Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №3
Технології розроблення програмного забезпечення
«Основи проектування розгортання.»

Тема: Редактор зображень

Виконала:

студентка групи ІА-32

Ейсмонт А.В.

Перевірив:

Мягкий М.Ю.

Київ 2025

Зміст

Тема: Основи проектування розгортання.

Мета: Навчитися проєктувати діаграми розгортання та компонентів для системи що проєктується, а також розробляти діаграми взаємодії, а саме діаграми послідовностей, на основі сценаріїв зроблених в попередній лабораторній роботі.

Завдання:
•Ознайомитись з короткими теоретичними відомостями.

• Проаналізувати діаграми створені в попередній лабораторній роботі а також тему системи та спроєктувати діаграму розгортання використання відповідно до обраної теми лабораторного циклу.

• Розробити діаграму компонентів для проєктованої системи.

• Розробити діаграму розгортання для проєктованої системи.

• Розробити як мінімум дві діаграми послідовностей для сценаріїв прописаних в попередній лабораторній роботі.

• На основі спроєктованих діаграм розгортання та компонентів доопрацювати програмну частину системи. Реалізація системи, додатково до попередньої реалізації, повинна містити як мінімум дві візуальні форми. В системі вже повинен бути повністю реалізована архітектура (повний цикл роботи з даними від вводу на формі до збереження їх в БД і подальшій виборці з БД та відображенням на UI).

• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму розгортання з описом, діаграму компонентів системи з описом, діаграми послідовностей, а також вихідний код системи, який було додано в цій лабораторній роботі.

Теоретичні відомості:

Діаграма розгортання

Діаграми розгортання представляють фізичне розташування системи, показуючи, на якому фізичному обладнанні запускається та чи інша складова програмного забезпечення [3]. Головними елементами діаграми є вузли, пов'язані інформаційними шляхами. Вузол (node) – це те, що може містити програмне забезпечення. Вузли бувають двох типів. Пристрій (device) – це фізичне обладнання: комп'ютер або пристрій, пов'язаний із системою. Середовище виконання (execution environment) – це програмне забезпечення, яке саме може включати інше програмне забезпечення, наприклад операційну систему або процес-контейнер (наприклад, вебсервер). Між вузлами можуть стояти зв'язки, які зазвичай зображують у вигляді прямої лінії. Як і на інших діаграмах, у зв'язків можуть бути атрибути множинності (для показання, наприклад, підключення 2х і більше клієнтів до одного сервера) і назва. У назві, як правило, міститься спосіб зв'язку між двома вузлами – це може бути назва протоколу (HTTP, IPC) або технологія, що використовується для забезпечення взаємодії вузлів (.NET Remoting, WCF). Вузли можуть містити артефакти (artifacts), які є фізичним уособленням програмного забезпечення; зазвичай це файли. Такими файлами можуть бути виконувані файли (такі як файли .exe, двійкові файли, файли DLL, файли JAR, збірки або сценарії) або файли даних, конфігураційні файли, HTML-документи тощо. Перелік артефактів усередині вузла вказує на те, що на даному вузлі артефакт розгортається в систему, що запускається. Артефакти можна зображати у вигляді прямокутників класів або перераховувати їхні імена всередині вузла. Якщо ви показуєте ці елементи у вигляді прямокутників класів, то можете додати значок документа або ключове слово «artifact». Можна супроводжувати вузли або артефакти значеннями у вигляді міток, щоб вказати різну цікаву інформацію про вузол, наприклад постачальника, операційну систему, місце розташування – загалом, усе, що спаде вам на думку. Часто у вас буде безліч фізичних вузлів для розв'язання однієї й тієї самої логічної задачі. Можна відобразити цей факт, намалювавши безліч прямокутників вузлів або поставивши число у вигляді значення-мітки.

