МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ВЕБІНТЕРФЕЙСУ СИСТЕМИ СКІНЧЕННО-ЕЛЕМЕНТНОГО АНАЛІЗУ ІЗ ВИКОРИСТАННЯМ БІБЛІОТЕКИ REACT»**

Виконав: студент 2 курсу, групи 8.1214-з
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
А.В. Трипак
(ініціали та прізвище)

Керівник декан математичного факультету, професор, д.т.н. Гоменюк С.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

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

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Трипаку Андрію Васильовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебінтерфейсу системи скінченно-елементного аналізу із використанням бібліотеки React

керівник роботи Гоменюк Сергій Іванович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 28 » лютого 2025 року № 327-с

2. Строк подання студентом роботи 28.11.2025 р.

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі, аналіз предметної області.
2. Проєктування.
3. Реалізація та тестування.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

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

7. Дата видачі завдання 14.03.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	28.03.2025	
2.	Збір вихідних даних.	18.04.2025	
3.	Обробка методичних та теоретичних	16.06.2025	
	джерел.		
4.	Розробка першого та другого розділу.	29.08.2025	
5.	Розробка третього розділу.	17.11.2025	
6.	Оформлення та нормоконтроль	24.11.2025	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	12.12.2025	

Студент _____ А.В. Трипак
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.І. Гоменюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка вебінтерфейсу системи скінченно-елементного аналізу із використанням бібліотеки React»: 68 с., 31 рис., 1 табл., 20 джерел, 4 додатка.

FASTAPI, FEA, JSON, NUMPY, PYTHON, REACT, REST API, TYPESCRIPT, UML, VISUALIZATION, VITE.

Об’єкт дослідження – вебінтерфейси систем скінченно-елементного аналізу.

Мета роботи – розробити легкий клієнт-серверний вебінтерфейс CEA з використанням React для задання параметрів моделі, запуску розрахунку та перегляду результатів у браузері.

Методи дослідження – аналітичний, моделювання, проєктування, програмний, обчислювальний експеримент.

У роботі розглянуто проєктування та реалізацію клієнт-серверної CEA-системи для 2D стержневих моделей. Клієнтська частина виконана засобами React із трьома способами введення: форма, JSON-редактор, імпорт JSON-файлу. Серверна частина реалізована на FastAPI з обчислювальним модулем на NumPy. Підтримуються ендпоінти application/json і multipart/form-data, валідація Pydantic і повідомлення про помилки.

Побудовано UML-діаграми: варіантів використання, діяльності, послідовності та розгортання. Інтерфейс має просту і логічну структуру «введення –> розрахунок –> результати», з табличним поданням переміщень/реакцій/зусиль та керуванням шарами відображення.

Таким чином, розроблений вебінтерфейс забезпечує зрозумілу роботу з простими CEA-моделями без інсталяції та авторизації, швидко виконує розрахунки, легко розширюється та підтримується завдяки мінімалістичній архітектурі й чітким API-контрактам.

SUMMARY

Master’s qualifying paper “Development of the Web Interface of the Finite Element Analysis System Using the React Library”: 68 pages, 31 figures, 1 table, 20 references, 4 supplements.

FASTAPI, FEA, JSON, NUMPY, PYTHON, REACT, REST API, TYPESCRIPT, UML, VISUALIZATION, VITE.

The object of the study is web interfaces of finite element analysis systems.

The aim of the study is to develop a lightweight client-server web interface for FEA using React to set model parameters, start calculations, and view results in a browser.

The methods of research are analytical, modeling, design, software, computational experiment.

The paper considers the the design and implementation of a client-server FEA system for 2D beam models. The client part is implemented using React with three input methods: form, JSON editor, and JSON file import. The server part is implemented on FastAPI with a computational module on NumPy. Application/json and multipart/form-data endpoints, Pydantic validation, and error messages are supported.

UML diagrams have been built: use cases, activities, sequences, and deployment. The interface has a simple and logical structure of «input -> calculation -> results» with a tabular representation of movements/reactions/forces and control of display layers.

Thus, the developed web interface provides clear operation with simple SEA models without installation and authorization, performs calculations quickly, and is easily expandable and maintainable thanks to its minimalist architecture and clear API contracts.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 8
1 Технічне завдання 10
1.1 Терміни та визначення 10
1.1.1 Загальні терміни 10
1.1.2 Технічні терміни 10
1.2 Функціональні вимоги 11
1.2.1 Призначення і цілі створення системи 11
1.2.2 Загальні функціональні можливості системи 11
1.3 Нефункціональні вимоги 12
1.4 Опис предметної області 12
1.5 Опис системи 13
1.6 Огляд та порівняння аналогів 14
2 Проєктування 16
2.1 Використання UML під час розробки системи 16
2.2 Діаграма варіантів використання 17
2.2.1 Опис варіантів використання 19
2.3 Діаграма діяльності 26
2.4 Діаграма послідовності 27
2.5 Діаграма розгортання 29
3 Програмна реалізація та тестування 32
3.1 Опис інструментів розробки 32
3.2 Основні компоненти та модулі системи 32
3.3 Встановлення та налаштування оточення 33
3.4 Опис API 36

3.5 Тестування проєкту 38
3.6 Керівництво користувача 40
3.6.1 Головна сторінка 40
3.6.2 Введення даних моделі через форму 40
3.6.3 Введення даних моделі через JSON-редактор 41
3.6.4 Імпорт моделі з файлу 42
3.6.5 Перегляд результатів 43
3.6.6 Візуалізація 44
Висновки 48
Перелік посилань 49
Додаток А Серверна частина 51
Додаток Б React fea.ts 58
Додаток В React Home.tsx 59
Додаток Г React Viewer2D 67

ВСТУП

З поширенням вебтехнологій зростає потреба в доступних та простих інструментах інженерного аналізу, які працюють без складного процесу встановлення та ліцензування. Багато готових продуктів на ринку пропонують великий функціонал, але є важкими для швидкого навчального або прототипувального використання. Також обмеженням є те, що хмарні рішення часто комерційні, а локальні інсталятори потребують ресурсів і адміністрування. В цих умовах актуально створити легкий вебінтерфейс для базових СЕА-сценаріїв із чітким інтерфейсом, мінімальним сервером і прозорим обміном даними.

Актуальність роботи полягає у побудові відкритого клієнт-серверного рішення, де користувач у браузері задає параметри, запускає розрахунок і одразу отримує результати у вигляді таблиць та деформованої схеми. Підхід поєднує простоту інтерфейсу зі стандартними інженерними практиками.

З огляду на це, сформульовано цілі та задачі дослідження.

Мета: розробити легкий клієнт-серверний вебінтерфейс СЕА з використанням React для задання параметрів моделі, запуску розрахунку та перегляду результатів у браузері.

Завдання:

– сформулювати функціональні й нефункціональні вимоги до системи;
– проаналізувати, описати та порівняти десктопні/хмарні СЕА-інструменти;
– спроєктувати клієнт-серверну архітектуру та моделі даних;
– реалізувати клієнтську і серверну частину;
– забезпечити валідацію та обробку помилок;
– протестувати ключові сценарії та надати керівництво користувача.

Об’єкт дослідження: вебінтерфейси систем скінченно-елементного аналізу.

Предмет дослідження: React, FastAPI, NumPy.

Методи дослідження: аналітичний, моделювання, проєктування, програмний, обчислювальний експеримент.

У першому розділі надано технічне завдання, функціональні/нефункціональні вимоги, подано аналіз предметної області та огляд аналогів із таблицею порівняння.

У другому розділі надано обґрунтування щодо доцільності використання UML, і наведено діаграми варіантів використання, діяльності, послідовності та розгортання.

У третьому розділі описано застосовані інструменти розробки, встановлення та налаштування середовища, надано детальний опис та структуру API, приклади unit-тестування та детальне керівництво користувача.

1 ТЕХНІЧНЕ ЗАВДАННЯ

1.1 Терміни та визначення

1.1.1 Загальні терміни

Система – вебзастосунок для базових задач скінченно-елементного аналізу з інтерфейсом на React.

React – JS-бібліотека для розробки вебзастосунків [11].

Fluent UI – набір UI-компонентів для React.

FastAPI – фреймворк для створення високопродуктивних REST API на Python [4].

СЕА – скінченно-елементний аналіз.

ДВВ – діаграма варіантів використання.

ДД – діаграма діяльності.

ДП – діаграма послідовності.

ДР – діаграма розгортання.

Користувач – суб’єкт, що взаємодіє з системою.

1.1.2 Технічні терміни

Вузол – точка дискретизації конструкції з двома ступенями свободи.

Елемент – стержень, що з’єднує два вузли та працює на розтяг/стиск.

Матриця жорсткості – матриця, що пов’язує вектори переміщень і сил.

Граничні умови – закріплення вузлів та вузлові навантаження.

JSON – текстовий формат обміну даними для опису моделі та результатів.

REST API – інтерфейс взаємодії клієнта і сервера через HTTP.

CORS – політика спільного використання ресурсів між різними джерелами.

SVG – векторна графіка.

SPA-застосунок – застосунок, що працює в межах однієї сторінки з динамічним оновленням контенту.

1.2 Функціональні вимоги

1.2.1 Призначення і цілі створення системи

Функціональне призначення системи – забезпечити користувача простим та зрозумілим вебінтерфейсом для постановки і розв’язання базових задач СЕА на прикладі 2D стержневих конструкцій з подальшою візуалізацією деформованої форми та відображенням числових результатів.

Експлуатаційне призначення – система експлуатується користувачами для розв’язання базових задач СЕА.

Мета створення системи – реалізувати мінімалістичний, зрозумілий і масштабований інтерфейс на React для взаємодії з СЕА-сервісами.

1.2.2 Загальні функціональні можливості системи

Система має надавати користувачам наступні можливості:

– введення параметрів моделі через форму;
– введення параметрів моделі через JSON-редактор;
– імпорт параметрів моделі з JSON-файлу;
– отримати приклад параметрів;
– запустити розрахунок;
– отримати результат у табличному вигляді;
– отримати візуалізацію;
– увімкнути/вимкнути шар візуалізації;
– введення вузлів/елементів/опор/навантаження/масштабу деформації.

1.3 Нефункціональні вимоги

Інтерфейс користувача. Адаптивне відображення однією колонкою за принципом mobile-first: панелі розміщені вертикальним стеком без внутрішніх скролів. Візуалізація у вигляді респонсивного SVG.

Підтримка браузерів. Актуальні версії наступних браузерів: Chrome, Edge, Opera, Firefox, Safari.

Вимоги до продуктивності. Початкове завантаження SPA не більше 1 секунди на сучасних пристроях за умови стабільного мережевого з'єднання. Розрахунок невеликих моделей на десятки вузлів/елементів < 100 мс.

Вимоги до безпеки та приватності. Інтерфейс взаємодіє з сервером лише через REST API з валідацією вхідних даних. Контроль доступу з клієнту регулюється засобами CORS. Сервер повинен мати обмеження на розмір завантажених файлів, не більше 25 Мб. Система повинна відображати повідомлення про помилки без відображення всього стеку.

1.4 Опис предметної області

Предметною областю є скінченно-елементний аналіз плоских стержневих конструкцій.

Предметна область охоплює плоскі стержневі системи, що працюють переважно на згин, стиск/розтяг і зсув у своїй площині. Такі конструкції є об'єктом аналізу для оцінювання їхньої жорсткості, міцності та експлуатаційної надійності.

В основі методу аналізу лежить дискретизація суцільної конструкції на кінцеву кількість менших, простих елементів (скінченних елементів), з'єднаних у вузлах. Для кожного елемента формулюються рівняння рівноваги, які потім об'єднуються в глобальну систему рівнянь для всієї конструкції.

Предметна область охоплює моделювання й розрахунок таких фізичних явищ, як статична поведінка (визначення переміщень, деформацій, внутрішніх

сил), динамічна поведінка (аналіз власних частот і форм коливань, відгук на динамічні навантаження) і стійкість (визначення критичних навантажень, за яких конструкція втрачає початкову форму рівноваги).

Вихідні дані задаються наступним чином: користувач задає дискретну модель через координати вузлів, зв’язки елементів, параметри матеріалу та перерізу, граничні умови і вузлові навантаження.

Кінцевою метою аналізу є отримання кількісних даних про роботу конструкції, які використовуються для її проєктування, перевірки несучої здатності та оптимізації. За результатами обчислень повертаються переміщення вузлів, реакції в опорах, осьові сили в елементах і координати деформованої схеми для відтворення у візуалізаторі.

Дана предметна область має наступні етапи:

– побудова геометричної моделі;
– визначення властивостей;
– прикладання навантажень;
– дискретизація;
– розв’язання системи рівнянь;
– обробка результатів.

1.5 Опис системи

Система має клієнт-серверну архітектуру. Розглянемо її складові частини. UI представляє собою SPA-застосунок, створенний засобами React та Fluent UI. Складається з наступних компонентів:

– панель для вибору методу введення даних (форма/редактор JSON/імпорт JSON-файлу);
– панель для відображення результатів розрахунку (таблиці переміщень, реакцій, осьових сил, метаінформації);
– панель для перегляду візуалізації результатів (SVG-перегляд оригінальної та деформованої форми з перемикачами шарів).

API-сервер реалізований засобами FastAPI та NumPy. Надає ендпоінти та відповідає за логіку розрахунків (формування глобальної матриці жорсткості, накладання граничних умов, розв’язання системи рівнянь, обчислення реакцій і осьових сил, формування деформованих координат).

Наданий поділ є традиційним для подібних систем, де клієнт відповідає за зручність введення й відображення, а сервер – за коректність розрахунків.

1.6 Огляд та порівняння аналогів

Так як CEA-системи відіграють важливу роль у багатьох галузях, то на ринку вже представлені подібні до нашої платформи. Розглянемо декілька з них.

SimScale – хмарна платформа CEA/CFD із широким колом аналізів, готовими навчальними матеріалами та спільнотою. Виділяється широкою функціональністю, але це робить її складнішою у використанні [13].

COMSOL Server – налає доступ до моделей COMSOL із браузера. Сильною стороною є велика кількість фізичних модулів та інтеграція з MATLAB. Натомість розгортання й ліцензування вимагають ресурсів [3].

ANSYS Cloud – хмарний доступ до продуктів ANSYS, але сама система орієнтована на великі промислові задачі та потребує екосистему ANSYS [2].

Autodesk Fusion 360 – інтегрована CAD/CAE-платформа з інструментами симуляції, але має ті же особливості, що і ANSYS Cloud, а саме зручна для користувачів Fusion 360 і сфокусована на екосистемі Autodesk [1].

Усі розглянуті аналоги здатні розв’язувати задачі CEA, але мають значний функціональний обсяг, вищу складність і потребують певного рівня технічних знань.

Наша система, на відміну від перелічених, має лише необхідні дії для 2D стержневих конструкції та простий вебінтерфейс. Це знижує поріг входу для користувачів, пришвидшує навчання і дозволяє легко розширювати функціонал.

У таблиці 1.1 наведено порівняння нашої системи і вищеописаних аналогів.

Таблиця 1.1 – Порівняння аналогів

Критерій / Платформа	**Наша система**	**SimScale**	**COMSOL Server**	**ANSYS Cloud**	**Fusion 360 (Simulation)**
2D стержнева задача	Так	Так	Так	Так	Так
Введення через форму/JSON/файл	Так	Частково	Частково	Частково	Частково
Візуалізація деформованої форми (браузер)	Так	Так	Так	Так	Так
Відкрите легке REST API	Так	–	–	–	–
Поріг входу інтерфейсу	Низький	Середній	Високий	Високий	Середній
Орієнтація	Навчання, демо, швидкі перевірки	Промислові / академічні задачі	Промислові / академічні	Промислові задачі	CAD / CAE користувачі
Інфраструктурні вимоги	Мінімальні	Хмара	Сервер / хмара	Хмара	Хмара / десктоп

Запропонована система хоч у має обмежену сферу застосування, заточену на базовий стержневий аналіз, але забезпечує простий та зрозумілий інтерфейс, що дозволяє скоротити час від постановки задачі до отримання результату і спростити супровід та розширення.

