Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота № 3

з дисципліни «Технології розроблення програмного забезпечення»

Тема лабораторної роботи: «Основи проектування розгортання»

Тема проєкта: «Аудіо редактор»

Виконала:
студентка групи ІА-33
Котик Іванна

Перевірив:
асистент кафедри ІСТ
Мягкий Михайло Юрійович

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.

Тема проєкта: 5. Аудіо редактор (singleton, adapter, observer, mediator, composite, client server). Аудіо редактор повинен володіти наступним функціоналом: представлення аудіо даних будь-якого формату в WAVE-формі, вибір і подальші операції копіювання / вставки / вирізання / деформації по сегменту аудіозапису, можливість роботи з декількома звуковими доріжками, кодування в найбільш поширених форматах (ogg, flac, mp3).

Короткі теоретичні відомості:

Діаграма розгортання (Deployment Diagram)

Діаграми розгортання представляють фізичне розташування системи, показуючи, на якому фізичному обладнанні запускається та чи інша складова програмного забезпечення.

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).

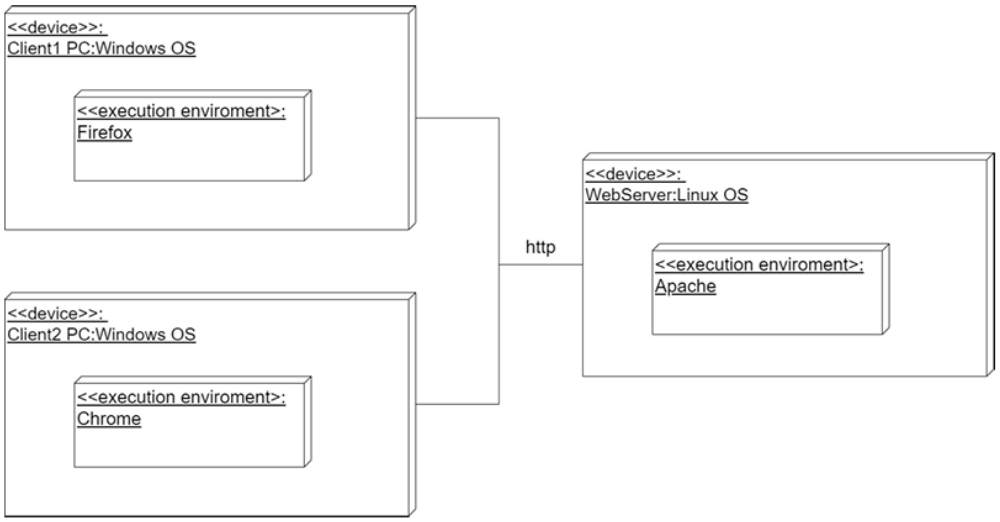


Рисунок 1 – Діаграма розгортання системи

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

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.

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 2.

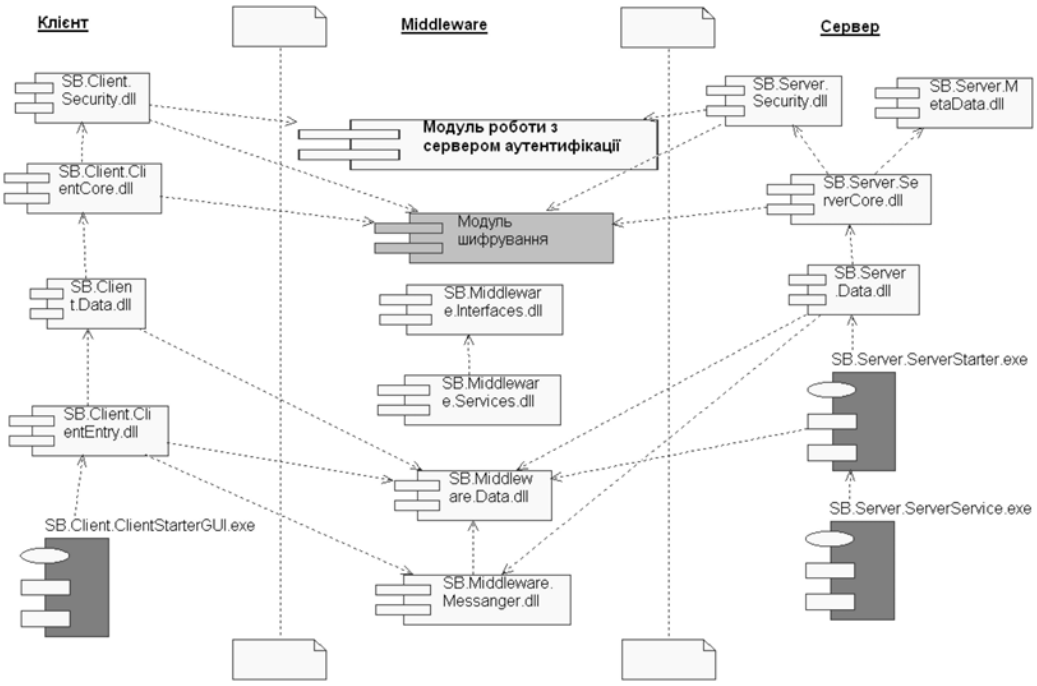


Рисунок 2 – Приклад діаграми компонентів

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 клієнт 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.

Діаграми 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). Приклад діаграми послідовності зображений на рисунку 3.

Рисунок 3 – Діаграма послідовності «Запит остатку на рахунку»

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. Побудова діаграми розгортання.

Опис діаграми розгортання.

Діаграма розгортання відображає фізичне розташування програмних компонентів системи «Аудіоредактор» на апаратних вузлах та зв’язки між ними.

Система реалізована за клієнт-серверною архітектурою. На клієнтському ПК розгортається артефакт AudioEditorClient.jar, який містить користувацький інтерфейс (відображення аудіо у вигляді хвильової форми, таймлайну, списку доріжок). Для синхронізації елементів інтерфейсу застосовано патерни Observer та Mediator.

На Application Server розгортається артефакт AudioEditorServer.war, що містить серверну частину програми. Основною точкою взаємодії між клієнтом і сервером є REST API, через який передаються запити користувача. Усередині серверного застосунку реалізовані компоненти-сервіси:

- AuthService – автентифікація та авторизація користувачів;
- ProjectService – створення, відкриття та збереження проєктів;
- TrackService – управління звуковими доріжками (Composite);
- SegmentService – операції з сегментами (копіювання, вирізання, вставка, деформація);

- ConverterService – кодування та декодування аудіо у формати WAV, MP3, FLAC, OGG (Adapter).

Для доступу до бази даних використовується шаблон Repository, що забезпечує відокремлення бізнес-логіки від шару збереження даних. У серверному артефакті виділено компоненти UserRepository, ProjectRepository, TrackRepository, SegmentRepository.

На окремому вузлі розташований Database Server, де виконується СУБД (PostgreSQL/MySQL) та розгорнутий артефакт AudioEditorDB, що зберігає дані про користувачів, проєкти, доріжки та сегменти.

Крім того, для зберігання фізичних аудіофайлів використовується File Storage Server, де розташовані файли у форматах WAV, MP3, FLAC, OGG. Серверна частина взаємодіє з файловим сховищем через операції File I/O.