Артефакти часто є реалізацією компонентів. Це можна показати, задавши значення-мітки всередині прямокутників артефактів. Основні види артефактів: • вихідні файли; • виконувані файли; • сценарії; • таблиці баз даних; • документи; • результати процесу розробки, UML-моделі. Можна також деталізувати артефакти, що входять до вузла; наприклад, додатково всередині файлу що розгортується вказати, які туди входять компоненти або класи. Така деталізація, як правило, не має сенсу на діаграмах розгортання, оскільки може зміщувати фокус уваги від моделі розгортання програмного забезпечення до його внутрішнього устрою, проте іноді може бути корисною. При цьому, можливо встановлювати зв'язки між компонентами/класами в межах різних вузлів. Діаграми розгортань розрізняють двох видів: описові та екземплярні. На діаграмах описової форми вказуються вузли, артефакти і зв'язки між вузлами без вказівки конкретного обладнання або програмного забезпечення, необхідного для розгортання. Такий вид діаграм корисний на ранніх етапах розроблення для розуміння, які взагалі фізичні пристрої необхідні для функціонування системи або для опису процесу розгортання в загальному ключі. Діаграми екземплярної форми несуть у собі екземпляри обладнання, артефактів і зв'язків між ними. Під екземплярами розуміють конкретні елементи – ПК із відповідним набором характеристик і встановленим ПЗ; цілком може бути, у межах однієї організації це може бути якийсь конкретний вузол (наприклад, ПК тестувальника Василя). Діаграми екземплярної форми розробляють на завершальних стадіях розроблення ПЗ – коли вже відомі та сформульовані вимоги до програмного комплексу, обладнання закуплено і все готово до розгортання. Діаграми такої форми являють собою скоріше план розгортання в графічному вигляді, ніж модель розгортання.

Діаграма компонентів UML

Діаграма компонентів UML є представленням проєктованої системи, розбитої на окремі модулі [3]. Залежно від способу поділу на модулі розрізняють три види діаграм компонентів: • логічні; • фізичні; • виконувані. Коли використовують логічне розбиття на компоненти, то у такому разі проєктовану систему віртуально уявляють як набір самостійних, автономних модулів (компонентів),

що взаємодіють між собою. 42 Коли на діаграмі представляють фізичне розбиття, то в такому разі на діаграмі компонентів показують компоненти та залежності між ними. Залежності показують, що класи в з одного компонента використовують класи з іншого компонента. Фізична модель використовується для розуміння які компоненти повинні бути зібрані в інсталяційний пакет. Також така діаграма показує зміни в якому компоненті будуть впливати на інші компоненти. Приклад такої діаграми наведено на рисунку 3.2. У цьому випадку на діаграмі показани фізичні файли (.exe та .dll), показана залежність між ними, а також всі файли робиті на три блоки: компоненти для серверної частини, компоненти для клієнтської та компоненти, умовно названі middleware, які повинні бути як на серверній так і на клієнтській стороні, тому що в них або є загальна бізнес-логіка, або є загальні інтерфейси методів та даних якими обмінюються між собою кліент та сервер. Компоненти можуть поділятися за фізичними одиницями – окремі вузли розподіленої системи – набір комп'ютерів і серверів; на кожному з вузлів можуть бути встановлені різні виконувані компоненти. Такий вид діаграм компонентів застарів і зазвичай замість нього використовують діаграму розгортань. На діаграмах компонентів з виконуваним поділом компонентів кожен компонент являє собою деякий файл – виконувані файли (.exe), файли вихідних кодів, сторінки html, бази даних і таблиці тощо. У цьому разі діаграма схожа на діаграму класів, але на більш верхньому рівні – рівні виконуваних файлів або процесів. Такий підхід дає інший розріз представлення системи. Діаграма компонентів розробляється для таких цілей: • візуалізації загальної структури вихідного коду програмної системи; • специфікації виконуваного варіанта програмної системи; • забезпечення багаторазового використання окремих фрагментів програмного коду; • представлення концептуальної та фізичної схем баз даних.