2 ПРОЄКТУВАННЯ

2.1 Використання UML під час розробки системи

UML – це стандартизована графічна мова моделювання, призначена для візуалізації, специфікації, побудови та документування об’єктно-орієнтованих програмних систем [17].

Використання UML під час розробки дозволяє створювати наочне представлення різних аспектів системи – її структури, поведінки, архітектури та взаємозв’язків між компонентами. Це сприяє кращому розумінню проєкту всіма учасниками, зокрема розробниками, аналітиками та зацікавленими сторонами [5, 6, 17].

Завдяки UML можна описувати вимоги до системи, моделювати функціональні можливості та бізнес-процеси, а також ефективно передавати ці вимоги від замовників до розробників. Такий підхід створює міцну основу для подальшого проєктування програмного забезпечення [6, 17].

UML надає структурований спосіб розробки, який дозволяє моделювати й візуалізувати різні частини системи на всіх етапах її життєвого циклу – від побудови класів і компонентів до опису розгортання та поведінки системи [5, 17].

Діаграми UML мають стандартизований вигляд і є зрозумілими для всіх учасників проєкту. Це сприяє ефективній комунікації, обміну знаннями та може використовуватися як частина офіційної документації, що полегшує подальше супроводження й розвиток системи [17].

Окрім того, існує велика кількість інструментів, які підтримують роботу з UML. Вони дозволяють створювати діаграми, виконувати зворотне проєктування коду та генерувати код із моделей, підвищуючи продуктивність розробки та забезпечуючи узгодженість між моделями і програмним кодом [5].

У даній роботі застосовано такі типи діаграм: діаграму варіантів використання для опису функціональних вимог і акторів, діаграму діяльності для

відображення алгоритмів і логічних гілок процесів, діаграму послідовності для моделювання часових взаємодій компонентів та діаграму розгортання для опису фізичного розміщення вузлів і артефактів.

2.2 Діаграма варіантів використання

Діаграма варіантів використання є одним із ключових типів діаграм у мові моделювання UML. Вона призначена для наочного відображення функціональних вимог системи та показує, як саме відбувається взаємодія між системою й зовнішніми учасниками – акторами, якими можуть бути користувачі або інші системи [18].

Основними складовими діаграми є актори, варіанти використання, асоціації, а також відношення включення та розширення [5, 18].

Актори – це зовнішні сутності, що взаємодіють із системою, наприклад користувачі, пристрої чи сторонні сервіси. Їх зазвичай зображують у вигляді фігурки людини або спеціального символу. Варіанти використання відображають сценарії або послідовності дій, які ініціює актор і які реалізують певну функціональність системи; вони позначаються овалами з назвами всередині.

Асоціації – це лінії, що поєднують акторів із варіантами використання, демонструючи можливі шляхи їхньої взаємодії. Відношення include застосовується, коли один варіант використання обов’язково включає функціональність іншого, тоді як extend показує можливість розширення або доповнення базового сценарію за певних умов.

Діаграма варіантів використання надає узагальнене уявлення про межі системи та її функціональні можливості. Вона допомагає краще зрозуміти вимоги, визначити ролі акторів і їхню взаємодію з системою, а також виявити потенційні прогалини або недоопрацьовані аспекти функціональності.

Цей тип діаграм є ефективним засобом комунікації між усіма учасниками проєкту – замовниками, аналітиками, дизайнерами й розробниками. Він сприяє

спільному розумінню та узгодженню функціональних вимог до майбутньої системи [18].

На рисунку 2.1 зображена ДВВ для вебінтерфейсу скінченно-елементного аналізу.

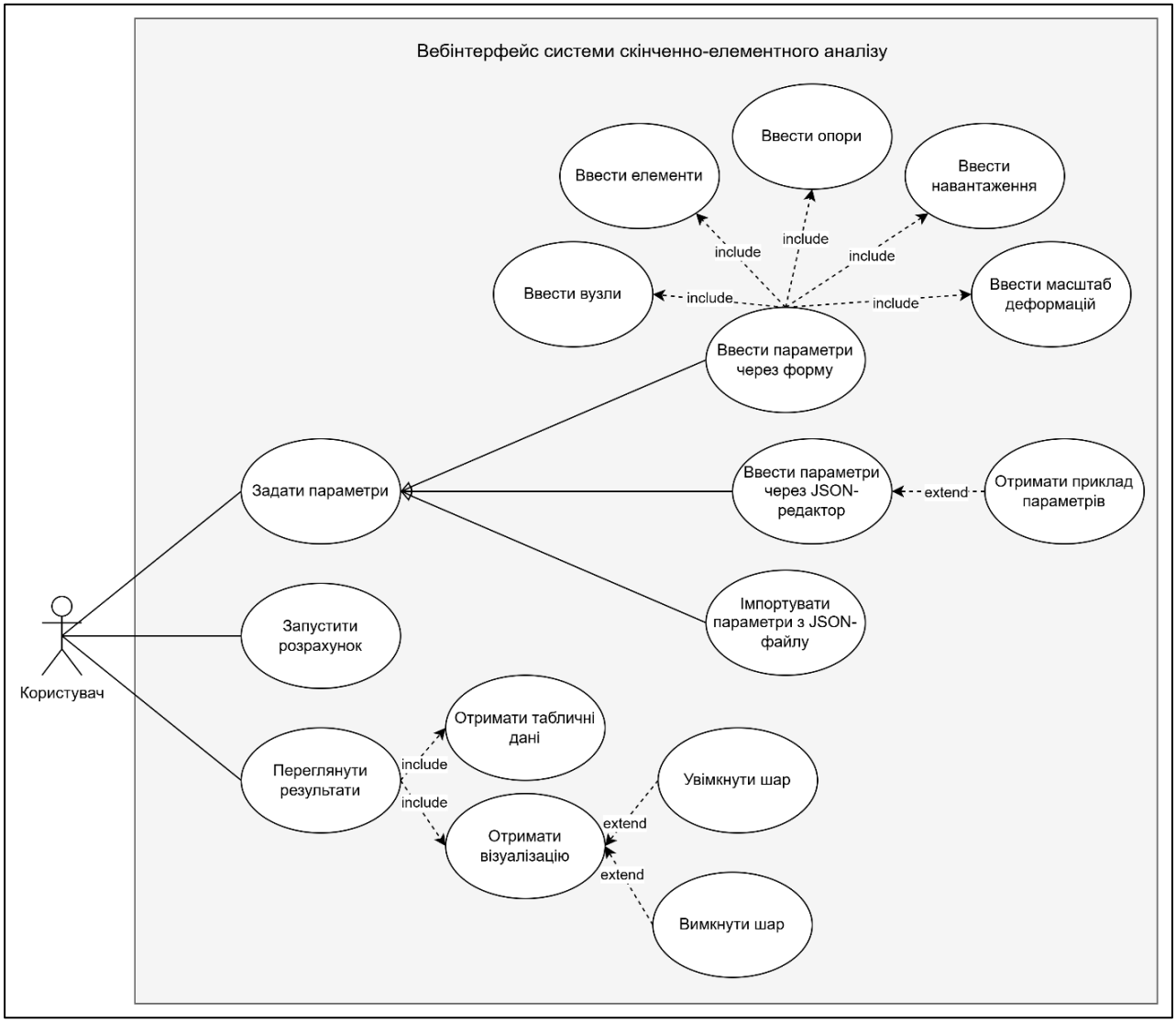


Рисунок 2.1 – Діаграма варіантів використання

Представлено актора «Користувач», який взаємодіє з системою через вебінтерфейс. Серед прецедентів можна виділити наступні ключові: «Задати параметри моделі», «Запустити розрахунок» і «Переглянути результати». Ці прецеденти забезпечують повний цикл бізнес-процесу системи.

Виділимо один із основних прецедентів – «Запустити розрахунок». Після натискання кнопки «Розрахувати», система виконує клієнтську валідацію

структури, індексів елементів, додатності A та E, достатності опор. Після валідації, система надсилає REST-запит до серверу, де виконується парсинг і серверна валідація. Обчислювальний модуль збирає глобальну матриці жорсткості, накладає граничні умови, розв’язує системи рівнянь, обчислює реакції і осьові сили. Після успішного розрахунку, система надсилає відповідь, що містить переміщення вузлів, реакції, осьові сили та допоміжні дані для побудови деформованих координат. У разі помилок користувач отримує відповідне повідомлення з підказками щодо виправлення або повтору спроби.

2.2.1 Опис варіантів використання

Прецедент «Задати параметри моделі».

Призначення: даний варіант використання надає можливість користувачу підготувати вхідні дані для розрахунку СЕА.

Передумови: користувач знаходиться на головній сторінці застосунку з панеллю введення.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач обирає спосіб задання параметрів: через форму, через JSON-редактор або імпорт із JSON-файлу. Система відображає відповідну секцію введення, перевіряє цілісність і базову коректність даних та фіксує готовність до розрахунку.

Виняткова ситуація 1: неповні або суперечливі дані – система підсвічує помилкові поля/рядки і блокує запуск розрахунку.

Виняткова ситуація 2: помилка парсингу/формату – система виводить відповідне повідомлення про помилку та пропонує виправити дані.

Прецедент «Ввести параметри через форму».

Призначення: даний варіант використання надає можливість користувачу ввести параметри моделі за допомогою інтерактивної форми.

Передумови: користувач знаходиться у вкладці «Форма» панелі введення.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач додає/змінює/видаляє вузли (x, y), елементи (n1, n2, A, E), задає опори (ux, uy), навантаження (Fx, Fy), встановлює масштаб деформацій. Система виконує валідацію полів, відображає підказки і готує дані до розрахунку.

Виняткова ситуація 1: індекси елементів посилаються на неіснуючі вузли – система блокує дію і підсвічує відповідні поля.

Виняткова ситуація 2: відсутні граничні умови – система відображає повідомлення про некоректну схему і блокує кнопку обчислення.

Прецедент «Ввести параметри через JSON-редактор».

Призначення: даний варіант використання надає можливість користувачу задати модель як JSON-структуру.

Передумови: користувач знаходиться у вкладці «JSON-редактор» панелі введення.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач вставляє/редагує JSON у відповідному полі. Опціонально користувач може форматувати JSON.

Виняткова ситуація 1: синтаксична помилка JSON – система показує повідомлення «Некоректний JSON» і не запускає розрахунок.

Виняткова ситуація 2: семантична помилка даних – система виводить відповідне повідомлення та блокує кнопку обчислення.

Прецедент «Імпортувати параметри з JSON-файлу».

Призначення: даний варіант використання надає можливість користувачу імпортувати модель із локального JSON-файлу.

Передумови: користувач знаходиться у вкладці «Завантаження файлу» панелі введення.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач обирає JSON-файл, та за потреби вводить перевизначення «scale». Система виконує валідацію та зчитує файл локально для попереднього перегляду.

Виняткова ситуація 1: неправильний тип файлу/порожній файл – система відхиляє завантаження і показує відповідне повідомлення.

Виняткова ситуація 2: структура моделі некоректна – система відображає відповідне повідомлення про помилку і блокує кнопку обчислення.

Прецедент «Отримати приклад параметрів».

Призначення: даний варіант використання надає можливість користувачу заповнити JSON-редактор типовим прикладом моделі.

Передумови: користувач знаходиться у вкладці «JSON-редактор».

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає кнопку «Приклад». Система виконує запит до відповідного ендпоінту, підставляє отриманий JSON у редактор і готує його до запуску розрахунку.

Виняткова ситуація 1: сервер недоступний – система виводить повідомлення «Сервер недоступний, спробуйте пізніше».

Виняткова ситуація 2: приклад містить некоректні для поточного валідатора значення – система підсвічує проблемні місця і пропонує виправити.

Прецедент «Запустити розрахунок».

Призначення: даний варіант використання надає можливість користувачу ініціювати обчислення CEA для введеної моделі.

Передумови: параметри моделі задані та пройшли первинну валідацію.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає «Розрахувати». Система відправляє запит на відповідний ендпоінт, відображає індикатор виконання, отримує відповідь і зберігає її для подальшого перегляду.

Виняткова ситуація 1: сингулярна або погано обумовлена система – система повідомляє “System is singular or ill-conditioned” з короткою підказкою.

Виняткова ситуація 2: технічна помилка сервера/мережі – система виводить відповідне повідомлення про помилку і пропонує повторити спробу.

Прецедент «Переглянути результати».

Призначення: даний варіант використання надає можливість користувачу переглянути результати виконаного розрахунку.

Передумови: обчислення успішно завершено та дані результатів отримані.

Основний потік подій: даний варіант використання починає виконуватися, коли система відображає результати у двох формах: табличній та графічній. Користувач переглядає числові значення і візуалізацію деформованої/оригінальної схеми. Даний варіант використання включає прецеденти «Отримати табличні дані» та «Отримати візуалізацію».

Виняткова ситуація 1: немає результатів – система показує повідомлення «Немає даних для відображення».

Виняткова ситуація 2: несумісність версій клієнт/сервер – система повідомляє про помилку форматів і пропонує оновити сторінку або конфігурацію.

Прецедент «Отримати табличні дані».

Призначення: даний варіант використання надає можливість користувачу переглянути числові результати у вигляді таблиць.

Передумови: обчислення успішно завершено та дані результатів отримані.

Основний потік подій: даний варіант використання починає виконуватися, коли система показує таблиці переміщень вузлів, реакцій у закріплених опорах, осьових сил у елементах, а також метадані. Користувач переглядає та аналізує значення.

Виняткова ситуація 1: в окремих полях відсутні дані – система відображає символ «–».

Виняткова ситуація 2: числові переповнення/NaN – система підсвічує аномальні значення та пропонує перевірити модель.

Прецедент «Отримати візуалізацію».

Призначення: даний варіант використання надає можливість користувачу переглянути початкову та деформовану схеми конструкції.

Передумови: обчислення успішно завершено та дані результатів отримані.

Основний потік подій: даний варіант використання починає виконуватися, коли система масштабує сцену під контейнер і відображає елементи/вузли у вигляді SVG-графіки. За замовчуванням показуються шари «Оригінальна» та «Деформована». Користувач може керувати шарами через відповідні перемикачі.

Виняткова ситуація 1: дані візуалізації неповні – система показує попередження та не відображає пошкоджений шар.

Прецедент «Увімкнути шар».

Призначення: даний варіант використання надає можливість користувачу відобразити вибраний шар візуалізації.

Передумови: користувач переглядає візуалізацію та вимкнено.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач активує перемикач шару, система відображає відповідний шар і оновлює відображення сцени.

Виняткова ситуація 1: шар недоступний – система блокує перемикач та показує підказку.

Виняткова ситуація 2: конфлікт відображення – система застосовує пріоритети шарів.

Прецедент «Вимкнути шар».

Призначення: даний варіант використання надає можливість користувачу приховати вибраний шар візуалізації.

Передумови: користувач переглядає візуалізацію та шар увімкнено.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач деактивує перемикач шару, система приховує відповідний шар і оновлює відображення сцени.

Виняткова ситуація 1: шар недоступний – система блокує перемикач та показує підказку.

Прецедент «Ввести вузли».

Призначення: даний варіант використання надає можливість користувачу задати координати вузлів моделі.

Передумови: користувач обрав спосіб введення параметрів через «Форма» або «JSON-редактор».

Основний потік подій: даний варіант використання починає виконуватися, коли користувач додає один або кілька вузлів, вводячи координати x, y. Система зберігає список у внутрішньому стані або JSON, перевіряє числові значення, показує підказки щодо формату та відображає оновлену нумерацію вузлів.

Виняткова ситуація 1: порожній список вузлів або нечислові значення координат – система підсвічує поля з помилками та блокує запуск розрахунку.

Виняткова ситуація 2: спроба видалити вузол, який використовується в одному чи більше елементів – система повідомляє про залежності та пропонує спочатку змінити/видалити відповідні елементи.

Прецедент «Ввести елементи».

Призначення: даний варіант використання надає можливість користувачу задати стержневі елементи між парами вузлів із параметрами перерізу та матеріалу.