Таким чином, діаграма відображає:

- вузли системи (Client PC, Application Server, Database Server, File Storage Server);
- артефакти, що на них розгортаються (.jar, .war, база даних, аудіофайли);
- основні сервіси і репозиторії всередині серверного застосунку;
- зв’язки між компонентами (HTTPS/REST, JDBC, File I/O).

Архітектура відповідає вимогам завдання та демонструє застосування патернів: Singleton (ядро редактора), Observer та Mediator (клієнтський інтерфейс), Composite (управління доріжками), Adapter (кодування аудіо), Client–Server (загальна структура) і Repository (доступ до БД).

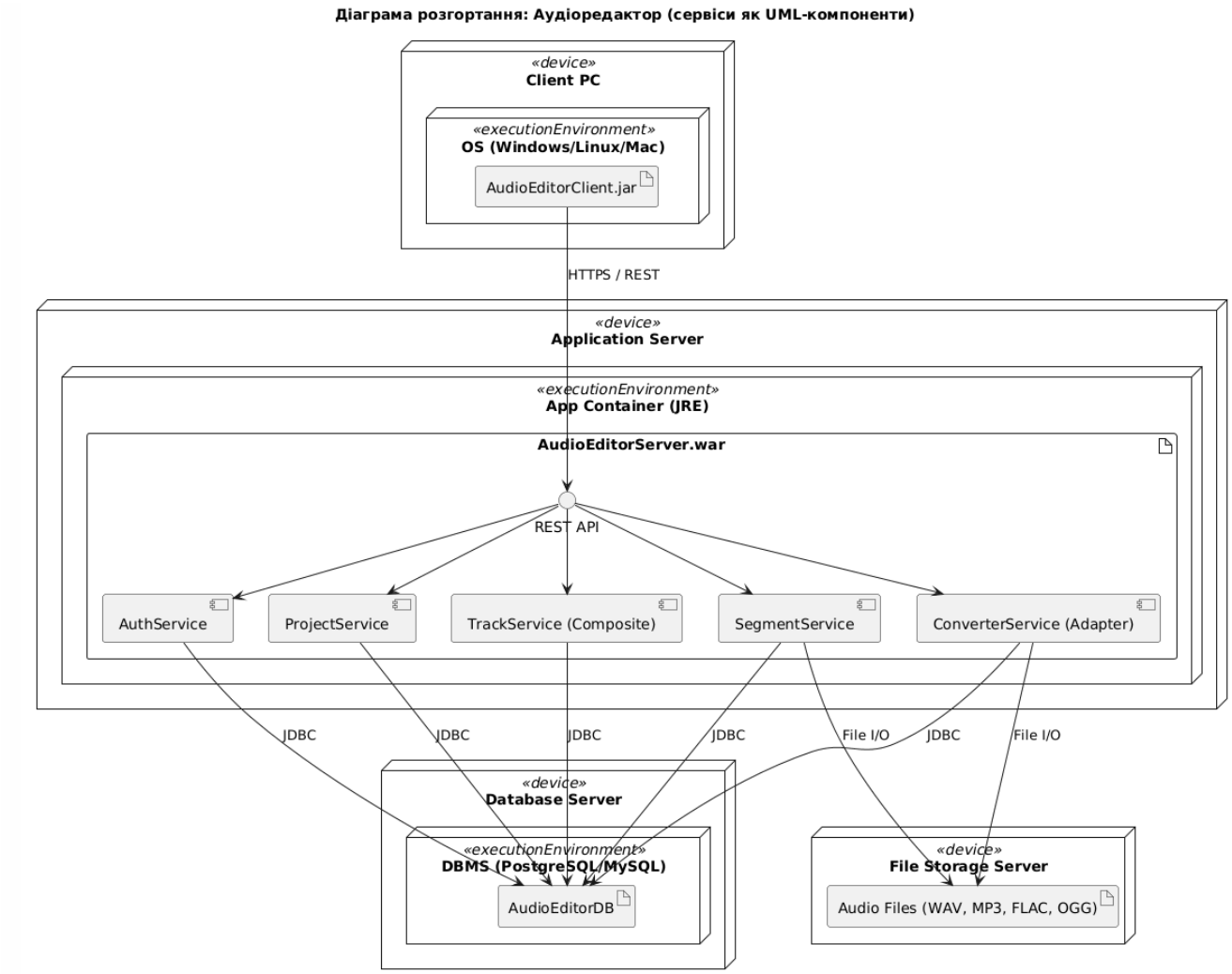


Рисунок 1 – Діаграма розгортання

2. Розробка діаграми компонентів для проєктованої системи.

Діаграма компонентів

На діаграмі відображена структура системи «Аудіоредактор» у вигляді основних компонентів, їхніх інтерфейсів та взаємозв’язків. Вона демонструє реалізовану частину програмного забезпечення, поділену на клієнтську, серверну частину, шар доступу до даних та зовнішні сервіси.

1. Клієнтська частина

Клієнт представлений артефактом AudioEditorClient.jar, який містить компонент REST Client. Завдяки цьому клієнт здійснює запити до серверної частини через протокол HTTP/REST та відображає результати користувачу.

2. Серверна частина (SOA)

Серверна логіка організована за принципами сервісно-орієнтованої архітектури. Вона включає:

- REST API – точку доступу до всіх серверних сервісів.
- AuthService (реалізує інтерфейс *IAuthAPI*) – відповідає за автентифікацію та управління користувачами.
- ProjectService (реалізує *IProjectAPI*) – керує створенням і редагуванням проєктів.
- TrackService (реалізує *ITrackAPI*, патерн *Composite*) – забезпечує роботу з багатодоріжковою структурою.
- SegmentService (реалізує *ISegmentAPI*) – виконує операції над сегментами (копіювання, вирізання, вставка, деформація).
- ConverterService (реалізує *IConverterAPI*, патерн *Adapter*) – здійснює перетворення аудіо у формати WAV, MP3, FLAC, OGG.

3. Шар доступу до даних (Repository Layer)

В системі застосовано шаблон Repository, який ізолює бізнес-логіку від деталей роботи з базою даних.

- Інтерфейс IRepository<T> описує базові методи доступу до даних.
- Конкретні реалізації: *UserRepository*, *ProjectRepository*, *TrackRepository*, *SegmentRepository*.

Усі вони інкапсульовані в єдиний пакет Repository Layer, який має прямий зв’язок із базою даних.

4. База даних

AudioEditorDB (PostgreSQL/MySQL) – реляційна база даних, у якій зберігається інформація про користувачів, проєкти, доріжки та сегменти.

5. Файлове сховище

Окремий вузол File Storage використовується для зберігання аудіофайлів у форматах WAV, MP3, FLAC, OGG.

- SegmentService здійснює читання та запис сегментів у файлах.
- ConverterService забезпечує імпорт/експорт у вибрані формати.

6. Зовнішні сервіси

Для кодування та декодування аудіофайлів застосовується артефакт FFmpeg / Codecs, який використовується *ConverterService* як зовнішній сервіс.

Таким чином, діаграма компонентів показує логічну структуру системи «Аудіоредактор», відображає модульність та інтеграцію з базою даних, файловим сховищем і зовнішніми сервісами, а також демонструє застосування ключових шаблонів проєктування.