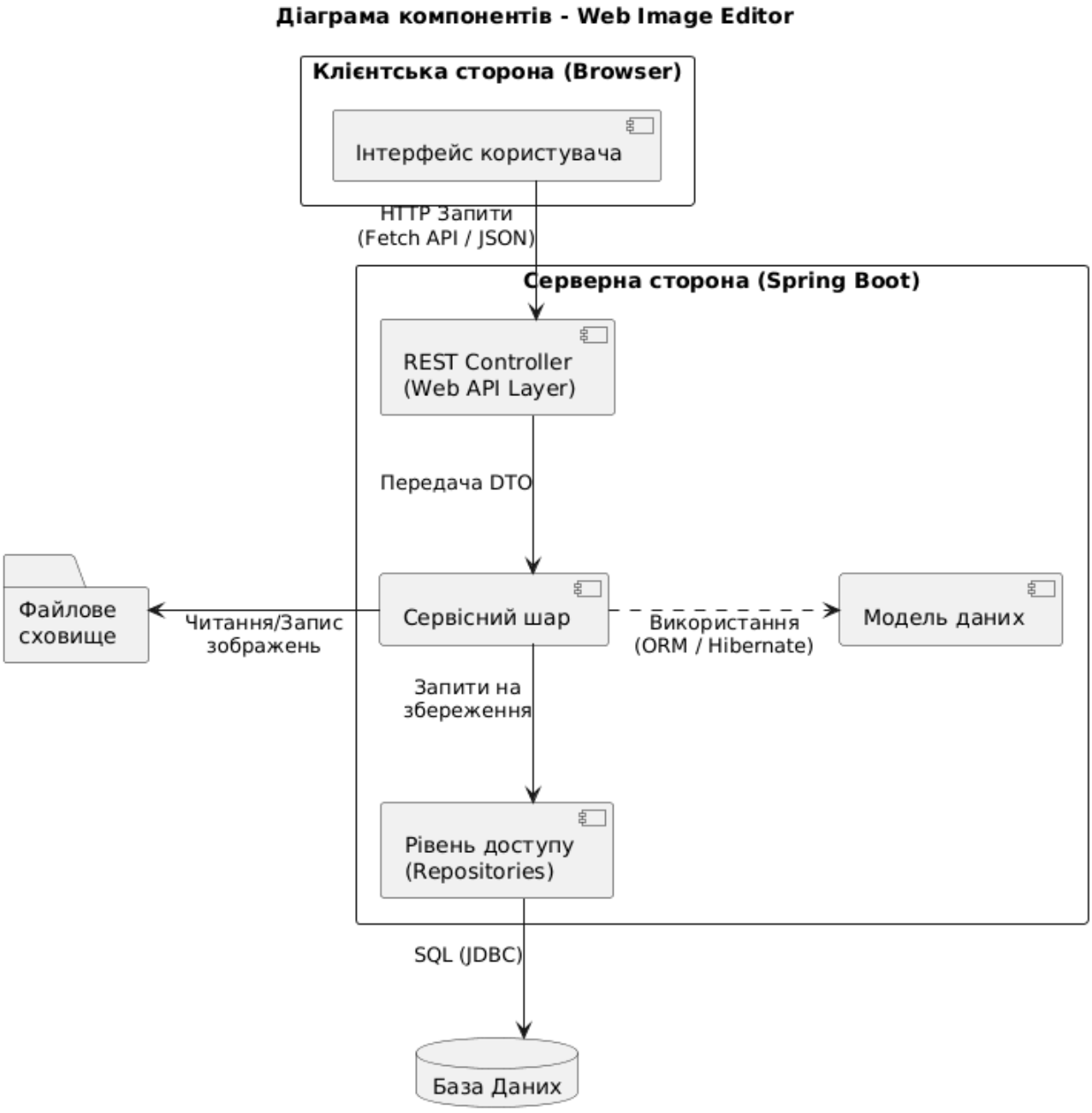
Діаграма послідовностей

Діаграма послідовностей (Sequence Diagram) – це один із типів діаграм у моделюванні UML (Unified Modeling Language), який використовується для моделювання взаємодії між об'єктами системи у певній послідовності часу. Вона відображає, як об'єкти обмінюються повідомленнями, показуючи порядок і

логіку виконання операцій. Діаграма складається з таких основних елементів: Актори (Actors): Зазвичай позначаються піктограмами або назвами. Це користувачі чи інші системи, які взаємодіють із системою. Актори можуть бути зовнішніми стосовно моделювання системи. Об'єкти або класи: Розміщуються горизонтально на діаграмі. Вони позначаються прямокутниками з іменем об'єкта або класу під прямокутником. Кожен об'єкт має «життєвий цикл», який представлений вертикальною пунктирною лінією (лінія життя). Повідомлення: Це лінії зі стрілками, які з'єднують об'єкти. Вони показують передачу повідомлень чи виклик методів. Стрілка може бути синхронною (звичайна стрілка) або асинхронною (лінія з відкритим трикутником) та з пунктирною лінією, що показує повернення результату. Активності: Вказують періоди, протягом яких об'єкт виконує певну дію. На діаграмі це позначається прямокутником, накладеним на лінію життя. Контрольні структури: Використовуються для відображення умов, циклів або альтернативних сценаріїв. Наприклад, блоки "alt" (альтернатива) або "loop" (цикл). Основні кроки створення діаграми послідовностей: • визначити акторів і об'єкти, які беруть участь у сценарії; • побудувати їхні лінії життя; • розробити послідовність передачі повідомлень між об'єктами; • додати умовні блоки або цикли за необхідності. Діаграми послідовностей є корисними для моделювання бізнес-процесів, проєктування архітектури систем і тестування. Вони дають змогу візуалізувати логіку взаємодії компонентів та виявити потенційні проблеми ще на етапі проєктування.