Передумови: користувач обрав спосіб введення параметрів через «Форма» або «JSON-редактор» та у моделі існує щонайменше два вузли.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач додає елемент, задає індекси вузлів n1, n2 та параметри A (площа) і E (модуль пружності). Система валідує, що індекси у межах списку вузлів, $n1 \neq n2$, значення $A > 0$, $E > 0$, та зберігає зміни.

Виняткова ситуація 1: неправильні індекси або $n1 = n2$ – система підсвічує помилку та не дозволяє зберегти рядок.

Виняткова ситуація 2: дублювання зв’язків між тими самими вузлами або нульові/від’ємні A/E – система попереджає про некоректні/підозрілі значення і блокує розрахунок до виправлення.

Прецедент «Ввести опори».

Призначення: даний варіант використання надає можливість користувачу задати граничні умови закріплення для вибраних вузлів.

Передумови: користувач обрав спосіб введення параметрів через «Форма» або «JSON-редактор» та у моделі існує щонайменше два вузли.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач обирає вузол(-и) і встановлює прапорці ux та/або uy для фіксації відповідних ступенів свободи. Система зберігає налаштування і виконує перевірку достатності закріплень для уникнення механізму.

Виняткова ситуація 1: недостатні опори – система попереджає «недостатні закріплення» і блокує розрахунок.

Виняткова ситуація 2: конфліктні/надлишкові закріплення – система попереджає про потенційні труднощі і пропонує скоригувати опори.

Прецедент «Ввести навантаження».

Призначення: даний варіант використання надає можливість користувачу задати вузлові навантаження за компонентами Fx та Fy.

Передумови: користувач обрав спосіб введення параметрів через «Форма» або «JSON-редактор» та у моделі існує щонайменше два вузли.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач обирає вузол(-и) і вводить значення сил Fx, Fy. Система перевіряє числовий формат та зберігає дані.

Виняткова ситуація 1: нечислові значення/відсутні поля для вибраного вузла – система підсвічує помилки і блокує запуск розрахунку.

Виняткова ситуація 2: навантаження задані на вузол, який відсутній у переліку вузлів – система повідомляє про невідповідність індексів.

Прецедент «Ввести масштаб деформацій».

Призначення: даний варіант використання надає можливість користувачу встановити коефіцієнт масштабування відображення деформацій у візуалізації.

Передумови: користувач обрав спосіб введення параметрів через «Форма» або «JSON-редактор».

Основний потік подій: даний варіант використання починає виконуватися, коли користувач вводить додатне числове значення “scale”, система приймає його, та за потреби корегує до встановлених меж і застосовує під час побудови деформованої форми.

Виняткова ситуація 1: некоректне значення масштабу – система показує підказку, нормалізує або блокує значення до виправлення.

Виняткова ситуація 2: невідповідність між масштабом та одиницями/розмірами сцени – система пропонує використати типовий діапазон значень.

2.3 Діаграма діяльності

Діаграма діяльності є одним із типів діаграм у мові моделювання UML, що використовується для графічного відображення потоку керування та послідовності дій у системі. Вона демонструє етапи або завдання, які виконуються під час певного процесу чи реалізації окремого варіанта використання. Дії на такій діаграмі позначаються прямокутниками із заокругленими кутами, а напрямок виконання процесу показується стрілками, які формують потік керування [5, 6].

Цей тип діаграм дозволяє моделювати не лише послідовні, а й альтернативні або паралельні сценарії виконання, використовуючи вилки та з’єднання. Завдяки об’єктним потокам можна відобразити передачу даних, об’єктів або повідомлень між окремими діями під час процесу. Початковий і кінцевий вузли визначають відповідно точку старту та завершення послідовності дій [5, 6].

Діаграма діяльності є універсальним та ефективним інструментом для моделювання й документування різноманітних процесів – від бізнес-процедур і робочих потоків до реалізації варіантів використання та внутрішньої логіки методів. Вона допомагає наочно досліджувати складні процеси, виявляти можливі вузькі місця або неефективні ділянки, а також сприяє кращій взаємодії між бізнес-аналітиками, розробниками й користувачами системи [5, 6].

На рисунку 2.2 надано ДД прецеденту «Запустити розрахунок».

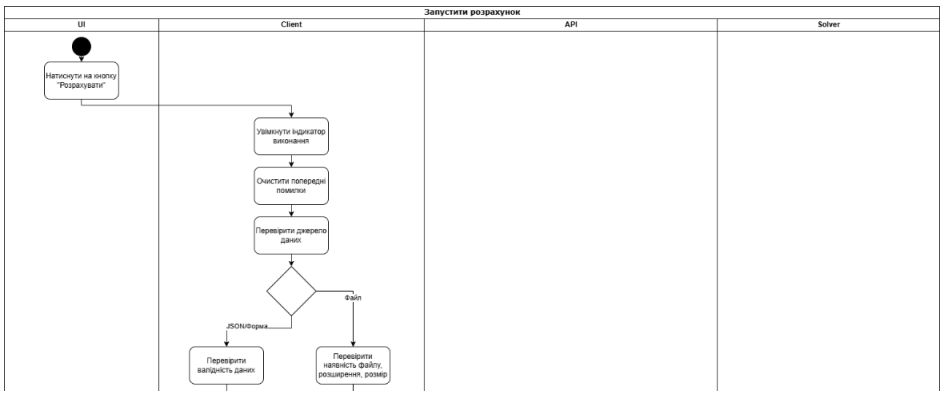


Рисунок 2.2 – ДД прецеденту «Запустити розрахунок» (аркуш 1)

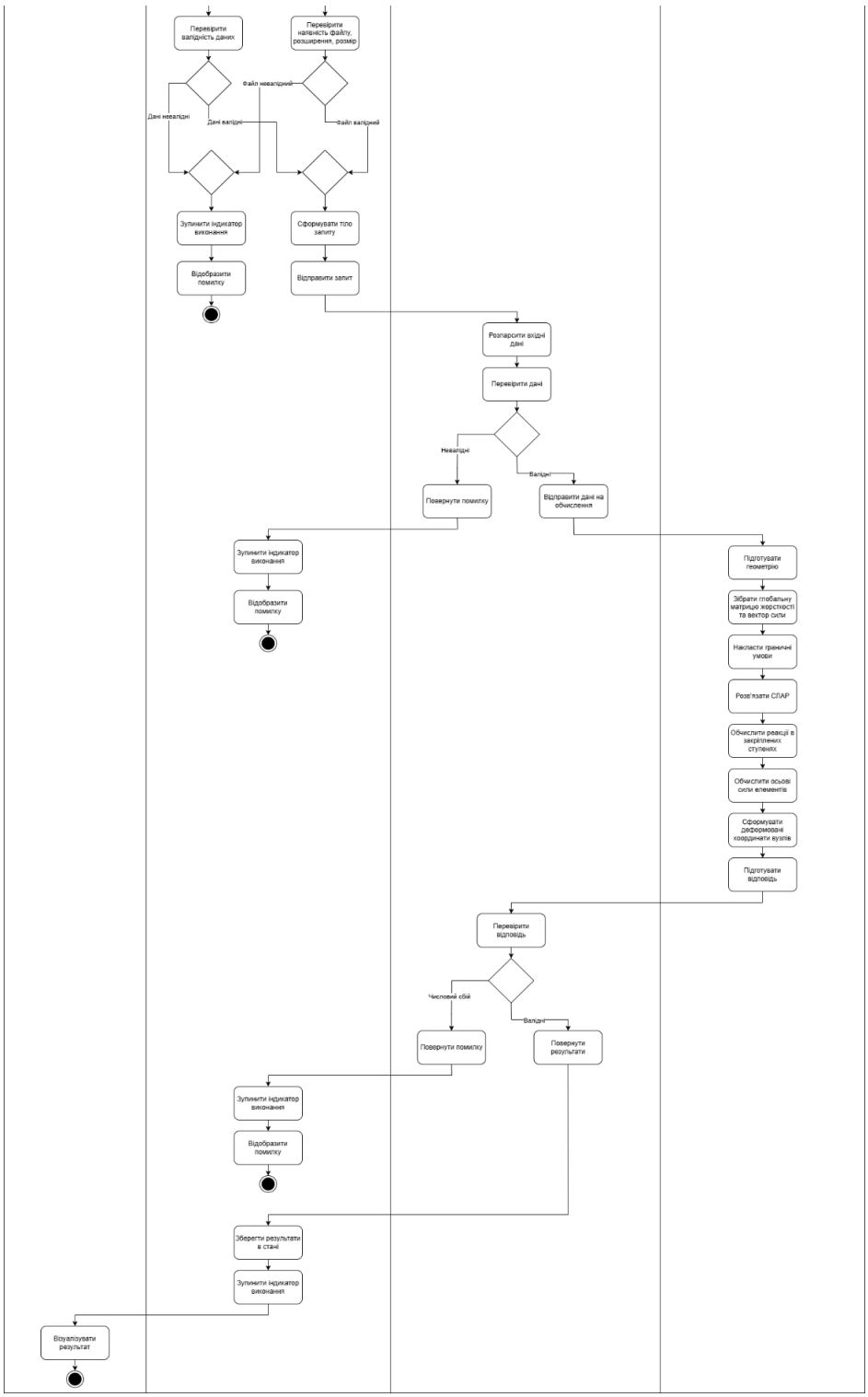


Рисунок 2.2 – ДД прецеденту «Запустити розрахунок» (аркуш 2)

2.4 Діаграма послідовності

Діаграма послідовності є одним із типів діаграм взаємодії в UML. Вона використовується для графічного зображення процесу обміну повідомленнями

між об’єктами системи у часовій послідовності. Така діаграма дає змогу зрозуміти порядок комунікації між компонентами та відобразити внутрішню поведінку системи під час виконання певного сценарію чи варіанта використання [5, 6].

На рисунку 2.3 наведено ДП для прецеденту «Запустити розрахунок».

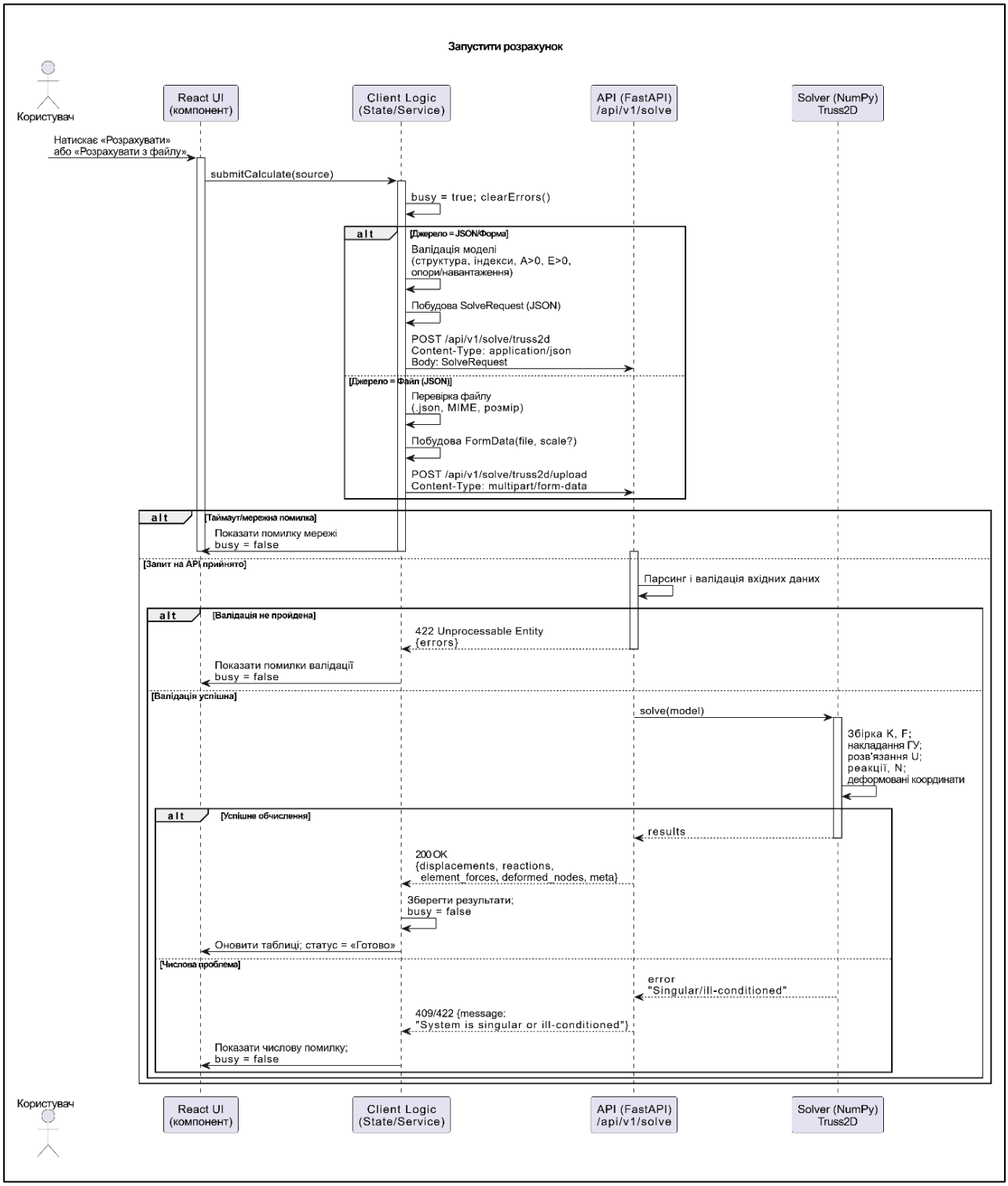


Рисунок 2.3 – ДП прецеденту «Запустити розрахунок»

На вертикальній осі діаграми розміщуються об’єкти або учасники взаємодії, які називають лініями життя. Вони показують існування об’єкта протягом певного проміжку часу. Горизонтальна вісь відображає часову послідовність подій, а повідомлення між об’єктами позначаються горизонтальними стрілками, що вказують на відправника, отримувача та напрямок комунікації [5, 6].

Діаграма послідовності дозволяє показати не лише обмін повідомленнями, а й внутрішню активність об’єктів, а саме виклики методів, операції тощо. Для цього використовуються фрагменти взаємодії, які охоплюють певні частини процесу та допомагають деталізувати його поведінку [5, 6].

Такий тип діаграм є корисним для аналізу складних сценаріїв взаємодії між компонентами системи, документування логіки комунікації, виявлення можливих проблем або затримок у передачі повідомлень. Він також сприяє кращому розумінню динамічної поведінки системи різними учасниками розробки [5, 6].

Діаграма послідовності може включати додаткові елементи для глибшого моделювання взаємодій. Умовні або циклічні фрагменти використовуються для опису розгалужень і повторень, а паралельні фрагменти – для відображення одночасного виконання кількох потоків взаємодії. Також можливе зображення створення та знищення об’єктів у процесі, а асинхронні повідомлення чи виклики із зворотним викликом позначаються спеціальними символами, що підвищує точність і наочність моделі [5, 6].

2.5 Діаграма розгортання

Діаграма розгортання в UML призначена для наочного відображення архітектури системи з погляду апаратного середовища, у якому вона функціонує. Вона демонструє, як елементи програмного забезпечення розміщуються та взаємодіють на фізичних або віртуальних вузлах. Вузли на діаграмі з’єднані мережевими лініями, що відображають способи комунікації між ними [5, 6].

На діаграмі також показуються артефакти – виконувані файли, бібліотеки, бази даних або інші компоненти, які розгортаються на відповідних вузлах. Така візуалізація дає змогу зрозуміти структуру системи з технічної точки зору, визначити взаємозалежності між компонентами та оцінити архітектурні зв’язки [5, 6].

Завдяки діаграмі розгортання можна проаналізувати вимоги до апаратного забезпечення, оптимізувати розподіл ресурсів, а також детально описати особливості комунікації між компонентами, зокрема використані протоколи зв’язку чи версії програмного забезпечення. Це робить діаграму важливим інструментом для планування, документування та супроводу складних розподілених систем [5, 6].

На рисунку 2.4 показано розгортання системи.