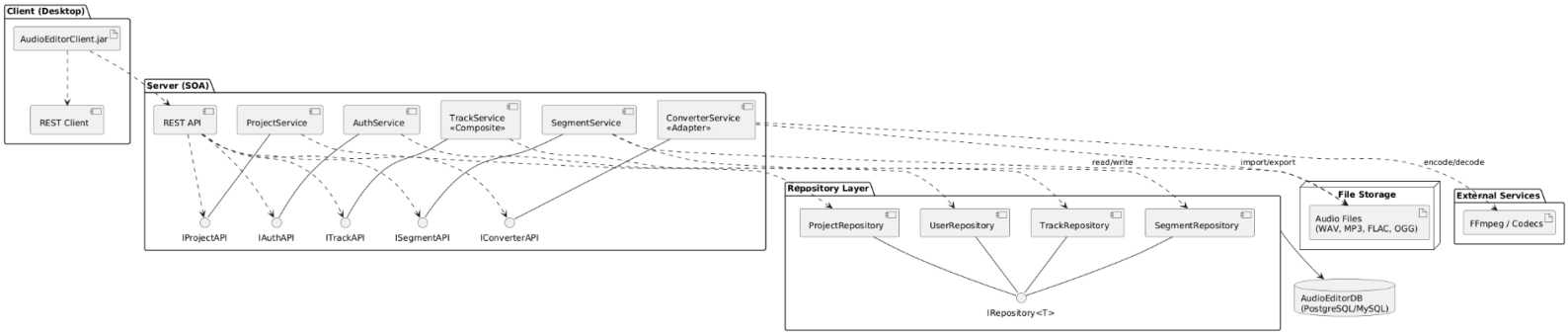


Рисунок 2 – Діаграма компонентів

3. Розробимо діаграму розгортання для проєктованої системи.

Опис діаграми розгортання

Діаграма розгортання демонструє фізичне розташування компонентів системи «Аудіоредактор» на апаратних вузлах та зв’язки між ними. Архітектура побудована за клієнт–серверною моделлю з урахуванням масштабованості та застосування основних шаблонів проєктування.

1. Клієнтський вузол

На Client PC розгортається артефакт AudioEditorClient.jar, що виконується у середовищі JRE. Він забезпечує роботу користувацького інтерфейсу: відображення хвильової форми аудіо, списку доріжок і таймлайну. Для синхронізації елементів інтерфейсу використано патерни Observer та Mediator. Взаємодія з сервером здійснюється через протокол HTTPS (порт 443).

2. Балансувальник навантаження

Load Balancer приймає вхідні запити клієнтів та розподіляє їх між екземплярами Application Server, що забезпечує масштабованість і стійкість системи.

3. Сервер застосунку

На Application Server (App Server xN) у середовищі JRE розгортається артефакт AudioEditorServer.war, що містить серверну логіку.
Основні компоненти серверного застосунку:

- REST API – головна точка взаємодії з клієнтом (Client–Server).
- AuthService – автентифікація та авторизація користувачів.
- ProjectService – створення та управління проєктами.
- TrackService – робота з багатодоріжковою структурою (Composite).

- SegmentService – операції над сегментами (копіювання, вставка, вирізання, деформація).
- ConverterService – кодування та декодування аудіо (Adapter).

Для ізоляції бізнес-логіки від роботи з базою даних застосовано Repository Layer, що включає: UserRepository, ProjectRepository, TrackRepository, SegmentRepository. Це реалізує патерн Repository.

4. Сервер бази даних

На Database Server працює DBMS (PostgreSQL/MySQL), де розгорнутий артефакт AudioEditorDB. Сервер застосунку взаємодіє з базою даних через JDBC (порт 5432/3306). Зберігаються дані про користувачів, проєкти, доріжки та сегменти.

5. Файлове сховище

На File Storage Server розташовані фізичні аудіофайли у форматах WAV, MP3, FLAC, OGG. Сервер застосунку взаємодіє з ними через операції File I/O (read/write).

6. Зовнішній сервіс

Для забезпечення конвертації аудіофайлів використовується External Service – FFmpeg/Codecs, що виконує операції encode/decode.

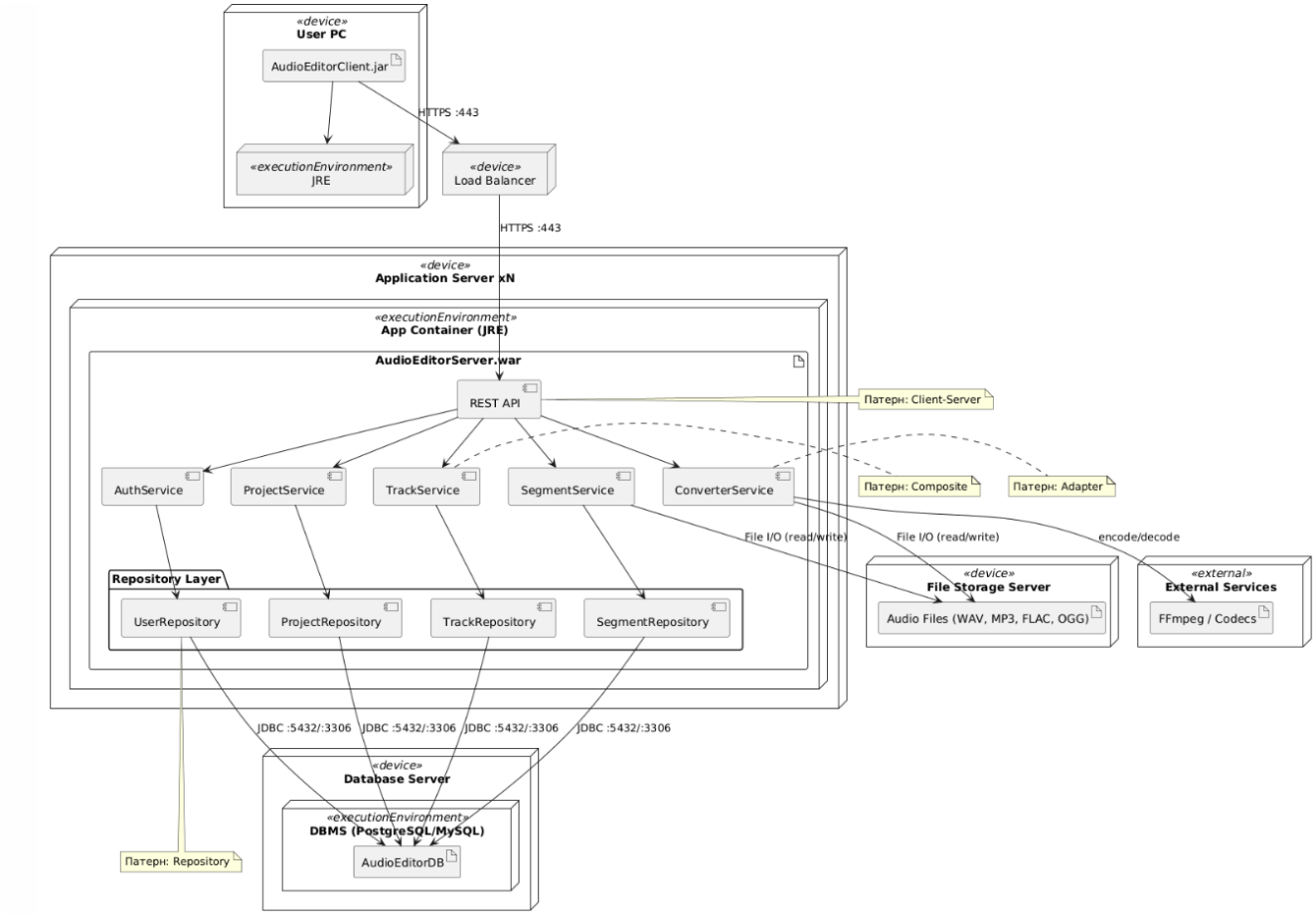


Рисунок 3 – Діаграма реалізованої частини системи

4. Розробка діаграм послідовностей.

