1",
  "eslint-config-prettier": "^8.3.0",
  "eslint-config-react-app": "^6.0.0",
  "eslint-import-resolver-alias": "^1.1.2",
  "eslint-plugin-flowtype": "8.0.3",
  "eslint-plugin-import": "^2.23.4",
  "eslint-plugin-jsx-a11y": "^6.4.1",
  "eslint-plugin-only-warn": "^1.0.3",
  "eslint-plugin-prettier": "^3.4.0",
  "eslint-plugin-react": "7.29.4",
  "eslint-plugin-react-hooks": "4.4.0",
  "eslint-plugin-simple-import-sort": "^7.0.0",
  "less": "^4.1.2",
  "prettier": "^2.3.0"
},
"scripts": {
  "start": "vite",
  "build": "vite build",
  "serve": "vite preview"
},
"eslintConfig": {
  "extends": [
    "react-app",
    "react-app/jest"
  ]
},
```

```
  "resolutions": {
    "react-error-overlay": "6.0.9"
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

Додаток Б

Лістинг файлу eslint.rc для конфігурування статистичого аналізу коду

```
const path = require('path');

module.exports = {
  extends: [
    'react-app',
    'airbnb',
    'plugin:jsx-a11y/recommended',
    'plugin:prettier/recommended',
```

```
  ],
  plugins: ['only-warn', 'react', 'jsx-a11y', 'simple-import-sort'],
  rules: {
    'import/prefer-default-export': 'off',
    'react/jsx-props-no-spreading': 'off',
    'react/jsx-uses-react': 'off',
    'react/react-in-jsx-scope': 'off',
    'react/require-default-props': 'off',
    'react/prop-types': 0,
    'simple-import-sort/imports': 'warn',
    'simple-import-sort/exports': 'warn',
    'prettier/prettier': [
      'warn',
      {
        singleQuote: true,
        trailingComma: 'all',
      },
    ],
    'arrow-body-style': ['warn', 'as-needed'],
  },
  overrides: [
    {
      files: ['*'],
      rules: {
        'simple-import-sort/imports': [
          'warn',
          {
            groups: [
              [
'^(assert|buffer|child_process|cluster|console|constants|crypto|dgram|dns|domain|events|fs|http|https|module|net|os|path|punycode|querystring|readline|repl|stream|string_decoder|sys|timers|tls|tty|url|util|vm|zlib|freelist|v8|process|async_hooks|http2|perf_hooks)(/.*|$)',
              ],
              ['^', '^@'],
              ['^(@src)(/.*|$)'],
              ['^\\u0000'],
              ['^\\.\\.(?!/?$)', '^\\.\\./?$'],
```

```
            ['^\\./(?=.*/)(?!/?$)', '^\\.(?!/?$)', '^\\../?$', '.styled$'],
            ['^.+\\.s?css$'],
          ],
        },
      ],
    },
  },
 ],
 settings: {
  'import/resolver': {
    alias: {
      map: [['@src', path.resolve(__dirname, 'src')]],
      extensions: ['.js', '.jsx', '.json'],
    },
  },
 },
};
```

Додаток В

Лістинг коду компонента React-роутера зі сторінками сайту

```
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import Layout from '../components/Layout';
import Browse from '../pages/Browse';
import Home from '../pages/Home';
import NoMatch from '../pages/NoMatch';
import Workspace from '../pages/Workspace';
import * as paths from './paths';
```

```
const SiteRoutes = () => (
  <Router>
    <Layout>
      <Routes>
        <Route path={`${paths.WORKSPACE}/:id`} element={<Workspace />} />
        <Route path={`${paths.WORKSPACE}`} element={<Workspace />} />
        <Route path={paths.HOME} element={<Home />} />
        <Route path={paths.BROWSE} element={<Browse />} />
        <Route path={'/*'} element={<NoMatch />} />
        <Route path={paths.NO_MATCH} element={<NoMatch />} />
      </Routes>
    </Layout>
  </Router>
);

export default SiteRoutes;
```