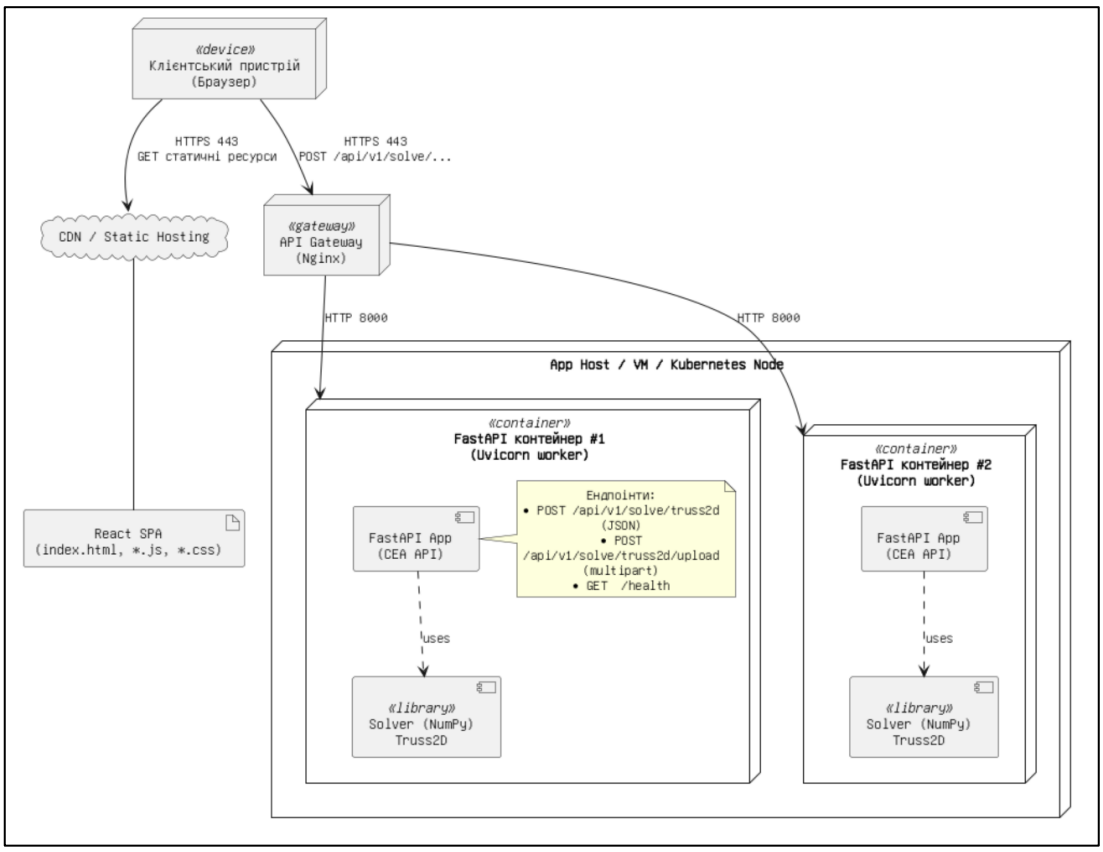


Рисунок 2.4 – Діаграма розгортання

Використання UML-нотації дає змогу послідовно описати функціональні можливості, алгоритми, інтеграційні взаємодії, процес розгортання та статичну

структуру доменної області сервісу. Такий підхід забезпечує узгодженість між вимогами, архітектурними рішеннями та реалізацією.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Опис інструментів розробки

Система реалізована з фокусом на прозорій архітектурі «клієнт-сервер», масштабованості та відтворюваності середовища.

При створені системи використовувалися наступні інструменти.

React – бібліотека для створення SPA з функціональними компонентами та хуками [11].

TypeScript – статична типізація для підвищення надійності та зрозумілості коду [15].

Vite – швидкий дев-сервер і зборщик фронтенда [19].

FastAPI – високопродуктивний серверний фреймворк Python із валідацією даних черз Pydantic [4].

Uvicorn – ASGI-сервер для запуску FastAPI [16].

NumPy – пакет для чисельної лінійної алгебри [8].

Pydantic – схеми запитів/відповідей та серверна валідація [9].

Python 3.11 / Node.js LTS – базові платформи для клієнта та сервера [7 ,10].

pytest + httpx/TestClient – юніт- та інтеграційні тести серверної частини [10, 14].

Vitest + React Testing Library – юніт-тестування клієнтської частини [12].

3.2 Основні компоненти та модулі системи

Основним бізнес-процесом є обчислення дискретної моделі, тому основним бізнес-прецедентом є «Запустити розрахунок». Тому основними компонентами/модулями є ті, що забезпечують цей процес.

Зі сторони клієнту можна виділити наступні сервіси та компоненти:

– api/fea.ts – сервіс, що відповідає за виклики API;

– components/ModelForm.tsx – компонент, який відповідає за відображення форми введення вузлів, елементів, опор, навантажень і scale;
– components/DataTable.tsx – компонент, який відповідає за відображення результату розрахунку в табличному вигляді;
– components/Viewer2D.tsx – компонент, який відповідає за візуалізацію даних у SVG-переглядачі, а також надає інструменти керування шарами.

Зі сторони серверу можна виділити файли, що відповідають за прийняття, обробку та повернення результату.

– app/routers/solve.py – відповідає за ендпоінти;
– app/models.py – містить Pydantic-схеми запиту/відповіді;
– app/solver/truss2d.py – відповідає за основний алгоритм.

Нижче наведено функцію apiSolve, яка відповідає за відправку даних на сервер, які були створені через форму або JSON-редактор (рис. 3.1).

```
export async function apiSolve(baseUrl: string, body: SolveRequest): Promise<SolveResponse> {
    const r :Response  = await fetch(`${baseUrl}/api/v1/solve/truss2d`, {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(body),
    });
    if (!r.ok) throw new Error(`Solve failed: ${r.status}`);
    return r.json();
}
```

Рисунок 3.1 – Функція apiSolve

З повною об’єднаною реалізацією серверної частини можна ознайомитися у додатку А, з основними файлами клієнтської сторони – у додатках Б, В, Г.

3.3 Встановлення та налаштування оточення

Початковими вимогами для розгортання системи є встановлені відповідні версії середовища, а саме Node.js LTS та Python 3.11 [7 ,10].

Після встановлення або оновлення залежностей середовища, переходимо до розгортання клієнтської частини.

Для цього будемо використовувати зборщик Vite з командою npm create vite@latest fea-client -- --template react-ts (див. рис. 3.2), яка дозволяє створити базову структуру vite-проєкту з автоматичним підключенням React у vite.config.js (див. рис. 3.3) та згенерувати файли для роботи з TypeScript [19].

```
~/projects/fea$ npm create vite@latest fea-client -- --template react-ts
```

Рисунок 3.2 – Створення клієнтської частини

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

// https://vite.dev/config/

export default defineConfig({  no usages
  plugins: [react()],
})
```

Рисунок 3.3 – Файл vite.config.js з плагіном React

Після створення проєкту, переходимо до його дикерторії за допомогою команди cd fea-client та встановлюємо залежність @fluentui/react-components за допомогою команди npm i @fluentui/react-components (рис. 3.4).

```
~/projects/fea$ npm i @fluentui/react-components
```

Рисунок 3.4 – Встановлення @fluentui/react-components

Для перевірки правильності розгортання клієнтської частини використовуємо команду npm run dev, яка запускає команду vite dev, що розгортає легкий сервер розробки. За замовчуванням сервер запускається на порту 5173 (рис. 3.5) [19].



Рисунок 3.5 – Запуск клієнтської частини

Далі переходимо до розгортання серверної частини, для цього будемо використовувати віртуальне середовище розробки, щоб уникнути встановлення глобальних залежностей. У корені проєкту створюємо теку під сервер, переходимо до неї та викликаємо команду python3 -m venv venv (рис. 3.6) [10, 14].

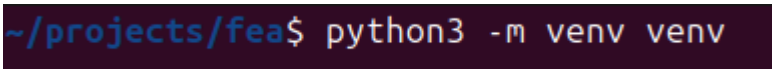


Рисунок 3.6 – Створення віртуального середовища

Далі активуємо віртуальне середовище за допомогою команди source venv/bin/activate (рис. 3.7) та встановляємо необхідні залежності (рис. 3.8) [10, 14].

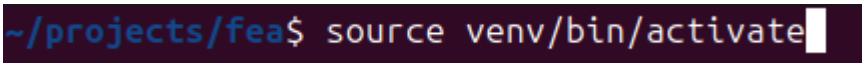


Рисунок 3.7 – Активація віртуального середовища

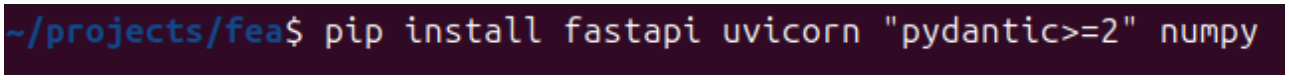


Рисунок 3.8 – Встановлення залежностей

Dev-сервер запускаємо у тому же віртуальному середовищі за допомогою команди uvicorn app.main:app --reload --host 0.0.0.0 --port 8000 (рис. 3.9) [16].

```
svarog@svarog:~/projects/fea/server$ source venv/bin/activate
(venv) svarog@svarog:~/projects/fea/server$ uvicorn main:app --reload --port 8000
INFO:     Will watch for changes in these directories: ['/home/svarog/projects/fea/s
erver']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [10353] using StatReload
INFO:     Started server process [10355]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

Рисунок 3.9 – Запуск dev-серверу

3.4 Опис API

У системі реалізовано REST API, що забезпечує взаємодію між клієнтом і сервером через HTTP-запити. Основною операцією є POST-запит із application/json або multipart/form-data. Усі запити обробляються через FastAPI, опис ендпоінтів наведено у Swagger UI (рис. 3.10).

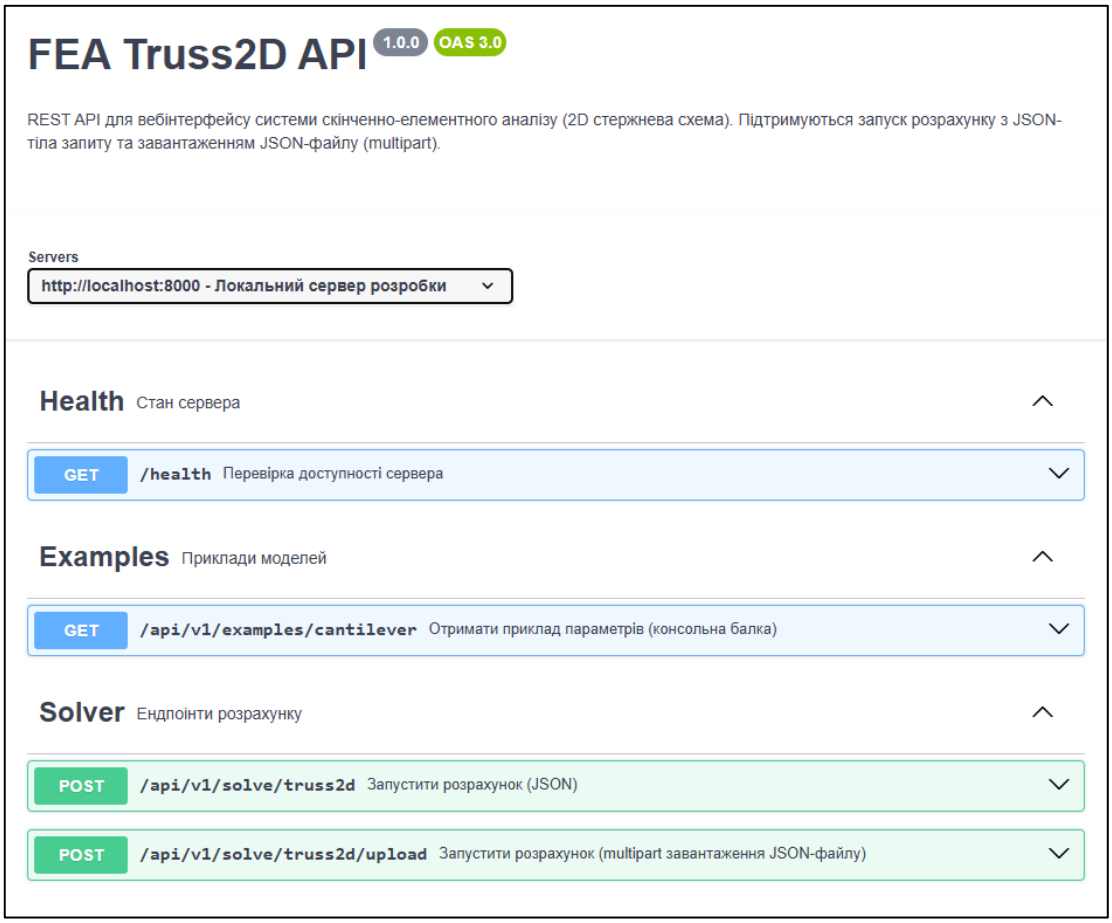


Рисунок 3.10 – Опис API у Swagger

Розглянемо ендпоінти системи.

GET /health відповідає за перевірку доступності сервера.

GET /api/v1/examples/cantilever відповідає за отримання прикладу параметрів моделі для JSON-редактора.

POST /api/v1/solve/truss2d (application/json) відповідає за запуск розрахунку з тіла запиту. Викликається при використані форми або JSON-редактора. На рисунку 3.11 надано приклад запиту.


```
{
 "nodes": [[0,0],[1,0],[1,1]],
 "elements": [{"n1":0,"n2":1,"A":0.01,"E":210000000000}],
 "supports": [{"node":0,"ux":true,"uy":true}],
 "loads": [{"node":2,"Fx":0,"Fy":-1000}],
 "scale": 50
}
```


Рисунок 3.11 – Приклад запиту

POST /api/v1/solve/truss2d/upload (multipart/form-data) відповідає за запуск розрахунку з JSON-файлу.

Система повертає наступні коди помилок:

– 400 Bad Request – некоректні поля/типи;
– 409 Conflict/422 Unprocessable Entity – числова проблема (сингулярність/погана обумовленість) або семантична помилка;
– 413 Payload Too Large – перевищено ліміт завантаження;
– 415 Unsupported Media Type – невідповідний MIME;
– 500 Internal Server Error – неочікувана помилка.

У разі успіху, система повертає статус 200 OK. Приклад успішної відповіді наведено на рисунку 3.12.

```
{
 "displacements": [{"node":0,"ux":0,"uy":0}, ...],
 "reactions": [{"node":0,"Rx":0,"Ry":0}, ...],
 "element_forces": [{"element":0,"N":123.45}, ...],
 "deformed_nodes": [[x1',y1'], [x2',y2'], ...],
 "meta": {"n_nodes":3,"n_elements":1,"time_ms":2}
}
```


Рисунок 3.12 – Приклад успішної відповіді

3.5 Тестування проєкту

Проведено модульні тести клієнта та сервера для перевірки цілісності бізнес-логіки, стабільності взаємодії з API та коректності UI-станів.

Для тестування зі сторони клієнта використовувався Vitest з React Testing Library. Ці два інструменти повністю покривають категорії тестування клієнта, від перевірки стану кнопок до API-запитів [12, 20].

На рисунку 3.13 надано приклад тестування функції apiSolve.

```
describe('apiSolve', () => {
    const payload: SolveRequest = {
        nodes: [{x:0, y:0},{x:1, y:0}, {x:1, y:1}],
        elements: [
            { n1:0, n2:1, A:0.01, E:210_000_000_000 },
            { n1:1, n2:2, A:0.01, E:210_000_000_000 },
            { n1:0, n2:2, A:0.01, E:210_000_000_000 }
        ],
        supports: [{ node:0, ux:true, uy:true }, { node:1, ux:false, uy:true }],
        loads: [{ node:2, fx:0, fy:-1000 }],
        scale: 50
    };

    beforeEach(() => {
        const mockResponse: SolveResponse = {
            displacements: [], reactions: [], element_forces: [], deformed_nodes: [],
            meta: { n_nodes: 3, n_elements: 3, time_ms: 2 }
        };
        vi.spyOn(global, 'fetch').mockResolvedValue({
            ok: true,
            status: 200,
            json: async () => mockResponse,
        } as unknown as Response);
    });
```

Рисунок 3.13 – Приклад тестування функції apiSolve

Зі сторони серверу використовувався стандартний інструмент – pytest. Він дозволяє протестувати як валідацію, так і різні чисельні випадки [14].