4.1. Діаграма послідовностей «Видалення аудіосегмента».

Дана діаграма описує сценарій видалення окремого аудіосегмента у системі «Аудіоредактор». Вона демонструє взаємодію користувача з клієнтським застосунком, серверними сервісами, базою даних і файловим сховищем, а також обробку виняткової ситуації, коли сегмент не існує.

Учасники:

- User – користувач, який ініціює видалення сегмента.
- UI (Client) – клієнтський застосунок, у якому користувач взаємодіє з інтерфейсом.

- REST API – серверний інтерфейс, що приймає запити від клієнта.
- ProjectService – координує роботу із сервісами проєкту, перевіряє коректність запиту.
- TrackService – перевіряє наявність доріжки, до якої належить сегмент.
- SegmentService – безпосередньо відповідає за сегменти: отримання, перевірку та видалення.
- Repository Layer – шар доступу до бази даних (реалізує патерн Repository).
- AudioEditorDB – база даних, де зберігаються метадані про проєкти, доріжки та сегменти.
- File Storage – файлове сховище, де зберігаються фізичні аудіофайли сегментів.

Основний сценарій (успіх):

1. User обирає сегмент у клієнтському інтерфейсі та натискає *«Видалити»*.
2. UI формує HTTP-запит DELETE /projects/{id}/tracks/{tid}/segments/{sid} і надсилає його на REST API.
3. REST API викликає метод deleteSegment у ProjectService.
4. ProjectService звертається до TrackService для перевірки, чи існує доріжка. TrackService через Repository Layer отримує дані з DB і підтверджує валідність доріжки.
5. ProjectService звертається до SegmentService, який також через Repository Layer отримує метадані сегмента (зокрема шлях до файлу у сховищі).
6. Якщо сегмент існує:
    - SegmentService викликає File Storage для видалення файлу;

- після підтвердження файл видалено, сервіс видаляє запис у AudioEditorDB;
- повертається підтвердження про успішне видалення;
- користувачу надсилається відповідь 200 OK із повідомленням «Сегмент видалено», а сегмент зникає з інтерфейсу.

Винятковий сценарій (сегмент не існує):

- Якщо у БД немає запису про сегмент:
  - Repository Layer повертає помилку NotFound;
  - SegmentService повідомляє ProjectService про помилку;
  - ProjectService формує відповідь ErrorResponse(404, "Segment not found");
  - користувач у інтерфейсі отримує повідомлення «Сегмент не існує».

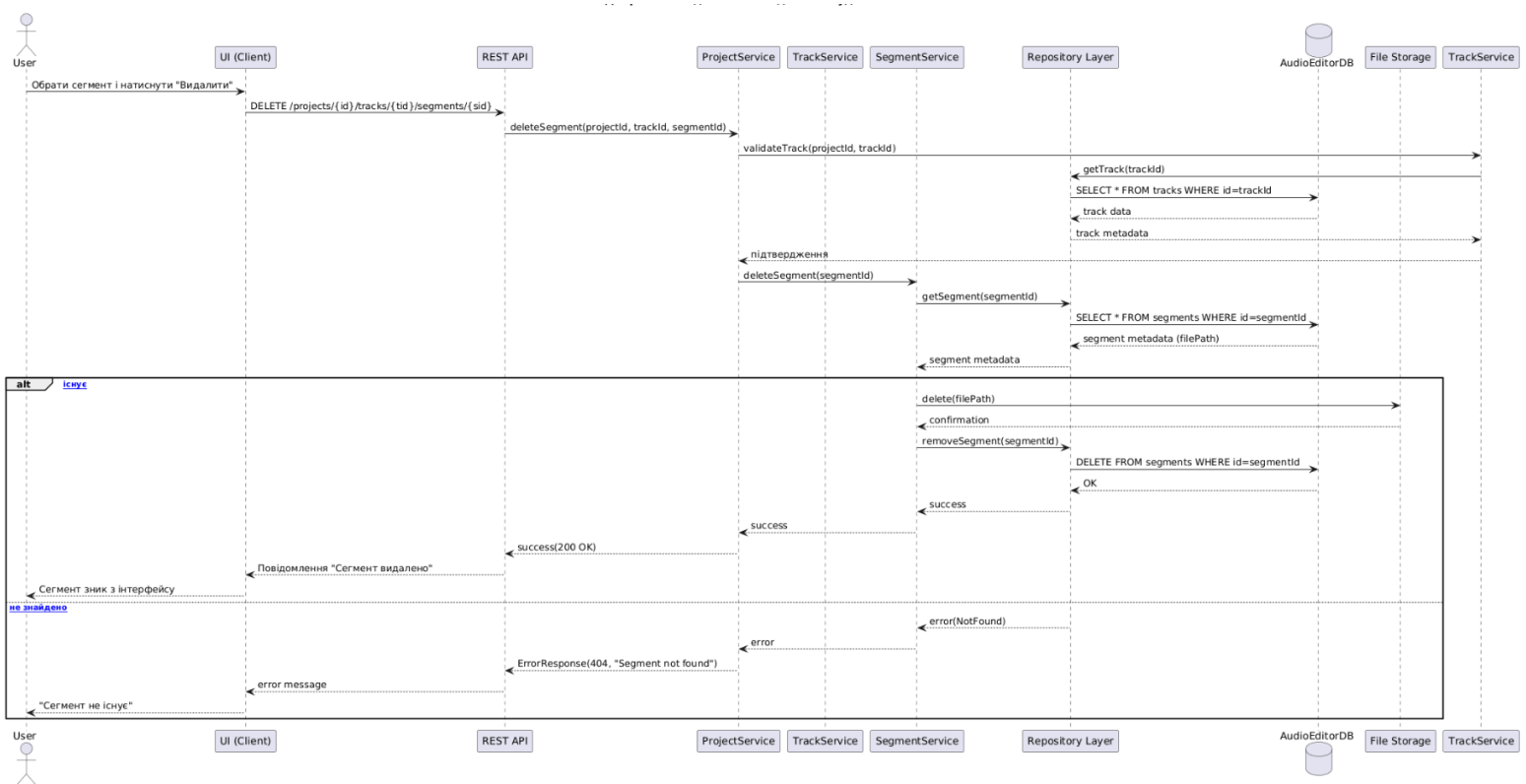


Рисунок 4.2 – Діаграма послідовностей «Видалення аудіосегмента»

4.2. Діаграма послідовностей «Відтворення аудіо»

Діаграма відображає процес відтворення аудіопроєкту у системі «Аудіоредактор» з урахуванням стандартного сценарію та виняткової ситуації, коли сегмент аудіофайлу відсутній у файловому сховищі.