Додаток Г

Лістинг кастомного хуку UseData

```
import axios from 'axios';

const url = `-`;

const useData = () => {
  const getAllMaps = async () =>
    axios
      .get(`${url}/maps`)
      .then((res) => res.data)
      .catch((err) => err);
```

```
  const getMap = async (id) =>
    axios
      .get(`${url}/map`, { params: { id: id.toString() } })
      .then((res) => res.data)
      .catch((err) => err);

  const addMap = async (map) =>
    axios
      .post(`${url}/map`, { params: { map } })
      .then((res) => res)
      .catch((err) => err);

  const updateMap = async (map) =>
    axios
      .put(`${url}/map`, { params: { map } })
      .then((res) => res.data)
      .catch((err) => err);

  return { getAllMaps, getMap, updateMap, addMap };
};

export default useData;
```

Додаток Ґ

Лістинг класу “Fluid”

```
class Fluid {
  constructor(N, visc, diff, boundObjects) {
    this.N = N;
    this.visc = visc || 0;
    this.diff = diff || 0;
    this.clear();
```

```
  this.boundObjects = boundObjects;
}

updateBoundObjects(boundObjects) {
  this.boundObjects = boundObjects;
}

clear() {
  const { N } = this;
  this.u = new Float32Array((N + 2) * (N + 2));
  this.u0 = new Float32Array((N + 2) * (N + 2));
  this.v = new Float32Array((N + 2) * (N + 2));
  this.v0 = new Float32Array((N + 2) * (N + 2));
  this.x = new Float32Array((N + 2) * (N + 2));
  this.x0 = new Float32Array((N + 2) * (N + 2));
}

step(dt) {
  this.velStep(this.N, this.u, this.v, this.u0, this.v0, this.visc, dt);
  this.densStep(this.N, this.x, this.x0, this.u, this.v, this.diff, dt);
  this.u0.fill(0);
  this.v0.fill(0);
  this.x0.fill(0);
}

addDensity(x, y, dd) {
  this.x0[IX(this.N, x, y)] += dd;
}

addForce(x, y, fx, fy) {
  this.u0[IX(this.N, x, y)] += fx;
  this.v0[IX(this.N, x, y)] += fy;
}

addDensityIX(ix, dd) {
  this.x0[ix] += dd;
}
```

```
addForceIX(ix, fx, fy) {
  this.u0[ix] += fx;
  this.v0[ix] += fy;
}

densityAt(x, y) {
  return this.x[IX(this.N, x, y)];
}

velocityAt(x, y) {
  const ix = IX(this.N, x, y);
  return [this.u[ix], this.v[ix]];
}

velStep(N, u, v, u0, v0, visc, dt) {
  this.addSource(N, u, u0, dt);
  this.addSource(N, v, v0, dt);
  this.diffuse(N, 1, u0, u, visc, dt);
  this.diffuse(N, 2, v0, v, visc, dt);
  this.project(N, u0, v0, u, v);
  this.advect(N, 1, u, u0, u0, v0, dt);
  this.advect(N, 2, v, v0, u0, v0, dt);
  this.project(N, u, v, u0, v0);
}

densStep(N, x, x0, u, v, diff, dt) {
  this.addSource(N, x, x0, dt);
  this.diffuse(N, 0, x0, x, diff, dt);
  this.advect(N, 0, x, x0, u, v, dt);
}

project(N, u, v, p, div) {
  for (let i = 1; i <= N; i += 1) {
    for (let j = 1; j <= N; j += 1) {
      // eslint-disable-next-line no-param-reassign
      div[IX(N, i, j)] =
```

```
        (-0.5 *
          (u[IX(N, i + 1, j)] -
            u[IX(N, i - 1, j)] +
            v[IX(N, i, j + 1)] -
            v[IX(N, i, j - 1)])) /
          N;
        // eslint-disable-next-line no-param-reassign
        p[IX(N, i, j)] = 0;
      }
    }
    this.setBnd(N, 0, div);
    this.setBnd(N, 0, p);

    this.linSolve(N, 0, p, div, 1, 4);

    for (let i = 1; i <= N; i += 1) {
      for (let j = 1; j <= N; j += 1) {
        // eslint-disable-next-line no-param-reassign
        u[IX(N, i, j)] -= 0.5 * N * (p[IX(N, i + 1, j)] - p[IX(N, i - 1, j)]);