Метою проведеного тестування було підтвердити коректність валідації даних, стабільність чисельного розв’язання та правильну обробку станів інтерфейсу

На рисунках 3.14 – 3.15 наведено приклади unit-тестування для функцій сервісу fea.tsx.

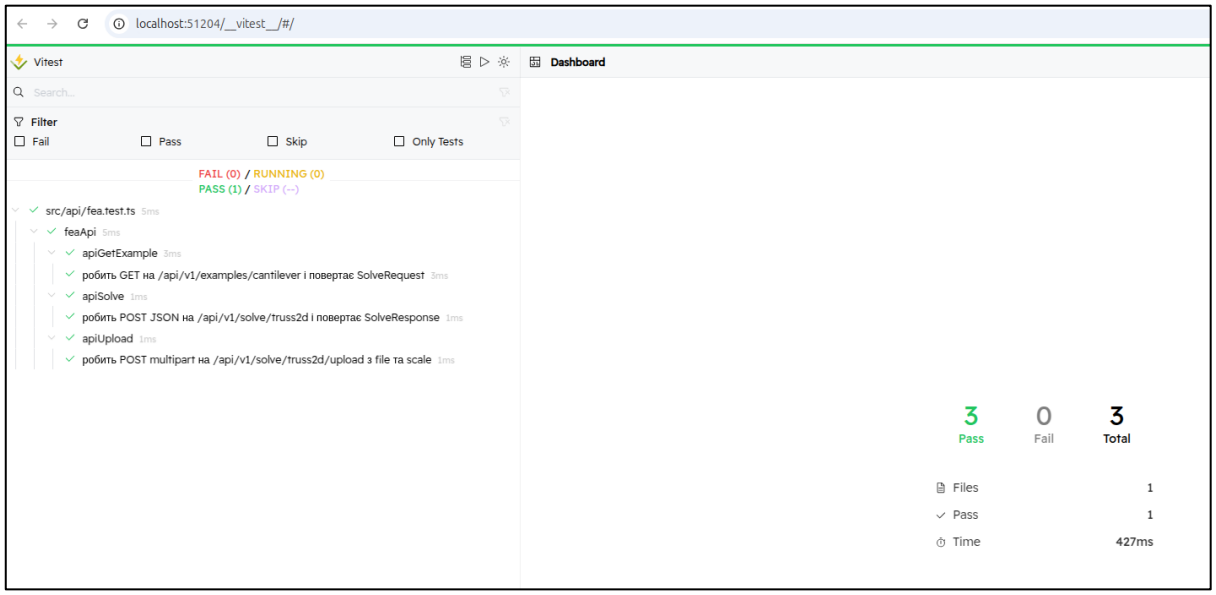


Рисунок 3.14 – Unit-тест функцій сервісу fea.tsx (браузер)

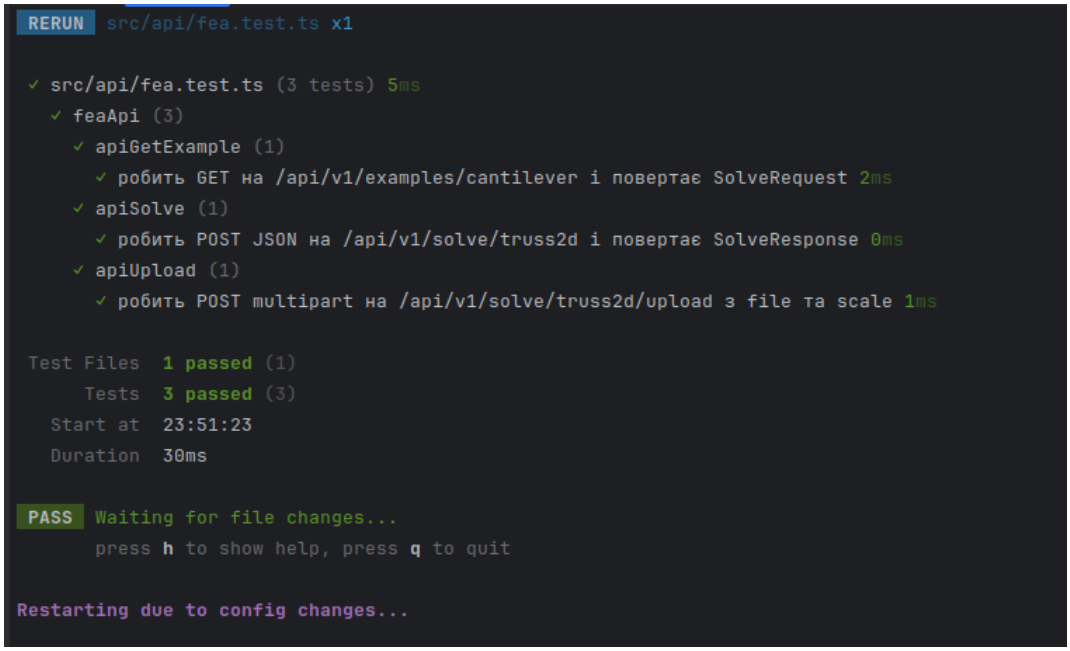


Рисунок 3.15 – Unit-тест функцій сервісу fea.tsx (консоль)

3.6 Керівництво користувача

3.6.1 Головна сторінка

Після відкриття сторінки у браузері, користувачу відображається одноколонковий інтерфейс із трьома режимами введення параметрів: «Форма», «Завантаження файлу», «JSON-редактор». Нижче розміщені секції «Результати» і «Візуалізація» з перемикачами шарів (рис. 3.16).

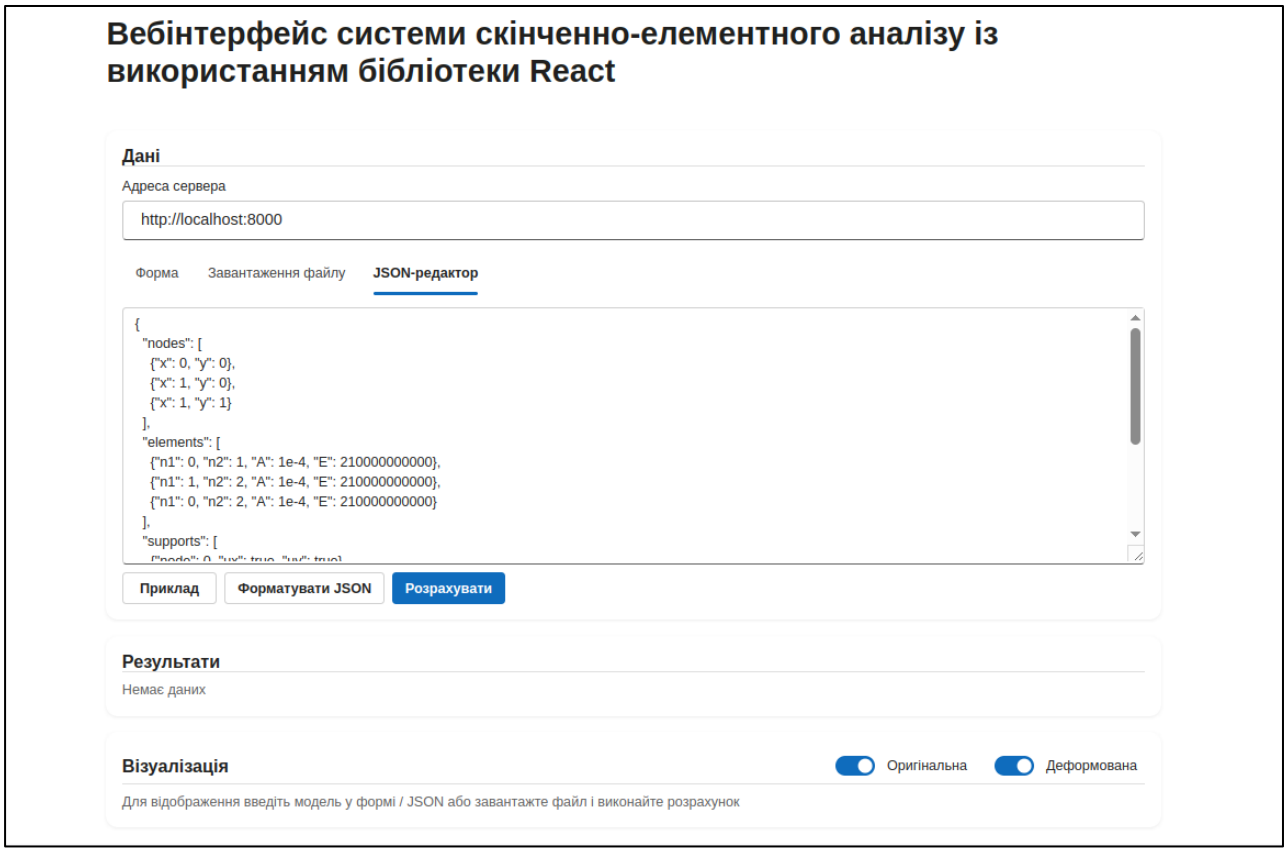


Рисунок 3.16 – Головна сторінка

3.6.2 Введення даних моделі через форму

Після того, як користувач натискає на режим введення даних «Форма», система відображає інтерактивну форму, яка має значення за замовчуванням (рис. 3.17).

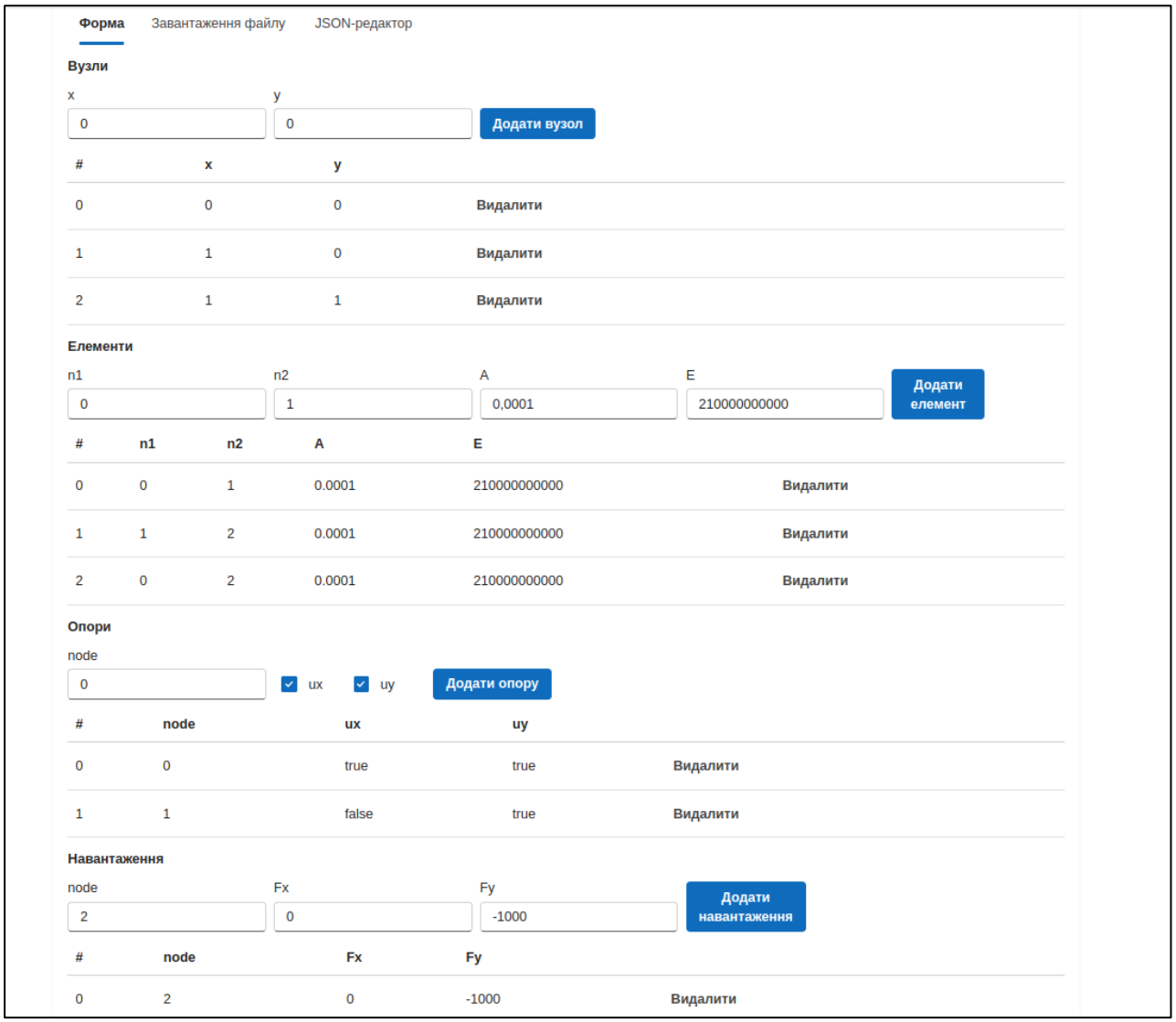


Рисунок 3.17 – Режим введення «Форма»

Користувач може додавати вузли (x, y), елементи (n1, n2, A, E), задавати опори (ux, uy) і навантаження (Fx, Fy), встановлювати scale. Кнопка «Розрахувати» за замовчуванням є неактивною, і активується тільки після успішного проходження базової валідації.

3.6.3 Введення даних моделі через JSON-редактор

Після того, як користувач натискає на режим введення даних «JSON-редактор», система відображає текстову область з попередньо завантаженим прикладом без форматування (рис. 3.18).

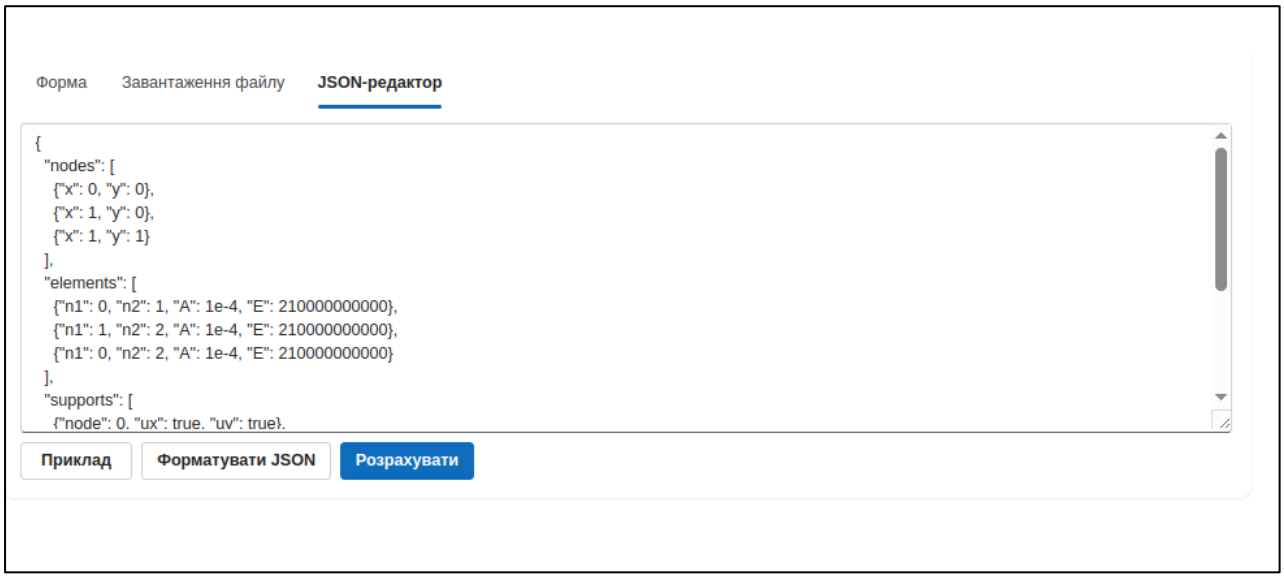


Рисунок 3.18 – Режим введення «JSON-редактор»

Користувач може вписувати з JSON напряму або підгрузити приклад моделі з серверу.