Хід роботи:

1. Діаграма компонентів



Основні компоненти:

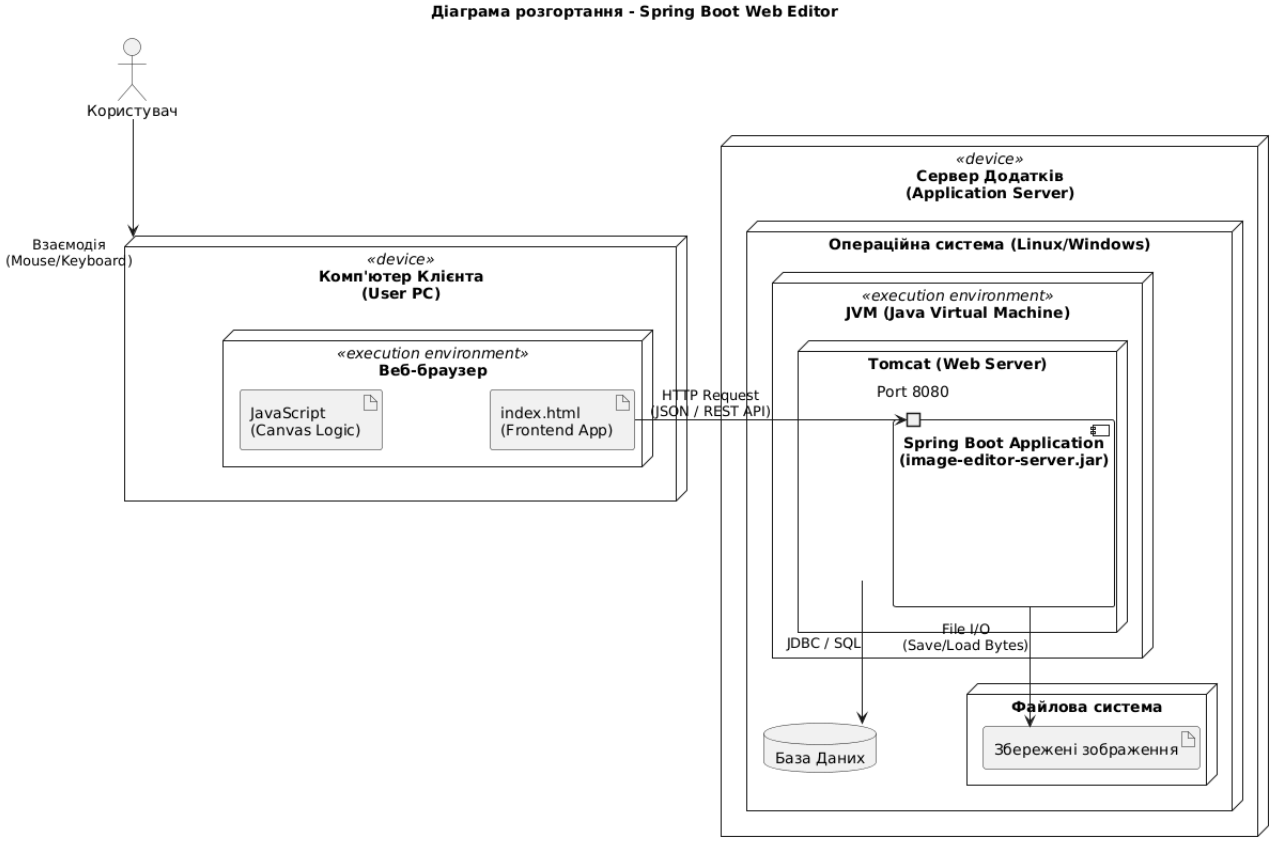
- Інтерфейс користувача: Клієнтська частина, що працює у браузері. Відповідає за візуалізацію та відправку HTTP-запитів.
- REST Controller: Вхідна точка на сервері. Приймає JSON-дані від клієнта та маршрутизує їх до бізнес-логіки.
- Сервісний шар: Містить основну логіку програми (реєстрація, обробка зображень).
- Модель даних (Entities): Класи, що відображають таблиці бази даних (User, Project, Layer).

- Рівень доступу (Repositories): Інтерфейси Spring Data JPA для виконання операцій з базою даних без написання SQL вручну.
- Файлове сховище: Директорія на сервері для зберігання фізичних файлів зображень.
- База Даних: Зберігає інформацію про користувачів та структуру проєктів.

Взаємодії:

1. Клієнт відправляє дані через HTTP на контролер.
2. Контролер викликає методи сервісного шару.
3. Сервіс маніпулює сутностями та зберігає файли у файлове сховище.
4. Для збереження метаданих сервіс звертається до репозиторіїв, які виконують SQL-запити до бази даних.