        // eslint-disable-next-line no-param-reassign
        v[IX(N, i, j)] -= 0.5 * N * (p[IX(N, i, j + 1)] - p[IX(N, i, j - 1)]);
      }
    }
    this.setBnd(N, 1, u);
    this.setBnd(N, 2, v);
  }

  advect(N, b, d, d0, u, v, dt) {
    let i0;
    let j0;
    let i1;
    let j1;
    let x;
    let y;
    let s0;
    let t0;
    let s1;
```

```
  let t1;

  const dt0 = dt * N;
  for (let i = 1; i <= N; i += 1) {
    for (let j = 1; j <= N; j += 1) {
      x = i - dt0 * u[IX(N, i, j)];
      y = j - dt0 * v[IX(N, i, j)];
      if (x < 0.5) x = 0.5;
      if (x > N + 0.5) x = N + 0.5;
      // eslint-disable-next-line no-bitwise
      i0 = x | 0;
      i1 = i0 + 1;
      if (y < 0.5) y = 0.5;
      if (y > N + 0.5) y = N + 0.5;
      // eslint-disable-next-line no-bitwise
      j0 = y | 0;
      j1 = j0 + 1;
      s1 = x - i0;
      s0 = 1 - s1;
      t1 = y - j0;
      t0 = 1 - t1;
      // eslint-disable-next-line no-param-reassign
      d[IX(N, i, j)] =
        s0 * (t0 * d0[IX(N, i0, j0)] + t1 * d0[IX(N, i0, j1)]) +
        s1 * (t0 * d0[IX(N, i1, j0)] + t1 * d0[IX(N, i1, j1)]);
    }
  }

  this.setBnd(N, b, d);
}

diffuse(N, b, x, x0, diff, dt) {
  const a = dt * diff * N * N;
  this.linSolve(N, b, x, x0, a, 1 + 4 * a);
}

linSolve(N, b, x, x0, a, c) {
```

```
  for (let k = 0; k < 30; k += 1) {
    for (let i = 1; i <= N; i += 1) {
      for (let j = 1; j <= N; j += 1) {
        // eslint-disable-next-line no-param-reassign
        x[IX(N, i, j)] =
          (x0[IX(N, i, j)] +
            a *
              (x[IX(N, i - 1, j)] +
                x[IX(N, i + 1, j)] +
                x[IX(N, i, j - 1)] +
                x[IX(N, i, j + 1)])) /
          c;
      }
    }
    this.setBnd(N, b, x);
  }
}

addSource(N, x, s, dt) {
  const size = N * N;
  // eslint-disable-next-line no-param-reassign
  for (let i = 0; i < size; i += 1) x[i] += dt * s[i];
}

setBnd(N, b, x) {
  for (let i = 1; i <= N; i += 1) {
    for (let j = 1; j <= N; j += 1) {
      if (this.boundObjects[IX(N, i, j)]) {
        // x[IX(N,i,j)] = (((x[IX(N,i - 1, j)] + x[IX(N,i + 1,j)] + x[IX(N,i, j - 1)] + x[IX(N,i,j + 1)])) / 4);
        x[IX(N, i, j)] = x[IX(N, i - 1, j)];
        x[IX(N, i, j)] -= x[IX(N, i, j - 1)];
        x[IX(N, i, j)] += x[IX(N, i + 1, j)];
        x[IX(N, i, j)] -= x[IX(N, i + 1, j + 1)];

        x[IX(N, i, j)] *= 0;
      }
    }
```

```
    }

    this.boundaryBoxAbsorb(N, b, x);
    this.boundaryBoxCorners(N, b, x);
  }

  boundaryBoxAbsorb(N, b, x) {
    for (let i = 1; i <= N; i += 1) {
      x[IX(N, 0, i)] = 0;
      x[IX(N, N + 1, i)] = 0;
      x[IX(N, i, 0)] = 0;
      x[IX(N, i, N + 1)] = 0;
    }
  }

  boundaryBoxReflect(N, b, x) {
    for (let i = 1; i <= N; i += 1) {
      x[IX(N, 0, i)] = b === 1 ? -x[IX(N, 1, i)] : x[IX(N, 1, i)];
      x[IX(N, N + 1, i)] = b === 1 ? -x[IX(N, N, i)] : x[IX(N, N, i)];
      x[IX(N, i, 0)] = b === 2 ? -x[IX(N, i, 1)] : x[IX(N, i, 1)];
      x[IX(N, i, N + 1)] = b === 2 ? -x[IX(N, i, N)] : x[IX(N, i, N)];
    }
  }

  boundaryBoxCorners(N, b, x) {
    x[IX(N, 0, 0)] = 0.5 * (x[IX(N, 1, 0)] + x[IX(N, 0, 1)]);
    x[IX(N, 0, N + 1)] = 0.5 * (x[IX(N, 1, N + 1)] + x[IX(N, 0, N)]);
    x[IX(N, N + 1, 0)] = 0.5 * (x[IX(N, N, 0)] + x[IX(N, N + 1, 1)]);
    x[IX(N, N + 1, N + 1)] = 0.5 * (x[IX(N, N, N + 1)] + x[IX(N, N + 1, N)]);
  }
}
```