Опціонально можна використовувати кнопку «Форматувати JSON» для відображення вмісту текстової області у більш структурованому вигляді, також форматування дозволяє помітити можливі помилки синтаксису.

Кнопка «Розрахувати» відразу є активною за умови, що користувач не змінював початковий приклад або завантажив його з сервера.

У випадку ручного редагування система блокує кнопку до проходження базової валідації.

3.6.4 Імпорт моделі з файлу

Після того, як користувач натискає на режим введення даних «Завантаження файлу», система відображає форму з кнопкою «Вибрати файл».

Користувач обирає локальний .json із параметрами моделі та опційно може вказати власне значення scale.

Після завантаження, система перевіряє тип та розмір файлу, і якщо перевірка успішна, то активує кнопку «Розрахувати » (рис. 3.19 – 3.21).



Рисунок 3.19 – Форма без файлу

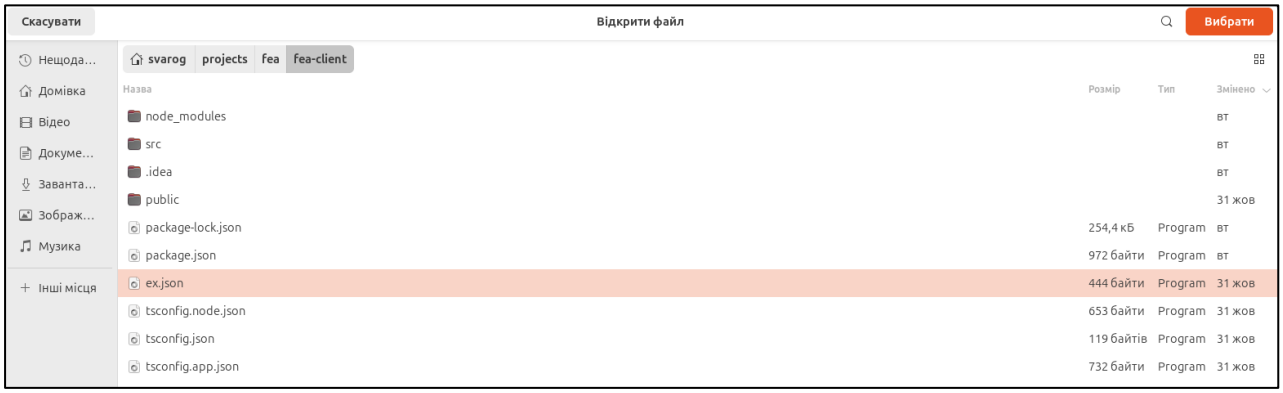


Рисунок 3.20 – Вибір файлу



Рисунок 3.21 – Форма з валідним файлом

3.6.5 Перегляд результатів

Після введення даних моделі будь-яким із перелічених способів, користувач може натиснути на кнопку «Розрахувати». Система відображає

анімацію виконання запиту і при успіху відображає таблицю результату розрахунку (рис. 3.22).



Результати

Переміщення вузлів

#	ux	uy
0	0.0000e+0	0.0000e+0
1	0.0000e+0	0.0000e+0
2	4.7619e-5	-4.7619e-5

Реакції в опорах (лише закріплені)

#	Rx	Ry
0	-2.0465e-15	-2.0465e-15
1	0.0000e+0	1000

Осьові сили в елементах (N)

елемент	N (тяга + / стиск -)
0	0.0000e+0
1	-1000
2	-1.5685e-15

Мета

параметр	значення
n_nodes	3
n_elements	3
sum_load	[0,-1000]
units	{"length":"m","force":"N","modulus":"Pa","area":"m^2"}

Рисунок 3.22 – Результат розрахунку

Таблиця відображається у секції «Результати» і містить інформацію про переміщення вузлів, реакцію в опорах, осьові сили в елементах, а також надає метадані розрахунку

3.6.6 Візуалізація

Після успішноного процесу розрахунку, система відображає оригінальну та деформовану схеми у масштабі відповідно до параметру scale (рис. 3.23).

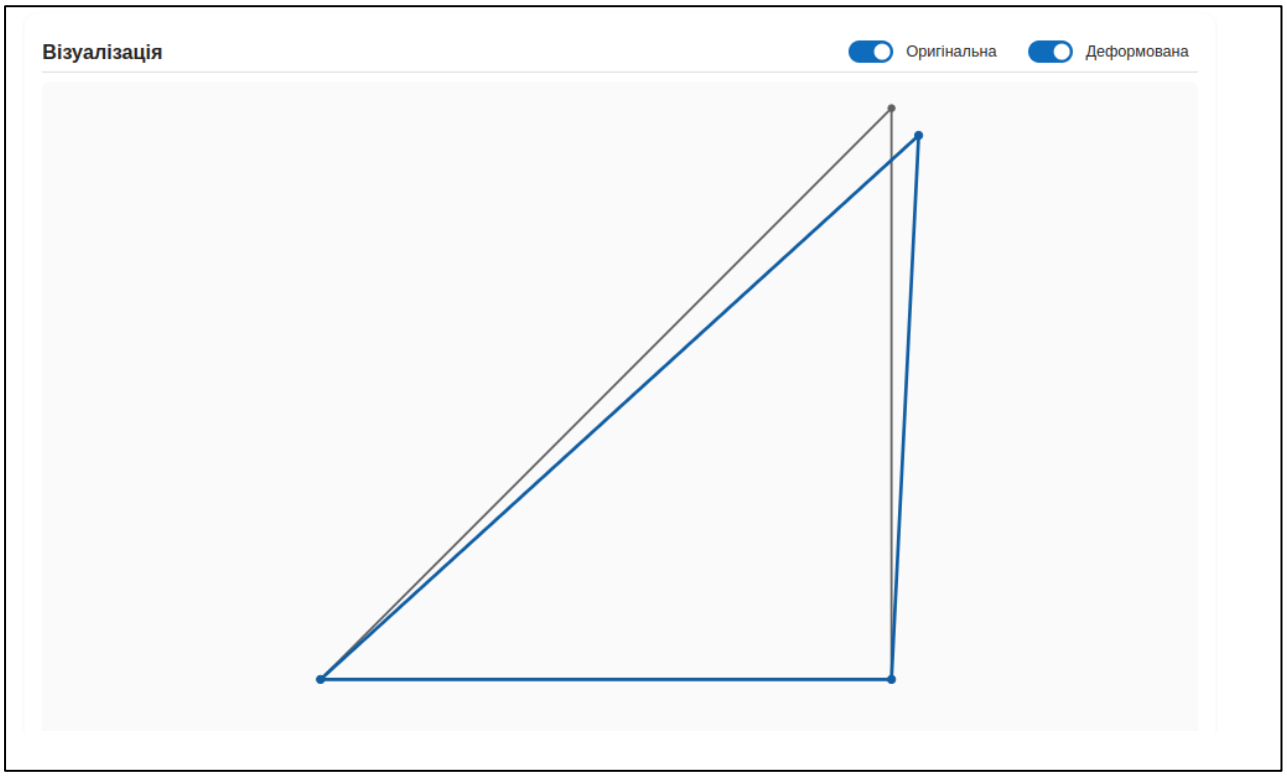


Рисунок 3.23 – Візуалізація розрахунку

Користувач може керувати відображенням схеми за допомогою віповідних перемикачів (рис. 3.24 – 3.25).

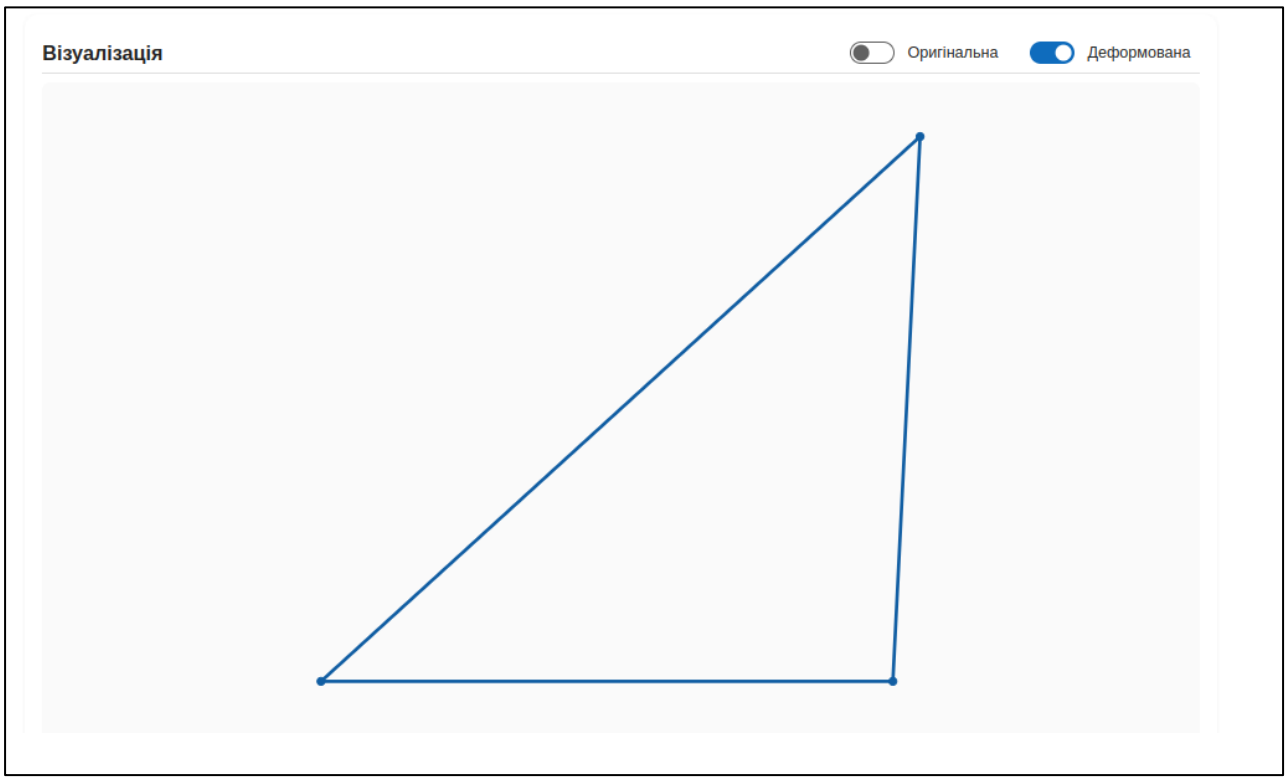


Рисунок 3.24 – Деформована модель

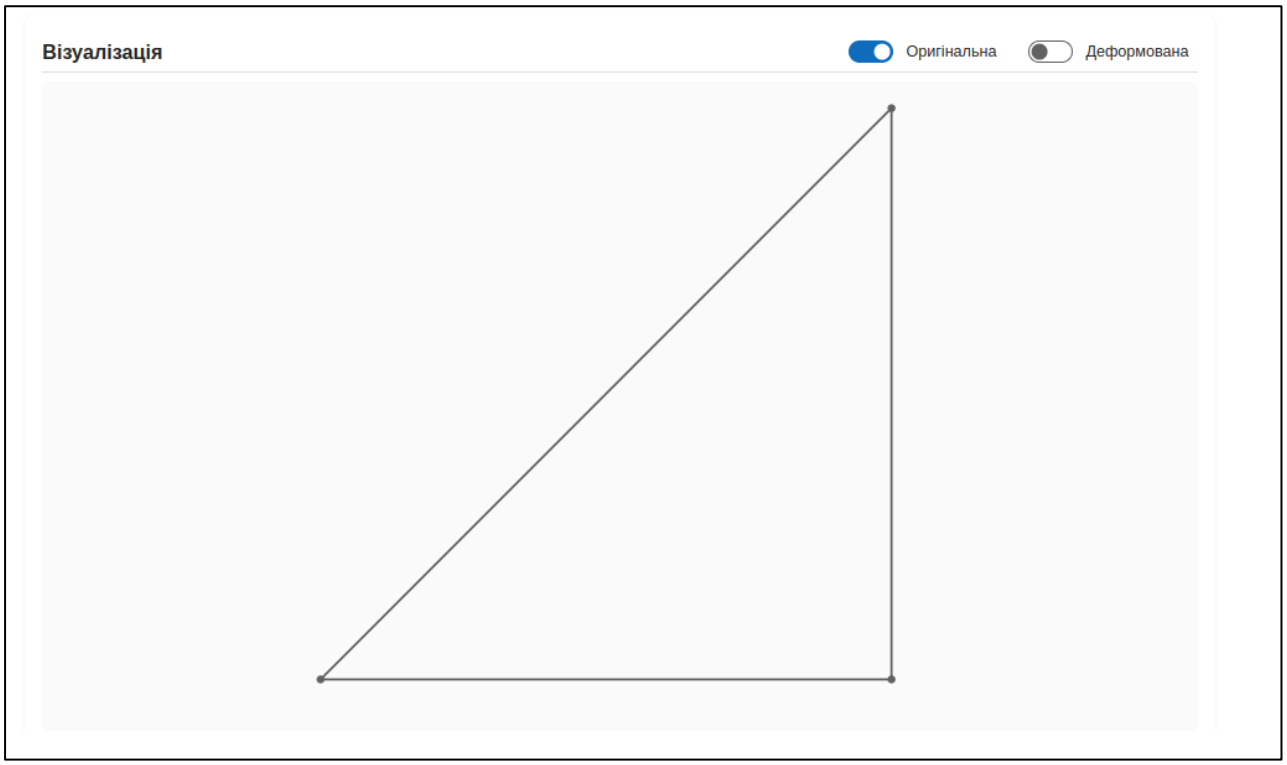


Рисунок 3.25 – Оригінальна модель

В залежності від кількості параметрів, змінюється і візуалізація. Для прикладу на рисунку 3.26 зображено вежу з трикутним верхом над двопролітною основою.

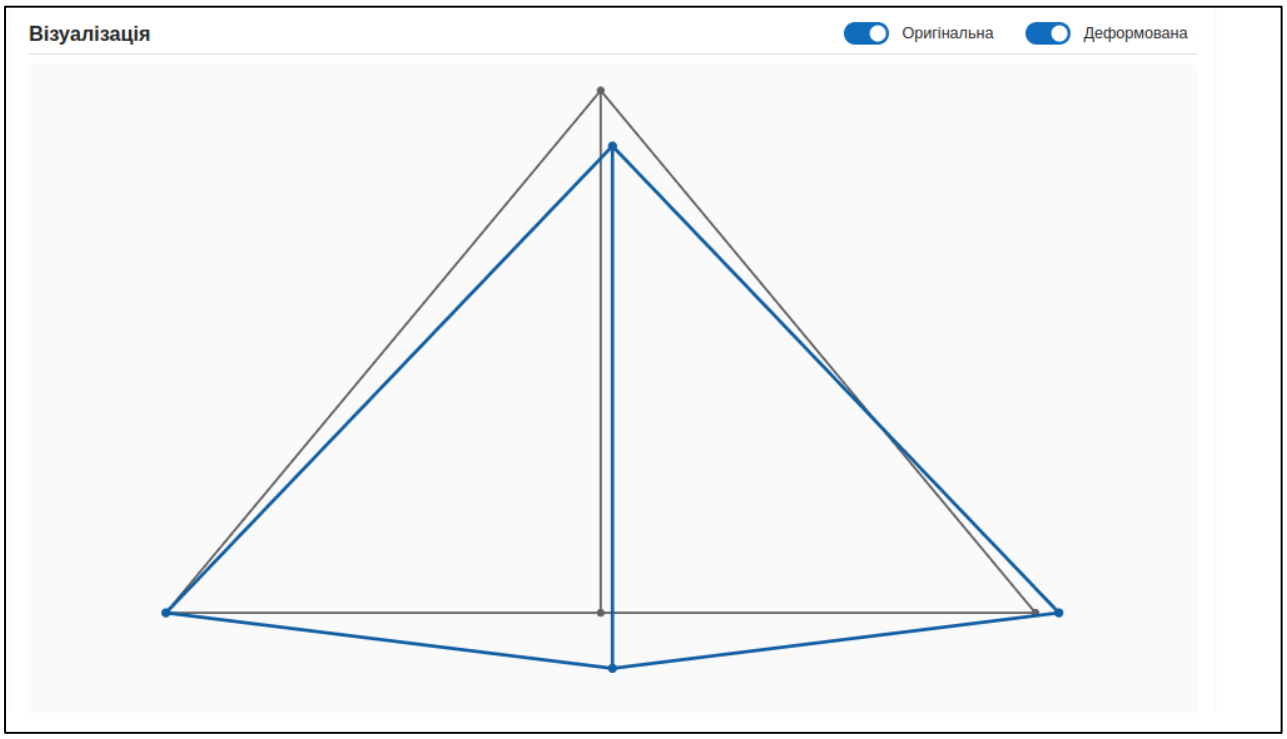


Рисунок 3.26 – Вежа з трикутним верхом над двопролітною основою

Система дозволяє розраховувати і візуалізовувати і більш складні моделі. На рисунку 3.27 зображено двопролітний кронштейн з 7 стрижнями і навантаження зверху праворуч.