2. Діаграма розгортання



Фізичні вузли:

- Комп'ютер Клієнта (User PC) — пристрій кінцевого користувача (ноутбук, планшет або стаціонарний ПК). Не вимагає встановлення спеціального програмного забезпечення, оскільки взаємодія відбувається через браузер.
    - Веб-браузер (Execution Environment): середовище виконання клієнтського коду. Відповідає за рендеринг графічного інтерфейсу та виконання скриптів.

    - index.html (Frontend App): Артефакт, що завантажується з сервера. Містить розмітку сторінки та JavaScript-код для роботи з HTML5 Canvas (малювання, обробка подій миші).

- Сервер Додатків (Application Server) — обчислювальний вузол, де розгорнуто серверну частину системи.
    - JVM (Java Virtual Machine): Віртуальна машина Java, що забезпечує запуск серверного додатка.
    - Spring Boot Application (Tomcat): Головний виконуваний компонент (JAR-архів). Містить вбудований веб-сервер Tomcat для обробки HTTP-запитів, REST-контролери та бізнес-логіку редактора.
    - База Даних: Реляційна база даних, що працює в межах сервера. Зберігає облікові записи користувачів та метадані проєктів.
    - Файлова система: Використовується сервером для фізичного збереження бінарних файлів зображень, які завантажують користувачі.

Актор:

Користувач — взаємодіє з системою через браузер, використовуючи стандартні пристрої введення (миша, клавіатура).

Зв'язки між вузлами:

- HTTP Request (JSON / REST API): Основний канал зв'язку між Клієнтом і Сервером. Клієнт надсилає команди (наприклад, збереження малюнка) у форматі JSON через мережу Інтернет або локальну мережу.
- JDBC (Java Database Connectivity): Внутрішній протокол взаємодії Java-додатка з базою даних для виконання SQL-запитів.
- File I/O: Прямий доступ сервера до локального диска для запису та зчитування файлів зображень.

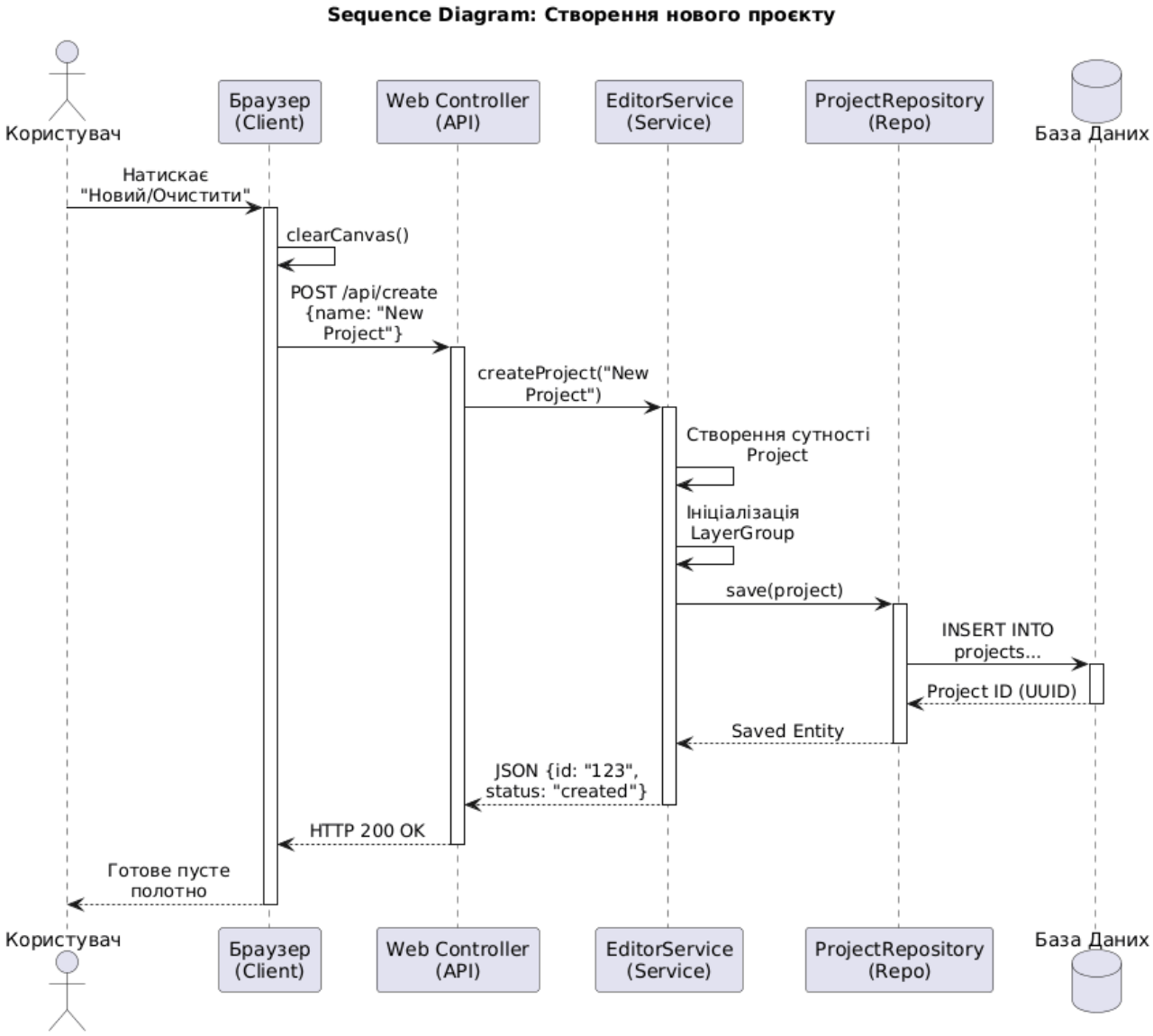
Взаємодія компонентів:

- Користувач малює у браузері; JavaScript конвертує малюнок у формат Base64.
- Браузер формує HTTP POST-запит і відправляє його на порт 8080 сервера.
- Spring Boot Application приймає запит, декодує зображення і зберігає файл у файлову систему.
- Інформація про новий проєкт записується у Базу Даних через JDBC.

– Сервер повертає статус успішного виконання (HTTP 200 OK) назад у браузер.

3. Діаграми послідовностей

Створення нового проєкту

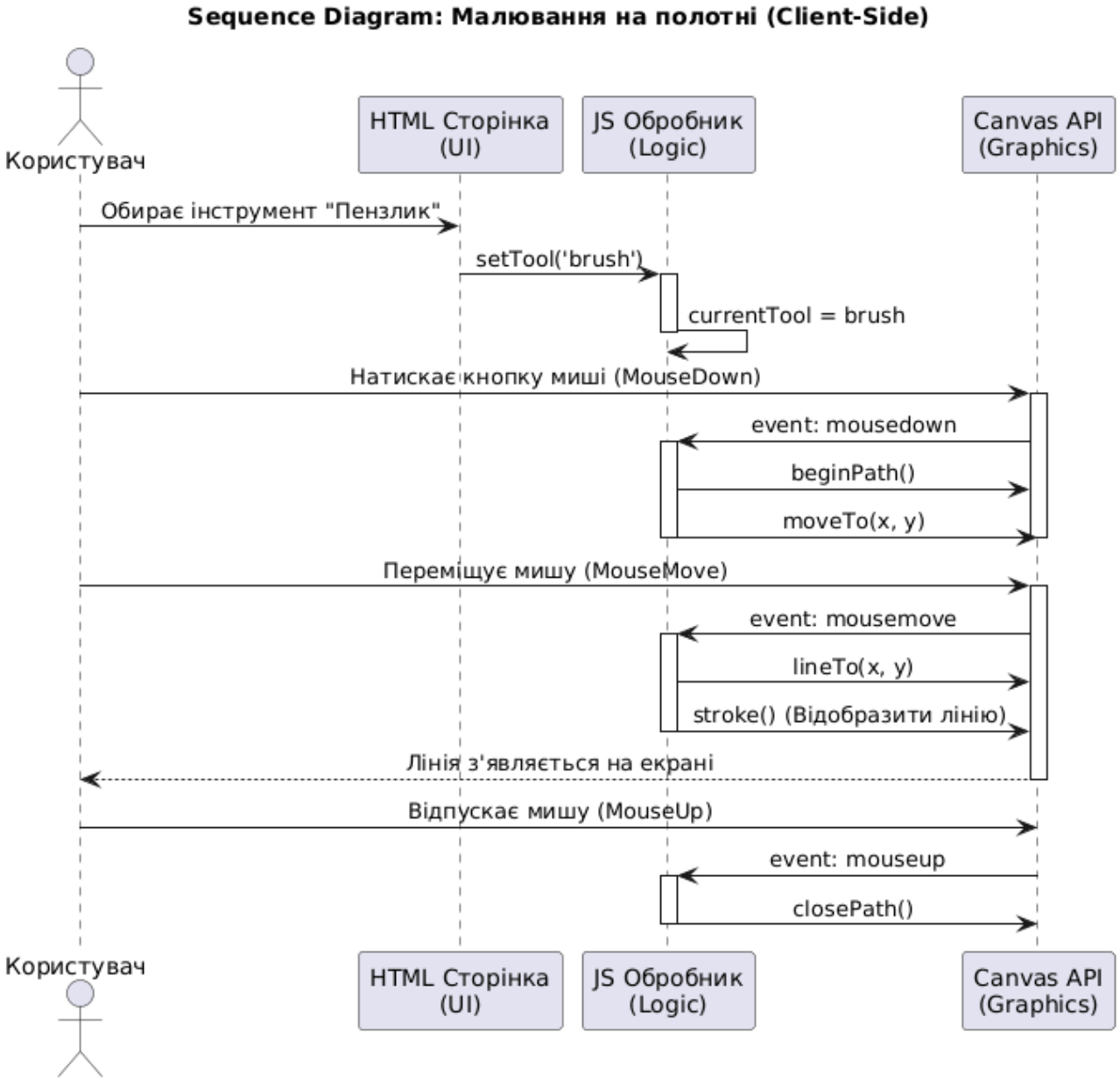


Створення проєкту

1. Користувач натискає кнопку «Очистити» або «Новий проєкт» у браузері.