Додаток Д

Лістинг кастомного хуку “useStemFluid”

```
const useStemFluid = ({ resolution, viscosity, diffuse, stemBound }) => {
  const stemBoundRef = useRef(stemBound);
  const [fluid, setFluid] = useState(
    new Fluid(resolution, viscosity, diffuse, stemBound),
  );
  const updateBoundObjects = (newBound) => {
    stemBoundRef.current = newBound;
    fluid.updateBoundObjects(newBound);
  };

  const clearDensity = () => {
    setFluid(new Fluid(resolution, viscosity, diffuse, stemBoundRef.current));
  };
  return {
    fluid,
    stemBoundRef,
    clearDensity,
    resolution,
    updateBoundObjects,
  };
};

export { IX, useStemFluid };
```

Додаток Е

Лістинг компоненту NewPage.jsx

```
import { message } from 'antd';
import { useState } from 'react';
import { useNavigate } from 'react-router-dom';
```

```
import useData from '../../../../hooks/UseData';
import { WORKSPACE } from '../../../../routes/paths';

import * as S from './NewMap.styled';

const NewMap = () => {
  const [name, setName] = useState('');
  const [loading, setLoading] = useState(false);
  const { addMap } = useData();
  const navigate = useNavigate();
  const onNameChange = (e) => {
    setName(e.target.value);
  };

  const onCreate = () => {
    if (name.length < 3 || name.length > 20) {
      message.error('Name`s length must >= 3 and <= 20 symbols');
      return;
    }
    setLoading(true);
    addMap({ resolution: 64, viscosity: 0, diffuse: 0, name })
      .then((res) => {
        if (res.data.response.id) {
          navigate(`${WORKSPACE}/${res.data.response.id}`);
          message.success('Map has been created');
        }
      })
      .catch(() => {
        message.error('Error creating map');
      });
  };

  return (
    <S.Wrapper>
      <S.Container>
        <S.Title>CREATE NEW MAP</S.Title>
```

```
        <S.InputContainer>
          <S.InputName
            placeholder="name"
            value={name}
            onChange={onNameChange}
          />
          <S.CreateButton
            disabled={loading}
            type="default"
            block
            onClick={onCreate}
          >
            CREATE
          </S.CreateButton>
        </S.InputContainer>
      </S.Container>
    </S.Wrapper>
  );
};

NewMap.propTypes = {};

export default NewMap;
```

Додаток Є

Лістинг класу “ICFDMAP”

```
import { useStemFluid } from '../pages/Workspace/components/StemFluid/StemFluid';

import { AIR_SOURCE_CODE, FAN_CODE, NONE_CODE, WENT_CODE } from './ObjectCodes';

export class ICFDMAP {
  constructor({ objects, resolution, viscosity, diffuse, id, name }) {
```

```
  this.resolution = resolution;
  this.viscosity = viscosity;
  this.diffuse = diffuse;
  this.id = id;
  this.name = name;

  if (!objects) {
    this.generateClearMap();
  } else {
    this.objects = objects;
    this.updateStemBoundRef();
  }
}

updateViscosity(value) {
  this.viscosity = value;
  this.stemFluid.fluid.visc = value;
}

updateDiffuse(value) {
  this.diffuse = value;
  this.stemFluid.fluid.diff = value;
}

objectsToStemBound() {
  return this.objects.map(
    ({ code }) =>
      code !== 0 &&
      code !== WENT_CODE &&
      code !== AIR_SOURCE_CODE &&
      code !== FAN_CODE,
  );
}

initStemFluid() {
  // eslint-disable-next-line react-hooks/rules-of-hooks
  this.stemFluid = useStemFluid({
```

```
      resolution: this.resolution,
      stemBound: this.objectsToStemBound(),
      viscosity: this.viscosity,
      diffuse: this.diffuse,
    });
    this.updateStemBoundRef();
  }

  addObject(code, x, y, orientation) {
    this.objects[this.IX(x, y)].code = code;
    if (orientation) this.objects[this.IX(x, y)].orientation = orientation;
    this.updateStemBoundRef();
  }

  removeObject(x, y) {
    this.objects[this.IX(x, y)].code = NONE_CODE;
    this.updateStemBoundRef();
  }

  updateStemBoundRef() {
    if (this.stemFluid) {
      this.stemFluid.updateBoundObjects(this.objectsToStemBound());
    }
  }

  generateClearMap() {
    this.objects = new Array(
      (this.resolution + 2) * (this.resolution + 2),
    ).fill(0);
    this.objects = this.objects.map(() => ({ code: 0 }));
    this.updateStemBoundRef();
  }

  IX(i, j) {
    return i + (this.resolution + 2) * j;
  }
}
```