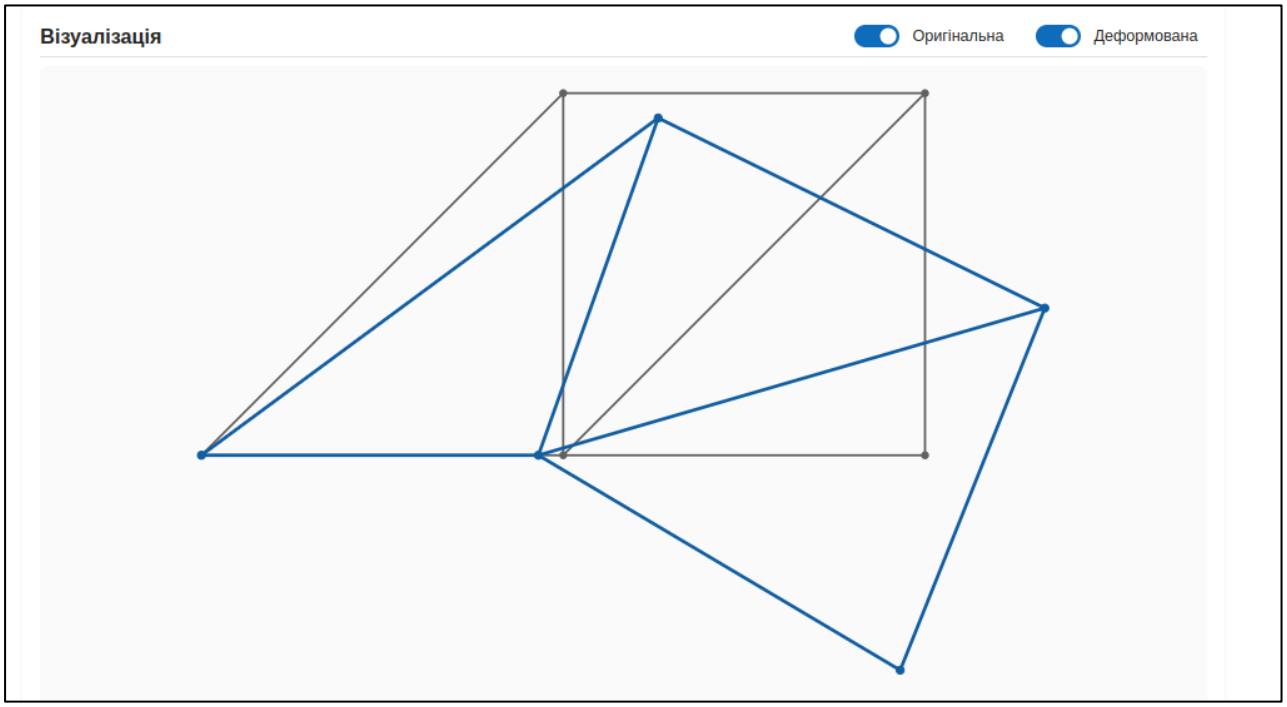


Рисунок 3.27 – Двопролітний кронштейн з 7 стрижнями і навантаження зверху праворуч

ВИСНОВКИ

У ході виконання кваліфікаційної роботи розроблено клієнт-серверний вебінтерфейс для базових задач скінченно-елементного аналізу 2D стержневих систем. Система дає змогу в браузері задавати геометрію та граничні умови, запускати розрахунок і отримувати результати у вигляді переміщень вузлів, реакцій і осьових зусиль, а також переглядати деформовану схему.

Відповідно до мети кваліфікаційної роботи реалізовано систему із використанням таких технологій:

– React для клієнтської частини;
– FastApi та NumPi для серверної логіки.

За поставленими задачами виконано такі етапи:

– сформовані функціональні й нефункціональні вимоги до системи;
– проаналізовані, описані та порівняні десктопні/хмарні CEA-інструменти;
– спроєктувано клієнт-серверну архітектуру та моделі даних;
– реалізувано клієнтську і серверну частину;
– забезпечена валідацію та обробка помилок;
– протестовані ключові сценарії та надано керівництво користувача.

ПЕРЕЛІК ПОСИЛАНЬ

1. Autodesk Fusion 360. URL: https://www.autodesk.com/products/fusion-360/overview (дата звернення: 02.08.2025).

2. ANSYS Cloud. URL: https://www.ansys.com/products/cloud (дата звернення: 30.07.2025).

3. COMSOL Server. URL: https://www.comsol.com/comsol-server (дата звернення: 03.08.2025).

4. FastAPI documentation. URL: https://fastapi.tiangolo.com/ (дата звернення: 12.09.2025).

5. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley Professional, 2003. 208 p.

6. Mrzyglocki F. UML Summarized: Key Concepts and Diagrams for Software Engineers, Architects, and Designers. Phillip Mrzyglocki, 2023. 59 p.

7. Node.js documentation. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 15.09.2025).

8. NumPy. URL: https://numpy.org/ (дата звернення: 16.09.2025).

9. Pydantic Validation. URL: https://docs.pydantic.dev/latest/ (дата звернення: 18.09.2025).

10. Python 3.11 documentation. URL: https://docs.python.org/3.11/ (дата звернення: 12.09.2025).

11. React dev. URL: https://react.dev/learn (дата звернення: 15.09.2025).

12. React Testing Library. URL: https://testing-library.com/docs/react-testing-library/intro/ (дата звернення: 04.10.2025).

13. SimScale. URL: https://www.simscale.com/ (дата звернення: 03.08.2025).

14. Sweigart A. The Big Book of Small Python Projects: 81 Easy Practice Programs. No Starch Press, 2021. 432 p.

15. TypeScript Documentation. URL: https://www.typescriptlang.org/docs/ (дата звернення: 15.09.2025).

16. Uvicorn. URL: https://uvicorn.dev/ (дата звернення: 16.09.2025).

17. Visual Paradigm. What is Unified Modeling Language (UML)? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/ (дата звернення: 07.05.2025).

18. Visual Paradigm. Use Case Diagram Guide. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/ (дата звернення: 07.05.2025).

19. Vite dev. URL: https://vite.dev/guide/ (дата звернення: 15.09.2025).

20. Wieruch R. The Road to React: Your journey to master plain yet pragmatic React.js. Independently published, 2025. 286 p.

ДОДАТОК А

Серверна частина

```
from __future__ import annotations

from typing import List, Optional, Dict, Any

import math
import numpy as np
import json
from fastapi import FastAPI, HTTPException, UploadFile, File, Form
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel, Field, ConfigDict, field_validator, ValidationError

class Node(BaseModel):
    x: float
    y: float

class Element(BaseModel):
    n1: int = Field(..., ge=0, description="Index of first node (0-based)")
    n2: int = Field(..., ge=0, description="Index of second node (0-based)")
    A: float = Field(..., gt=0, description="Cross-sectional area")
    E: float = Field(..., gt=0, description="Young's modulus")

    @field_validator("n2")
    @classmethod
    def nodes_must_differ(cls, v, info):
        if "n1" in info.data and v == info.data["n1"]:
            raise ValueError("n1 and n2 must refer to different nodes")
        return v

class Support(BaseModel):
    node: int = Field(..., ge=0)
    ux: bool = False
    uy: bool = False

class Load(BaseModel):
    node: int = Field(..., ge=0)
    fx: float = 0.0
    fy: float = 0.0

class SolveRequest(BaseModel):
```

```
    model_config = ConfigDict(extra="forbid")

    nodes: List[Node]
    elements: List[Element]
    supports: List[Support] = Field(default_factory=list)
    loads: List[Load] = Field(default_factory=list)
    scale: float = Field(1.0, gt=0, description="Scale factor for deformed shape coordinates")

    @field_validator("elements")
    @classmethod
    def validate_elements_nodes(cls, elements, info):
        nodes = info.data.get("nodes", [])
        nmax = len(nodes)
        for e in elements:
            if e.n1 >= nmax or e.n2 >= nmax:
                raise ValueError("Element references a node index that does not exist")
        return elements

class SolveResponse(BaseModel):
    displacements: List[List[float]]
    reactions: List[Optional[List[float]]]
    element_forces: List[float]
    deformed_nodes: List[List[float]]
    meta: Dict[str, Any]


app = FastAPI(
    title="FEA Minimal Server (2D Truss)",
    version="1.0.0",
    description="Minimal FastAPI server that solves small 2D truss problems for a React-based UI.",
)

origins = [
    "http://localhost:5173",
    "http://127.0.0.1:5173",
    "http://localhost:3000",
    "http://127.0.0.1:3000",
]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

```
def dof_index(node_index: int, comp: str) -> int:
    base = 2 * node_index
    return base if comp == "x" else base + 1


def assemble_global_stiffness(n_nodes: int, elements: List[Element], nodes: List[Node]) -> np.ndarray:
    K = np.zeros((2 * n_nodes, 2 * n_nodes), dtype=float)
    for e in elements:
        x1, y1 = nodes[e.n1].x, nodes[e.n1].y
        x2, y2 = nodes[e.n2].x, nodes[e.n2].y
        L = math.hypot(x2 - x1, y2 - y1)
        if L <= 0:
            raise ValueError("Zero-length element detected")
        c = (x2 - x1) / L
        s = (y2 - y1) / L
        k_local = (e.A * e.E / L) * np.array(
            [
                [ c*c,  c*s, -c*c, -c*s],
                [ c*s,  s*s, -c*s, -s*s],
                [-c*c, -c*s,  c*c,  c*s],
                [-c*s, -s*s,  c*s,  s*s],
            ],
            dtype=float,
        )

        dofs = [
            dof_index(e.n1, "x"),
            dof_index(e.n1, "y"),
            dof_index(e.n2, "x"),
            dof_index(e.n2, "y"),
        ]

        for i in range(4):
            for j in range(4):
                K[dofs[i], dofs[j]] += k_local[i, j]
    return K


def build_force_vector(n_nodes: int, loads: List[Load]) -> np.ndarray:
    F = np.zeros(2 * n_nodes, dtype=float)
    for load in loads:
        F[dof_index(load.node, "x")] += load.fx
        F[dof_index(load.node, "y")] += load.fy
    return F
```

```
def apply_constraints(K: np.ndarray, F: np.ndarray, supports: List[Support], n_nodes: int):

    constrained = np.zeros(2 * n_nodes, dtype=bool)
    for s in supports:
        if s.ux:
            constrained[dof_index(s.node, "x")] = True
        if s.uy:
            constrained[dof_index(s.node, "y")] = True

    free = ~constrained

    Kff = K[np.ix_(free, free)]
    Kfc = K[np.ix_(free, constrained)]
    Kcf = K[np.ix_(constrained, free)]
    Kcc = K[np.ix_(constrained, constrained)]
    Ff = F[free]
    Fc = F[constrained]

    return free, constrained, Kff, Kfc, Kcf, Kcc, Ff, Fc


def solve_truss(request: SolveRequest) -> SolveResponse:
    n_nodes = len(request.nodes)
    if n_nodes == 0:
        raise HTTPException(status_code=400, detail="No nodes provided")
    if len(request.elements) == 0:
        raise HTTPException(status_code=400, detail="No elements provided")

    K = assemble_global_stiffness(n_nodes, request.elements, request.nodes)
    F = build_force_vector(n_nodes, request.loads)

    free, constrained, Kff, Kfc, Kcf, Kcc, Ff, Fc = apply_constraints(K, F, request.supports, n_nodes)

    try:
        uf = np.linalg.solve(Kff, Ff)
    except np.linalg.LinAlgError as exc:
        raise HTTPException(status_code=422, detail=f"System is singular or ill-conditioned: {exc}")


    u = np.zeros(2 * n_nodes, dtype=float)
    u[free] = uf

    R_full = K @ u - F

    # Pack outputs
    displacements = [[u[2 * i + 0], u[2 * i + 1]] for i in range(n_nodes)]
```

```
    reactions: List[Optional[List[float]]] = []
    for i in range(n_nodes):
        rx = R_full[2 * i + 0] if constrained[2 * i + 0] else None
        ry = R_full[2 * i + 1] if constrained[2 * i + 1] else None
        reactions.append(None if (rx is None and ry is None) else [rx if rx is not None else 0.0, ry if ry is not None else 0.0])

    # Element axial forces
    elem_forces: List[float] = []
    for e in request.elements:
        x1, y1 = request.nodes[e.n1].x, request.nodes[e.n1].y
        x2, y2 = request.nodes[e.n2].x, request.nodes[e.n2].y
        L = math.hypot(x2 - x1, y2 - y1)
        c = (x2 - x1) / L
        s = (y2 - y1) / L
        ue = np.array([
            u[dof_index(e.n1, "x")],
            u[dof_index(e.n1, "y")],
            u[dof_index(e.n2, "x")],
            u[dof_index(e.n2, "y")],
        ])
        # axial extension: [-c, -s, c, s] · ue
        axial_extension = np.array([-c, -s, c, s]) @ ue
        N = (e.E * e.A / L) * axial_extension  # positive = tension
        elem_forces.append(float(N))

    # Deformed shape
    scale = request.scale
    deformed_nodes = [
        [request.nodes[i].x + scale * displacements[i][0], request.nodes[i].y + scale * displacements[i][1]]
        for i in range(n_nodes)
    ]

    meta = {
        "n_nodes": n_nodes,
        "n_elements": len(request.elements),
        "sum_load": [float(np.sum(F[0::2])), float(np.sum(F[1::2]))],
        "units": {"length": "m", "force": "N", "modulus": "Pa", "area": "m^2"},
    }

    # Build reactions per node with None for completely free nodes (for cleaner UI binding)
    reactions_clean: List[Optional[List[float]]] = []
    for i in range(n_nodes):
        cx = constrained[2 * i + 0]
        cy = constrained[2 * i + 1]
```

```
        if not (cx or cy):
            reactions_clean.append(None)
        else:
            rx = R_full[2 * i + 0] if cx else 0.0
            ry = R_full[2 * i + 1] if cy else 0.0
            reactions_clean.append([float(rx), float(ry)])

    return SolveResponse(
        displacements=[[float(a), float(b)] for a, b in displacements],
        reactions=reactions_clean,
        element_forces=[float(N) for N in elem_forces],
        deformed_nodes=[[float(a), float(b)] for a, b in deformed_nodes],
        meta=meta,
    )

@app.get("/health")
async def health():
    return {"status": "ok"}

@app.get("/api/v1/examples/cantilever", response_model=SolveRequest)
async def example_cantilever() -> SolveRequest:
    nodes = [Node(x=0.0, y=0.0), Node(x=1.0, y=0.0), Node(x=1.0, y=1.0)]
    elements = [
        Element(n1=0, n2=1, A=1e-4, E=210e9),
        Element(n1=1, n2=2, A=1e-4, E=210e9),
    ]
    supports = [Support(node=0, ux=True, uy=True)]
    loads = [Load(node=2, fx=0.0, fy=-1000.0)]  # 1000 N downward
    return SolveRequest(nodes=nodes, elements=elements, supports=supports, loads=loads,
scale=1000.0)