2. JavaScript клієнта очищає полотно (canvas) і відправляє запит POST /api/create на сервер для реєстрації нового сеансу роботи.
3. Web Controller приймає запит і передає його в EditorService.
4. Сервіс створює нову сутність Project та ініціалізує кореневу групу шарів (LayerGroup).
5. Через ProjectRepository дані записуються в Базу Даних.

6. Сервер повертає ID створеного проєкту.
7. Клієнт отримує підтвердження і готовий до малювання.

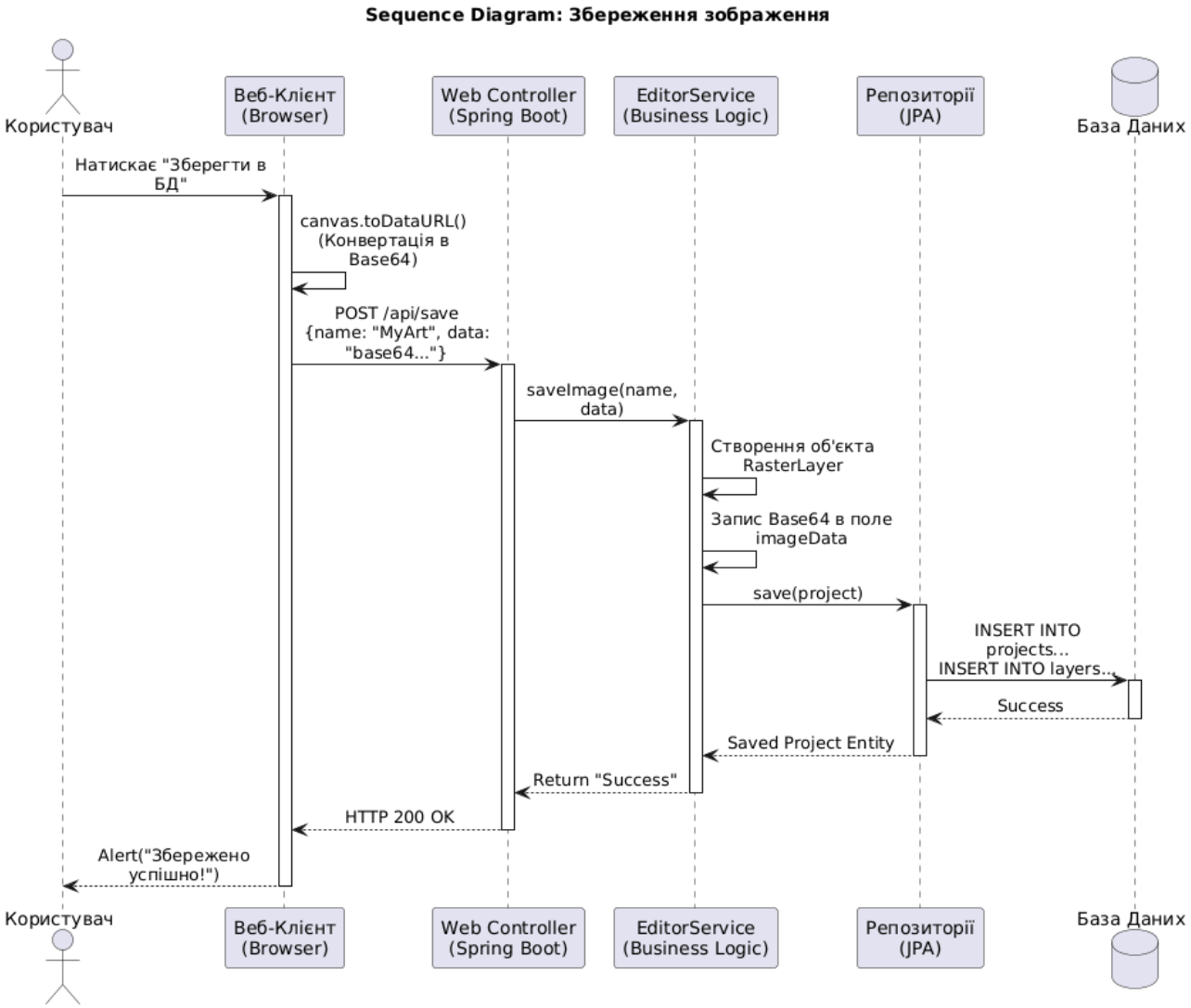
Малювання (Редагування)