Основний сценарій (успішне відтворення)

1. Користувач у графічному інтерфейсі (UI) обирає дію «Відтворити».
2. UI надсилає запит до REST API: GET /projects/{id}/play.
3. REST API передає запит у ProjectService для запуску відтворення проєкту.
4. ProjectService звертається до Repository Layer для отримання метаданих проєкту.
    - Виконується SQL-запит до AudioEditorDB, звідки повертаються дані про проєкт.
5. ProjectService звертається до TrackService для отримання списку доріжок.
    - TrackService отримує метадані доріжок через репозиторій з бази даних.
6. Для кожної доріжки викликається SegmentService, який отримує список сегментів з бази даних.
7. SegmentService звертається до File Storage для читання файлів аудіосегментів.
    - Файли сегментів успішно зчитуються та повертаються у вигляді аудіо-даних.
8. SegmentService об’єднує сегменти у трек і передає дані у TrackService, а той — у ProjectService.
9. ProjectService передає готовий аудіопотік через REST API у AudioPlayer.

10. AudioPlayer розпочинає відтворення й повідомляє про статус «playing».
11. Користувач отримує повідомлення «Відтворення розпочато», і звук починає програватися.

Виняткова ситуація (файл відсутній або пошкоджений)

1. Під час запиту до File Storage для отримання сегмента повертається помилка FileNotFound.
2. SegmentService передає помилку у TrackService, а далі — у ProjectService.
3. ProjectService формує відповідь ErrorResponse (404, "Файл сегмента відсутній").
4. REST API повертає повідомлення у UI.
5. Користувач бачить повідомлення «Неможливо відтворити — файл відсутній».

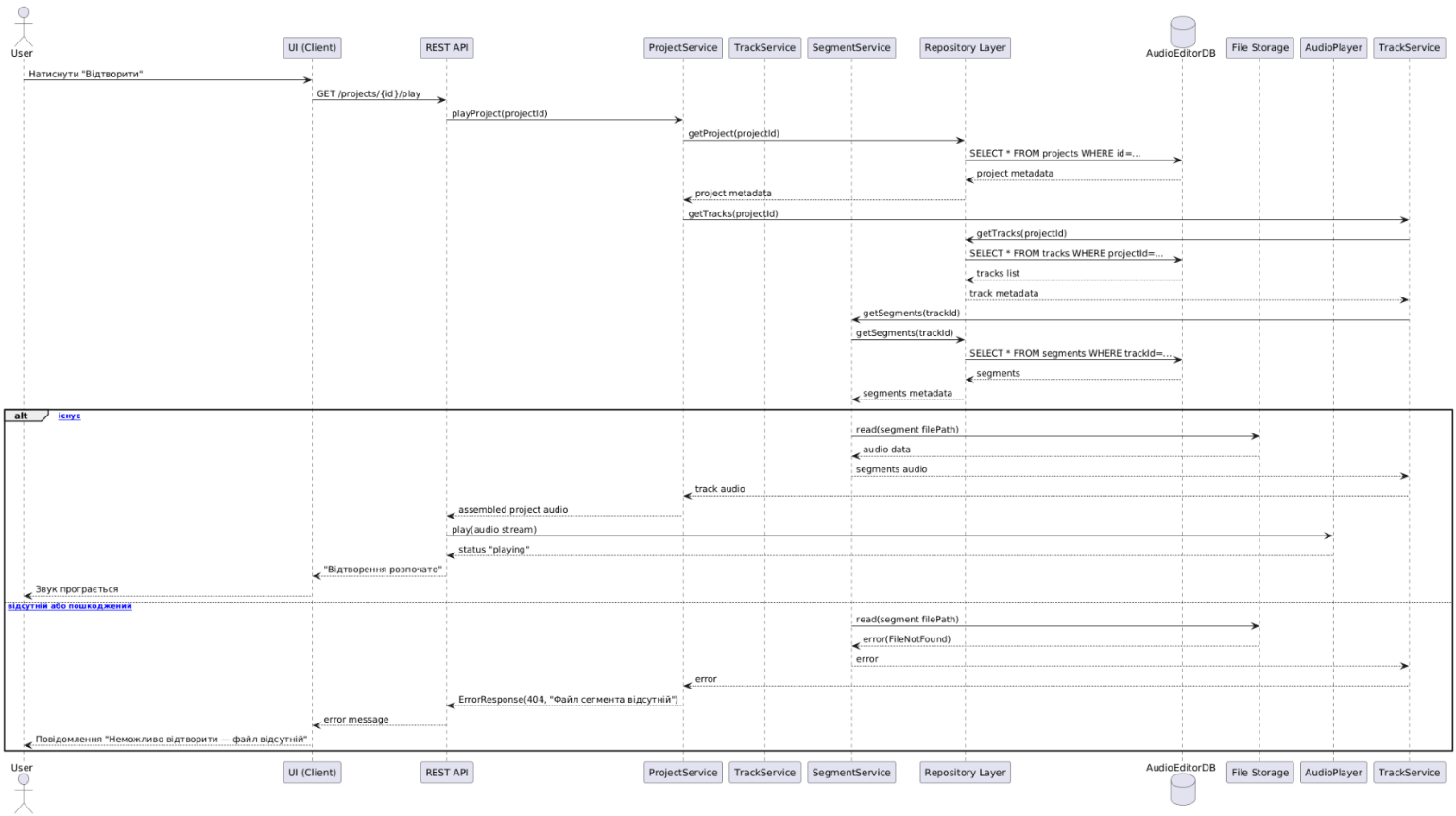


Рисунок 4.2 – Діаграма компонентів «Відтворення аудіо»

4.3. Діаграма послідовсостей «Експорт»

Діаграма моделює процес експорту аудіопроєкту та містить як «щасливий» шлях, так і виняткову ситуацію, коли конвертація неможлива через непідтримуваний формат/кодек.

Учасники

- User — користувач, який ініціює експорт.
- UI (Client) — настільний клієнтський застосунок.
- REST API — серверна точка входу для запитів від клієнта.
- ProjectService — координує процес експорту для поточного проєкту.
- TrackService — надає доріжки, що входять до проєкту.
- SegmentService — повертає сегменти доріжок і збирає з них суцільні треки.
- Repository Layer — шар доступу до БД (шаблон Repository).
- AudioEditorDB — база даних метаданих (проєкти, доріжки, сегменти).
- File Storage — файлове сховище фізичних аудіофайлів.
- ConverterService (Adapter) — адаптер до зовнішніх кодеків/енкодера.
- FFmpeg / Codecs — зовнішній модуль кодування/декодування.