Додаток Ж

Лістинг компоненту Canvas.jsx

```
import { useContext, useEffect, useRef } from 'react';
import { Graphics, Stage } from '@inlet/react-pixi';

import ResponsibleSizeContext from '../../../../contexts/ResponsibleSize';

import { useCanvas } from './hooks/useCanvas';
import * as S from './Canvas.styled';

const Canvas = ({ workspace, map }) => {
  const stageRef = useRef();
  const containerRef = useRef();

  const { toolbar } = workspace;

  const { canvasWidth, canvasHeight, setCanvasWidth, setCanvasHeight } =
    useContext(ResponsibleSizeContext);

  const { startSceneLooping, stopSceneLooping, handleControls } = useCanvas({
    canvasWidth,
    canvasHeight,
    toolbar,
    stageRef,
    MAP: map,
  });

  const handleResize = () => {
    setCanvasWidth(containerRef?.current?.offsetWidth);
    setCanvasHeight(containerRef?.current?.offsetWidth);
  };
```

```
  useEffect(() => {
    window.addEventListener('resize', handleResize);
    handleResize();

    return () => {
      stopSceneLooping();
    };

    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, []);

  useEffect(() => {
    stageRef.current?.app.renderer.resize(canvasWidth, canvasHeight);
  }, [canvasHeight, canvasWidth]);

  return (
    <S.Wrapper>
      <S.Container ref={containerRef}>
        <Stage
          id="canvas-video"
          ref={stageRef}
          onMouseDown={handleControls}
          onMouseUp={handleControls}
          onMouseMove={handleControls}
          onTouchEnd={handleControls}
          onTouchMove={handleControls}
          onTouchStart={handleControls}
          onContextMenu={handleControls}
        >
          <Graphics tint={0xffffff} draw={startSceneLooping} />
        </Stage>
      </S.Container>
    </S.Wrapper>
  );
};
```

```
Canvas.propTypes = {};

export default Canvas;
```

Додаток З

Лістинг модулю requests.js

```
const { MongoClient } = require('mongodb');
require('dotenv-flow').config();

const username = process.env.DB_USERNAME;
const password = process.env.DB_PASSWORD;

const url = `mongodb+srv://${username}:${password}@icfd.lwisl.mongodb.net/?retryWrites=true&w=majority`;
const client = new MongoClient(url);

const dbName = 'ICFD';
const collectionName = 'MAPS';

async function connectCollection() {
  await client.connect();
  const db = client.db(dbName);
  const collection = db.collection(collectionName);

  return { collection };
}

async function getAllMaps() {
  const { collection } = await connectCollection();
  return collection.find({}).toArray();
}

async function getMap(findId) {
  const { collection } = await connectCollection();
  return collection.findOne({ id: findId });
```

```
}

async function updateMap(map) {
  const { collection } = await connectCollection();
  return collection.updateOne(
    { id: map.id },
    {
      $set: {
        objects: map.objects,
        resolution: map.resolution,
        viscosity: map.viscosity,
        diffuse: map.diffuse,
      },
    },
  );
}

async function addMap(map) {
  const { collection } = await connectCollection();
  return collection.insertOne(map);
}

module.exports = { getAllMaps, getMap, addMap, updateMap };
```