@app.post("/api/v1/solve/truss2d/upload", response_model=SolveResponse)
async def solve_upload(file: UploadFile = File(...), scale: Optional[float] = Form(None)) ->
SolveResponse:
    try:
        raw = await file.read()
        try:
            data = json.loads(raw.decode("utf-8"))
        except Exception as e:
            raise HTTPException(status_code=400, detail=f"Invalid JSON: {e}")

        req = SolveRequest.model_validate(data)
        if scale is not None:
            req.scale = float(scale)
        return solve_truss(req)
```

```
    except HTTPException:
        raise
    except ValidationError as ve:
        raise HTTPException(status_code=422, detail=f"Validation error: {ve}")
    except Exception as ex:  # pragma: no cover
        raise HTTPException(status_code=500, detail=str(ex))

@app.post("/api/v1/solve/truss2d", response_model=SolveResponse)
async def solve_endpoint(payload: SolveRequest) -> SolveResponse:
    try:
        return solve_truss(payload)
    except HTTPException:
        raise
    except Exception as ex:  # pragma: no cover
        raise HTTPException(status_code=500, detail=str(ex))


@app.get("/")
async def root():
    return {"message": "FEA Minimal Server is running. See /docs for OpenAPI UI."}


if __name__ == "__main__":
    import uvicorn

    uvicorn.run("main:app", host="0.0.0.0", port=8000, reload=True)
```

ДОДАТОК Б

React fea.ts

```
import type {SolveRequest, SolveResponse} from "../types/fea";

export async function apiGetExample(baseUrl: string): Promise<SolveRequest> {
   const r = await fetch(`${baseUrl}/api/v1/examples/cantilever`);
   if (!r.ok) throw new Error(`Example fetch failed: ${r.status}`);
   return r.json();
}
export async function apiSolve(baseUrl: string, body: SolveRequest): Promise<SolveResponse> {
   const r = await fetch(`${baseUrl}/api/v1/solve/truss2d`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(body),
   });
   if (!r.ok) throw new Error(`Solve failed: ${r.status}`);
   return r.json();
}
export async function apiUpload(
   baseUrl: string,
   file: File,
   overrideScale?: number
): Promise<SolveResponse> {
   const fd = new FormData();
   fd.append("file", file, file.name);
   if (overrideScale !== undefined && overrideScale !== null && !Number.isNaN(overrideScale))
{
      fd.append("scale", String(overrideScale));
   }
   const r = await fetch(`${baseUrl}/api/v1/solve/truss2d/upload`, {
      method: "POST",
      body: fd,
   });
   if (!r.ok) throw new Error(`Upload failed: ${r.status}`);
   return r.json();
}
```

ДОДАТОК В

React Home.tsx

```
import {useMemo, useRef, useState} from "react";
import {
  Button,
  Spinner,
  Input,
  Textarea,
  TabList,
  Tab,
  MessageBar,
  MessageBarBody,
  MessageBarTitle,
  Switch,
  Field
} from "@fluentui/react-components";
import {apiGetExample, apiSolve, apiUpload} from "../api/fea";
import type {SolveRequest, SolveResponse} from "../types/fea";
import {pretty} from "../utils/format";
import {Panel} from "../components/Panel";
import {DataTable} from "../components/DataTable";
import {Viewer2D} from "../components/Viewer2D";
import {ModelForm} from "../components/ModelForm";

export default function Home() {
  const [baseUrl, setBaseUrl] = useState("http://localhost:8000");
  const [tab, setTab] = useState("form");

  const ex = "{\n" +
    "  \"nodes\": [\n" +
    "    {\"x\": 0, \"y\": 0},\n" +
    "    {\"x\": 1, \"y\": 0},\n" +
    "    {\"x\": 1, \"y\": 1}\n" +
    "  ],\n" +
    "  \"elements\": [\n" +
    "    {\"n1\": 0, \"n2\": 1, \"A\": 1e-4, \"E\": 210000000000},\n" +
    "    {\"n1\": 1, \"n2\": 2, \"A\": 1e-4, \"E\": 210000000000},\n" +
    "    {\"n1\": 0, \"n2\": 2, \"A\": 1e-4, \"E\": 210000000000}\n" +
    "  ],\n" +
    "  \"supports\": [\n" +
    "    {\"node\": 0, \"ux\": true, \"uy\": true},\n" +
```

```
    "    {\"node\": 1, \"ux\": false, \"uy\": true}\n" +
    "  ],\n" +
    "  \"loads\": [\n" +
    "    {\"node\": 2, \"fx\": 0, \"fy\": -1000}\n" +
    "  ],\n" +
    "  \"scale\": 1000\n" +
    "}\n";

const [jsonText, setJsonText] = useState(ex);

const [overrideScale, setOverrideScale] = useState<string>("");
const fileInput = useRef<HTMLInputElement | null>(null);
const [uploadFile, setUploadFile] = useState<File | null>(null); // ← нове

const [busy, setBusy] = useState(false);
const [error, setError] = useState<string | null>(null);
const [req, setReq] = useState<SolveRequest | null>(null);
const [res, setRes] = useState<SolveResponse | null>(null);

const handleGetExample = async () => {
    setError(null);
    setBusy(true);
    try {
        const example = await apiGetExample(baseUrl);
        setReq(example);
        setJsonText(JSON.stringify(example, null, 2));
        setTab("editor");
    } catch (e: any) {
        setError(e.message || String(e));
    } finally {
        setBusy(false);
    }
};

const handleSolveJson = async () => {
    setError(null);
    setBusy(true);
    try {
        const parsed = JSON.parse(jsonText) as SolveRequest;
        const r = await apiSolve(baseUrl, parsed);
        setReq(parsed);
        setRes(r);
    } catch (e: any) {
        if (e instanceof SyntaxError) setError("Некоректний JSON у полі редактора.");
        else setError(e.message || String(e));
    } finally {
```

```
        setBusy(false);
    }
};

const onFilePicked = (f: File | undefined) => {
    setUploadFile(f ?? null);
};

const handleUploadSubmit = async () => {
    if (!uploadFile) {
        setError("Оберіть .json файл з моделлю.");
        return;
    }
    setError(null);
    setBusy(true);
    try {
        const scale = overrideScale.trim() ? Number(overrideScale) : undefined;

        const text = await uploadFile.text();
        const parsed = JSON.parse(text) as SolveRequest;
        if (scale !== undefined) parsed.scale = scale;
        setReq(parsed);

        const r = await apiUpload(baseUrl, uploadFile, scale);
        setRes(r);
    } catch (e: any) {
        setError(e.message || String(e));
    } finally {
        setBusy(false);
        setUploadFile(null);
        if (fileInput.current) fileInput.current.value = "";
    }
};

const handleSolveFromForm = async (model: SolveRequest) => {
    setError(null);
    setBusy(true);
    try {
        const r = await apiSolve(baseUrl, model);
        setReq(model);
        setRes(r);
    } catch (e: any) {
        setError(e.message || String(e));
    } finally {
        setBusy(false);
    }
```

```
  };

  const nodesOrig = useMemo(
    () => (req ? (req.nodes.map((n) => [n.x, n.y]) as [number, number][]) : []),
    [req]
  );
  const nodesDef = useMemo(
    () => (res ? (res.deformed_nodes.map((p) => [p[0], p[1]]) as [number, number][]) : undefined),
    [res]
  );

  const [showOriginal, setShowOriginal] = useState(true);
  const [showDeformed, setShowDeformed] = useState(true);

  return (
    <div className="app">
      <div className="stack">
        <h1>Вебінтерфейс системи скінченно-елементного аналізу із використанням
бібліотеки React</h1>
        <Panel title="Дані">
          <div style={{ display: "flex", flexDirection: "column", gap: 12 }}>
            <Field label="Адреса сервера">
              <Input value={baseUrl} onChange={(_, v) => setBaseUrl(v.value)} size="large"
/>
            </Field>

            <TabList selectedValue={tab} onTabSelect={(_, d) => setTab(String(d.value))}>
              <Tab id="form" value="form">Форма</Tab>
              <Tab id="upload" value="upload">Завантаження файлу</Tab>
              <Tab id="editor" value="editor">JSON-редактор</Tab>
            </TabList>

            {tab === "form" && (
              <ModelForm onSubmit={handleSolveFromForm} busy={busy} />
            )}

            {tab === "upload" && (
              <div style={{ display: "grid", gap: 12 }}>
                <Field label="Файл моделі (.json)">
                  <input
                    ref={fileInput}
                    type="file"
                    accept="application/json,.json"
                    onChange={(e) => onFilePicked(e.currentTarget.files?.[0])}
                  />
                </Field>
```

```
                    <Field label="Перевизначити scale (опційно)">
                      <Input
                        placeholder="напр. 1000"
                        value={overrideScale}
                        onChange={(_, v) => setOverrideScale(v.value)}
                      />
                    </Field>
                    <div style={{ display: "flex", gap: 8 }}>
                      <Button
                        appearance="primary"
                        onClick={handleUploadSubmit}
                        disabled={busy || !uploadFile}
                      >
                        Розрахувати
                      </Button>
                      {uploadFile && <span style={{ alignSelf: "center"
}}>{uploadFile.name}</span>}
                    </div>
                  </div>
                )}

                {tab === "editor" && (
                  <div style={{ display: "grid", gap: 8, gridTemplateRows: "1fr auto" }}>
                    <Textarea
                      resize="vertical"
                      style={{
                        minHeight: 260,
                        fontFamily:
                          "ui-monospace, SFMono-Regular, Menlo, Monaco, Consolas,
'Liberation Mono', 'Courier New', monospace",
                      }}
                      value={jsonText}
                      onChange={(_, d) => setJsonText(d.value)}
                    />
                    <div
                      style={{
                        display: "flex",
                        gap: 8,
                        alignItems: "center",
                        justifyContent: "space-between",
                        flexWrap: "wrap",
                      }}
                    >
                      <div style={{ display: "flex", gap: 8 }}>
                        <Button appearance="secondary" onClick={handleGetExample}
disabled={busy}>
```

```
                        Приклад
                      </Button>
                      <Button
                        appearance="secondary"
                        onClick={() => setJsonText((t) => JSON.stringify(JSON.parse(t), null,
2))}
                      >
                        Форматувати JSON
                      </Button>
                      <Button appearance="primary" onClick={handleSolveJson}
disabled={busy}>
                        Розрахувати
                      </Button>
                    </div>
                  </div>
                </div>
              )}

              {busy && (
                <div style={{ display: "flex", alignItems: "center", gap: 8 }}>
                  <Spinner size="small" /> Обробка запиту…
                </div>
              )}

              {error && (
                <MessageBar intent="error">
                  <MessageBarBody>
                    <MessageBarTitle>Помилка</MessageBarTitle>
                    {error}
                  </MessageBarBody>
                </MessageBar>
              )}
            </div>
          </Panel>
          <Panel title="Результати">
            {!res && (
              <div style={{ color: "var(--colorNeutralForeground3)" }}>
                Немає даних
              </div>
            )}
            {res && (
              <div style={{ display: "grid", gap: 16 }}>
                <div>
                  <h4 style={{ margin: "8px 0" }}>Переміщення вузлів</h4>
                  <DataTable
                    columns={["#", "ux", "uy"]}
```

```
                rows={res.displacements.map((d, i) => [i, pretty(d[0]), pretty(d[1])])}
              />
            </div>

            <div>
              <h4 style={{ margin: "8px 0" }}>Реакції в опорах (лише закріплені)</h4>
              <DataTable
                columns={["#", "Rx", "Ry"]}
                rows={
                  res.reactions
                    .map((r, i) => (r ? [i, pretty(r[0]), pretty(r[1])] : null))
                    .filter(Boolean) as (string | number)[][]
                }
              />
            </div>
            <div>
              <h4 style={{ margin: "8px 0" }}>Осьові сили в елементах (N)</h4>
              <DataTable
                columns={["елемент", "N (тяга + / стиск -)"]}
                rows={res.element_forces.map((N, i) => [i, pretty(N)])}
              />
            </div>

            <div>
              <h4 style={{ margin: "8px 0" }}>Мета</h4>
              <DataTable
                columns={["параметр", "значення"]}
                rows={Object.entries(res.meta).map(([k, v]) => [
                  k,
                  typeof v === "object" ? JSON.stringify(v) : String(v),
                ])}
              />
            </div>
          </div>
        )}
      </Panel>
      <Panel
        title="Візуалізація"
        right={
          <div style={{ display: "flex", gap: 12, alignItems: "center" }}>
            <Switch
              label="Оригінальна"
              checked={showOriginal}
              onChange={(_, d) => setShowOriginal(!!d.checked)}
            />
            <Switch
```

```
                        label="Деформована"
                        checked={!!showDeformed}
                        onChange={(_, d) => setShowDeformed(!!d.checked)}
                      />
                    </div>
                  }
                >
                  {(!req || !res) && (
                    <div style={{ color: "var(--colorNeutralForeground3)" }}>
                      Для відображення введіть модель у формі / JSON або завантажте файл і виконайте розрахунок
                    </div>
                  )}
                  {req && (
                    <Viewer2D
                      nodes={nodesOrig}
                      deformed={nodesDef}
                      elements={req.elements.map((e) => ({ n1: e.n1, n2: e.n2 }))}
                      showOriginal={showOriginal}
                      showDeformed={showDeformed}
                    />
                  )}
                </Panel>
              </div>
            </div>
          );
        }
```

ДОДАТОК Г

React Viewer2D

```
import * as React from "react";
import { useMemo, useLayoutEffect, useRef, useState } from "react";
import { fitPoints } from "../utils/geometry";

export const Viewer2D: React.FC<{
  nodes: [number, number][];
  deformed?: [number, number][];
  elements: { n1: number; n2: number }[];
  height?: number;
  showOriginal?: boolean;
  showDeformed?: boolean;
}> = ({ nodes, deformed, elements, height, showOriginal = true, showDeformed = true }) => {
  const wrapRef = useRef<HTMLDivElement | null>(null);
  const [w, setW] = useState(800);

  useLayoutEffect(() => {
    if (!wrapRef.current) return;
    const ro = new ResizeObserver(entries => {
      const cr = entries[0].contentRect;
      setW(Math.max(320, cr.width));
    });
    ro.observe(wrapRef.current);
    return () => ro.disconnect();
  }, []);

  const allPts = useMemo(() => {
    const orig = nodes as [number, number][];
    const def = (deformed ?? []) as [number, number][];
    return def.length ? orig.concat(def) : orig;
  }, [nodes, deformed]);

  const h = height ?? Math.round(Math.min(560, Math.max(280, w * 0.65)));

  const { sx, sy, tx, ty } = useMemo(() => fitPoints(allPts, w, h, 24), [allPts, w, h]);
  const mapPt = (p: [number, number]) => ({ x: tx + sx * p[0], y: ty + sy * p[1] });

  return (
    <div ref={wrapRef}>
      <svg className="responsive" height={h}>
```

```
            {showOriginal && (
              <g stroke="var(--colorNeutralForeground3)" fill="none">
                {elements.map((e, i) => {
                  const p1 = mapPt(nodes[e.n1]);
                  const p2 = mapPt(nodes[e.n2]);
                  return <line key={i} x1={p1.x} y1={p1.y} x2={p2.x} y2={p2.y}
strokeWidth={2} />;
                })}
                {nodes.map((p, i) => {
                  const m = mapPt(p);
                  return <circle key={i} cx={m.x} cy={m.y} r={3} fill="var(--
colorNeutralForeground3)" />;
                })}
              </g>
            )}

            {showDeformed && deformed && (
              <g stroke="var(--colorBrandForegroundLink)" fill="none">
                {elements.map((e, i) => {
                  const p1 = mapPt(deformed[e.n1]);
                  const p2 = mapPt(deformed[e.n2]);
                  return <line key={i} x1={p1.x} y1={p1.y} x2={p2.x} y2={p2.y}
strokeWidth={3} />;
                })}
                {deformed.map((p, i) => {
                  const m = mapPt(p);
                  return <circle key={i} cx={m.x} cy={m.y} r={3.5} fill="var(--
colorBrandForegroundLink)" />;
                })}
              </g>
            )}
          </svg>
        </div>
    );
};
```