Основний (успішний) сценарій

1. User у UI обирає команду «Експорт у MP3».
2. UI надсилає запит POST /projects/{id}/export?format=mp3 до REST API.
3. REST API делегує запит у ProjectService: exportProject(id, mp3).
4. ProjectService → Repository Layer → AudioEditorDB: послідовно отримуються метадані про проєкт, його доріжки та сегменти (три окремі SELECT-запити).

5. SegmentService → File Storage: читаються всі сегменти для конкретних доріжок; SegmentService формує «зібраний трек» і повертає його у TrackService.
6. TrackService → ProjectService: передає дані всіх зібраних треків.
7. ProjectService → ConverterService (Adapter): викликається конвертація convertToFormat(allTracks, mp3).
8. ConverterService → File Storage: зчитується проміжний WAV-файл.
9. ConverterService → FFmpeg/Codecs: виконується кодування (encode).
10. FFmpeg/Codecs → ConverterService: повертається готовий MP3-файл.
11. ConverterService → File Storage: збереження результату project.mp3, отримання підтвердження.
12. ConverterService → ProjectService: повертається шлях до збереженого файлу.
13. ProjectService → REST API → UI: відповідь із посиланням для завантаження; UI повідомляє User про успішне збереження.

Винятковий сценарій (непідтримуваний кодек/формат)

- Під час кроку кодування FFmpeg/Codecs генерує виняток UnsupportedFormat.
- ConverterService повідомляє ProjectService про помилку («Формат не підтримується»).
- ProjectService формує помилкову відповідь у REST API: ErrorResponse(415, "Unsupported Media Type").
- REST API → UI → User: користувач отримує зрозуміле повідомлення: «Обраний формат не підтримується».

Ключові моменти та використані шаблони

- Client–Server: взаємодія від UI до REST API і далі до сервісів.
- Repository: усі доступи до AudioEditorDB проходять через Repository Layer, що ізолює бізнес-логіку від зберігання.
- Composite (неявно в процесі складання треків із сегментів): SegmentService збирає сегменти в суцільні доріжки, які повертає TrackService.
- Adapter: ConverterService абстрагує інтеграцію з FFmpeg/Codecs.
- Розділено метадані (у БД) та бінарний аудіоконтент (у File Storage), що відображає реальний життєвий цикл обробки.