Малювання інструментом

1. Користувач обирає інструмент на панелі (HTML).
2. JS Обробник встановлює активний стан інструменту (зміна кольору, товщини).
3. Користувач натискає кнопку миші на полотні. Браузер генерує подію mousedown.
4. Скрипт починає новий графічний шлях (beginPath).
5. Під час руху миші (mousemove) скрипт отримує координати курсору і малює лінії за допомогою HTML5 Canvas API.
6. Зображення оновлюється миттєво на екрані користувача без звернення до сервера.

Збереження на сервер



1. Користувач натискає кнопку «Зберегти».
2. Браузер конвертує пікселі полотна у текстовий рядок формату Base64 (метод toDataURL).
3. Клієнт формує JSON-об'єкт і відправляє його через HTTP POST на ендпоінт /api/save.
4. Web Controller приймає дані та передає їх у EditorService.
5. Сервіс створює сутність шару (RasterLayer), записуючи туди отримані дані зображення.
6. Через Repository відбувається каскадне збереження проєкту та шарів у Базу Даних.
7. Сервер повертає статус успішного виконання, і клієнт показує повідомлення користувачеві.

Винятки:

- Помилка мережі: Якщо сервер недоступний, клієнт (JS) ловить помилку (catch) і виводить повідомлення «Server Error».
- Помилка БД: Якщо дані занадто великі або порушена цілісність, транзакція відкочується, і сервер повертає код помилки 500.

4. Питання до лабораторної роботи:

1. Що собою становить діаграма розгортання?

це UML-діаграма, яка показує, як програмні компоненти розміщені на фізичних або віртуальних вузлах (напр. сервери, ПК, мобільні пристрої) та як вони між собою взаємодіють

2. Які бувають види вузлів на діаграмі розгортання?

Є апаратні (напр. сервер, клієнтський комп'ютер, телефон) та програмні (операційні системи, сервери додатків, бази даних)

3. Які бувають зв'язки на діаграмі розгортання?

На діаграмі розгортання бувають асоціативні та комунікаційні зв'язки, що показують обмін даними між вузлами

4. Які елементи присутні на діаграмі компонентів?

На діаграмі компонентів є самі компоненти, інтерфейси, порти, пакети та залежності між ними

5. Що становлять собою зв'язки на діаграмі компонентів?

Зв'язки на діаграмі компонентів відображають залежність одного компонента від іншого

6. Які бувають види діаграм взаємодії?

Діаграма послідовностей, діаграма комунікацій, діаграма таймінгу та діаграма огляду взаємодії

7. Для чого призначена діаграма послідовностей?

Діаграма послідовностей потрібна, щоб показати порядок викликів методів і обмін повідомленнями між об'єктами у часі

8. Які ключові елементи можуть бути на діаграмі послідовностей?

Об'єкти(тробіж учасники), їх життєві лінії, повідомлення (синхронні чи асинхронні виклики), активації та блоки управління

9. Як діаграми послідовностей пов'язані з діаграмами варіантів використання?

Кожен сценарій з діаграми use case можна деталізувати у вигляді діаграми послідовностей, щоб показати кроки виконання

10. Як діаграми послідовностей пов’язані з діаграмами класів?

Класи, що описані у діаграмі класів, стають об’єктами у діаграмі послідовностей, і на ній видно, як вони взаємодіють між собою

Вихідний код: https://github.com/eismonta/TRPZ_2025/tree/main/image-editor

Висновок:

Протягом виконання лабораторної роботи було спроєктовано діаграму компонентів, діаграму розгортання та діаграми послідовностей, після чого доопрацьовано програмну частину системи: реалізовано дві візуальні форми, забезпечено повний цикл роботи з даними