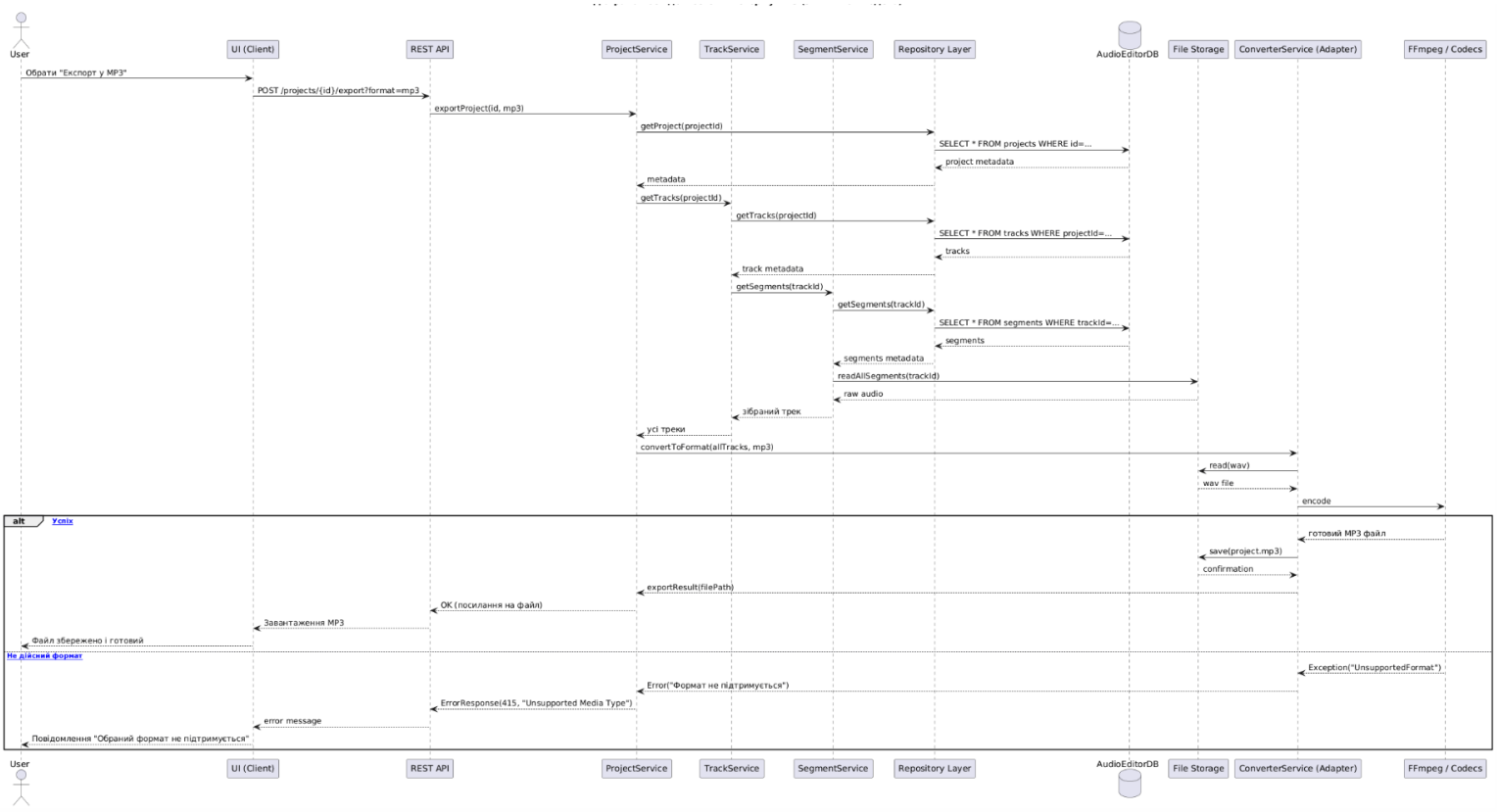
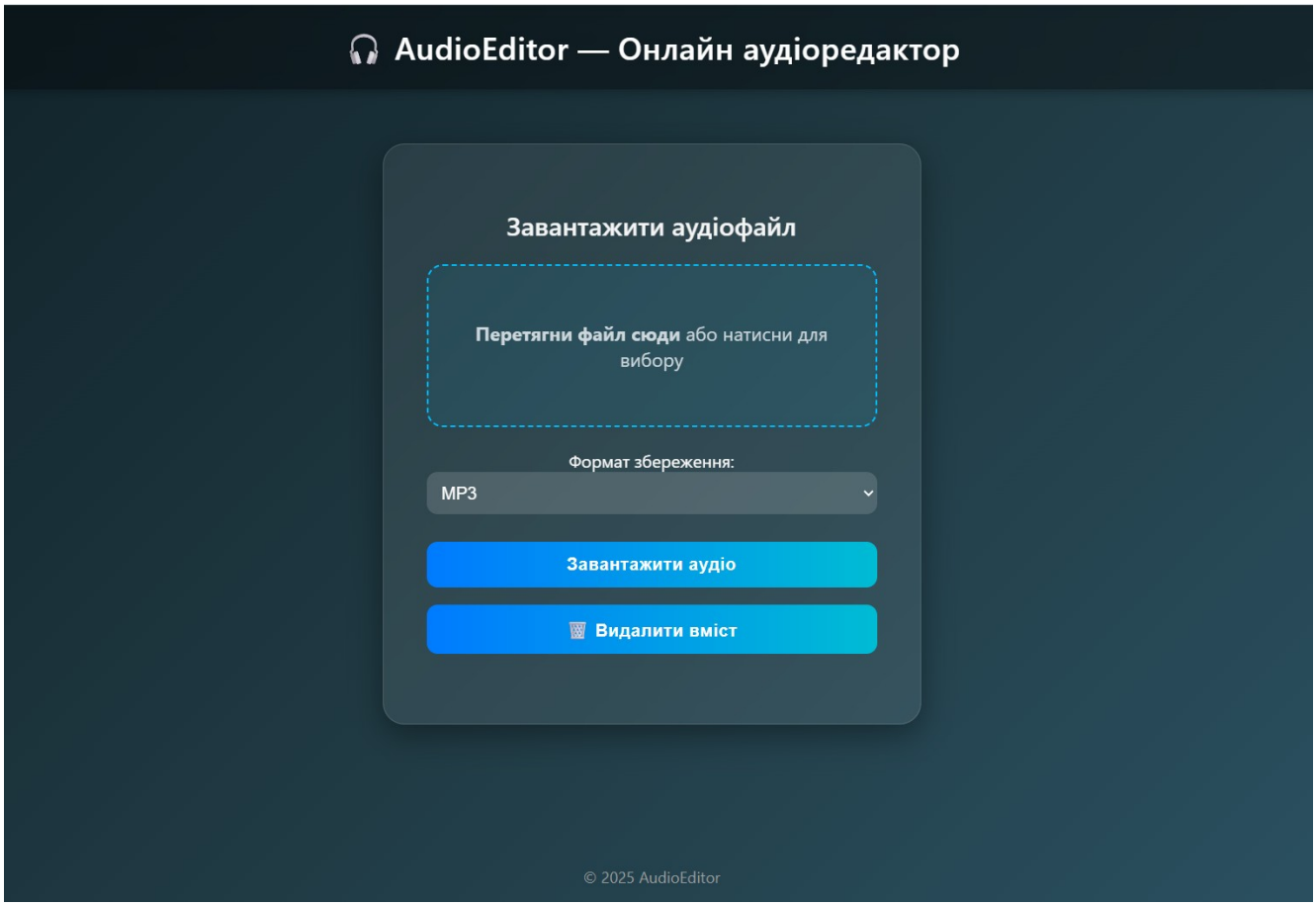
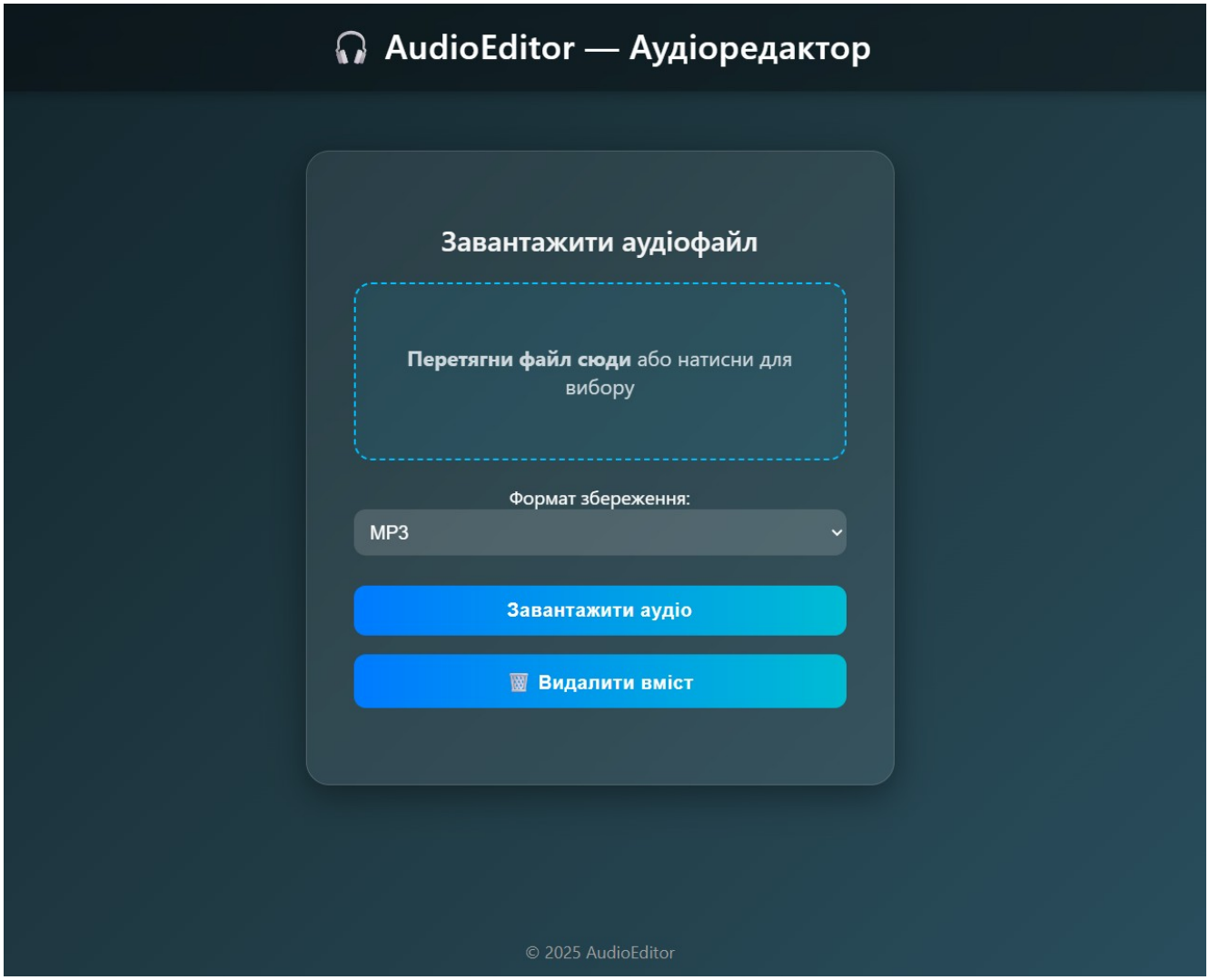
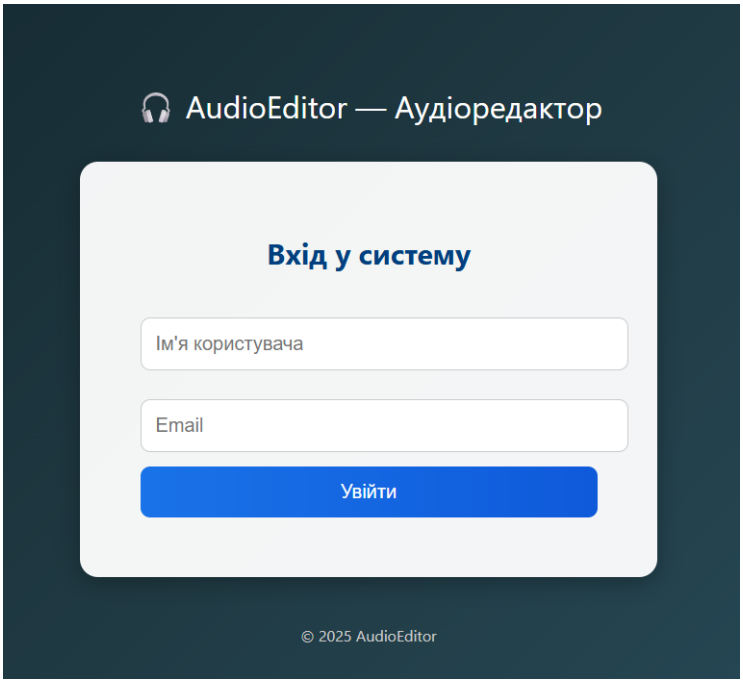


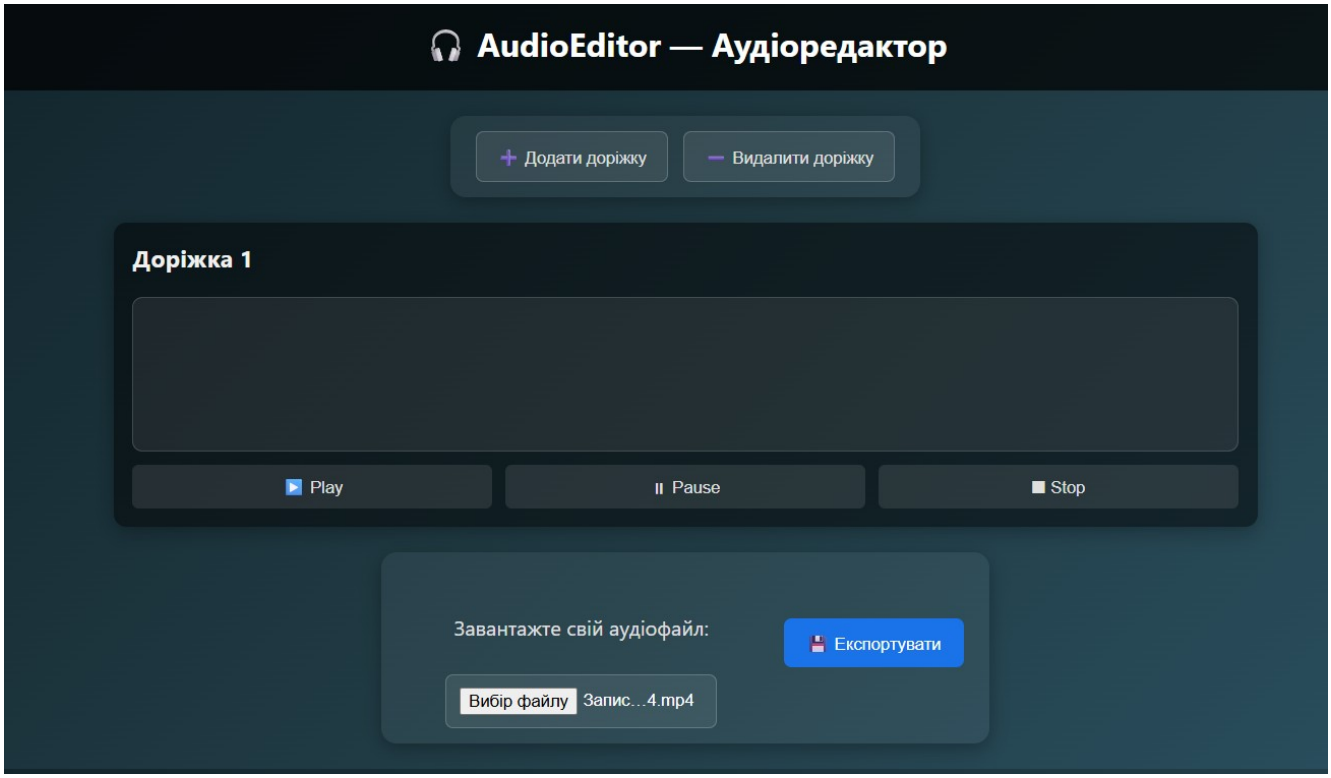
Рисунок 4.3 – Діаграма послідовностей для експорту

5. Допрацювання програмної частини.




AudioEditor — Аудіоредактор
Завантажити аудіофайл
Перетягни файл сюди або натисни для вибору
Формат збереження:
MP3
Завантажити аудіо
Видалити вміст
© 2025 AudioEditor


AudioEditor — Аудіоредактор
Вхід у систему
Ім'я користувача
Email
Увійти
© 2025 AudioEditor



Відповіді на контрольні запитання:

1. Що собою становить діаграма розгортання?

Діаграма розгортання – представляють фізичне розташування системи, показуючи, на якому фізичному обладнанні запускається та чи інша складова програмного забезпечення.

2. Які бувають види вузлів на діаграмі розгортання?

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).

3. Які бувають зв’язки на діаграмі розгортання?

Можуть бути: асоціації (фізичні з’єднання між вузлами), комунікаційні лінії (мережеві зв’язки), залежності (відображають використання одного елемента іншим).

4. Які елементи присутні на діаграмі компонентів?

Компоненти системи, інтерфейси (надавані та потрібні), залежності між ними, а також артефакти, які реалізують ці компоненти. Часто додають підсистеми для структурування.

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

Зв’язки показують відношення використання чи залежності: один компонент може використовувати інтерфейс іншого, або бути реалізованим через певний артефакт. Це дозволяє зрозуміти, як частини системи взаємодіють.

6. Які бувають види діаграм взаємодії?

В UML виділяють: діаграми послідовностей, діаграми комунікацій, діаграми часових обмежень (timing), діаграми огляду взаємодій. Всі вони відображають динамічні аспекти системи.

7. Для чого призначена діаграма послідовностей?

Вона потрібна для того, щоб показати обмін повідомленнями між об’єктами у часі. Це допомагає зрозуміти порядок виконання операцій і логіку роботи сценарію системи.

8. Які ключові елементи можуть бути на діаграмі послідовностей?

Основні елементи: актори, об'єкти чи класи, життєві лінії (lifelines), повідомлення (синхронні та асинхронні виклики), блоки альтернатив та циклів, умови виконання.

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

Діаграми варіантів використання показують, які функції виконує система з точки зору користувача, а діаграми послідовностей деталізують ці функції, описуючи послідовність взаємодій між об'єктами.

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

Діаграма класів описує статичну структуру системи, а діаграма послідовностей - динамічну поведінку об'єктів, створених із цих класів. Вона фактично демонструє, як класи співпрацюють у конкретних сценаріях.

Висновки:

У процесі виконання цієї лабораторної роботи ми набули практичних навичок з проєктування програмних систем. Було розроблено ключові UML-діаграми для системи «Аудіо редактор», включаючи 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.

Також ми значно покращили застосунок для системи «Аудіо редактор», спроєктуваши ці діаграми на практиці.