Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Бакалавр
(назва освітнього ступеня)

на тему: Розробка музичного веб-маркетплейсу з використанням хмарних технологій

Виконав: студент 4 курсу, групи СП-42
спеціальності 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

Макоїд О. М.
(підпис) (прізвище та ініціали)

Керівник Мудрик І. Я.
(підпис) (прізвище та ініціали)

Нормоконтроль Стоянов Ю. М.
(підпис) (прізвище та ініціали)

Завідувач кафедри Петрик М. Р.
(підпис) (прізвище та ініціали)

Рецензент
(підпис) (прізвище та ініціали)

Тернопіль 2025

АНОТАЦІЯ

Кваліфікаційна робота бакалавра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення». ТНТУ, 2025. Сторінок 83, таблиць 2, рисунків 34, презентація.

В даній роботі продемонстровано процес проєктування, розробки та розгортання сервісу для торгівлі аудіо продуктами. Сайт реалізовано у якості міні кластеру з двох контейнерів, а саме основного сервісу у формі вебсайту та API для накладання водяних знаків. Було проведено аналіз конкурентів, що надають схожий функціонал, виділено їх переваги та недоліки для подальшого формування плану розробки.

Об’єктом дослідження є веб-сервіс для комерційного розповсюдження аудіопродукції з механізмами захисту авторських прав.

Метою роботи є створення хмарного веб-маркетплейсу з вбудованою системою водяного захисту музичних творів.

Основна бізнес-логіка імплементована на мові Go з використанням фреймворку Fiber, тоді, як система накладання водяних знаків на Python з використанням фреймворку FastAPI та аудіостеганографії. Система розгорнута як кластер з використанням Docker-контейнерів та AWS-сервісів, а процес тестування й розгортання автоматизовано за допомогою Jenkins.

Методи дослідження включають: аналіз конкурентних систем, моделювання архітектури, тестування функціональних компонентів та автоматизоване CI/CD-розгортання.

Ключові слова: веб-маркетплейс, хмарні технології, водяний захист, Go, Fiber, аудіостеганографія, AWS, Docker, Jenkins.

ABSTRACT

Bachelor's qualification thesis. Ternopil Ivan Puluj National Technical University, Department of Software Engineering, specialty 121 "Software Engineering". TNTU, 2025. 83 pages, 2 tables, 34 figures, presentation.

This thesis demonstrates the process of designing, developing, and deploying a service for trading audio products. The website is implemented as a mini-cluster of two containers — the main service in the form of a website and an API for watermark embedding. A competitor analysis was conducted, highlighting their advantages and disadvantages to inform the development plan.

The object of study is a web service for the commercial distribution of audio content with copyright protection mechanisms.

The goal of the work is to create a cloud-based music web marketplace with a built-in watermarking system for musical works.

The core business logic is implemented in Go using the Fiber framework, while the watermarking system is built in Python using the FastAPI framework and audio steganography. The system is deployed as a cluster using Docker containers and AWS services, with automated testing and deployment via Jenkins.

Research methods include: competitor system analysis, system architecture modeling, functional component testing, and automated CI/CD deployment.

Keywords: web marketplace, cloud technologies, watermark protection, Go, Fiber, audio steganography, AWS, Docker, Jenkins.

ЗМІСТ

ВСТУП

Стрімкий розвиток цифрових технологій різко змінює динаміку глобального ринку, перетворивши інтернет на головний простір для комерції, творчості та комунікації. У цьому новому середовищі виникають цілі екосистеми, в яких продукти та послуги поширюються миттєво, а конкуренція формується не лише за якістю, а й за зручністю доступу, швидкістю взаємодії та адаптивністю сервісів. Досить чутливо такі зміни відчуває музична індустрія — одна з тих сфер, вразливість яких помічається одразу.

Поширення аудіо контенту в онлайн просторі не просто є ще одним способом популяризувати власну творчість студіям з великих лейблів, але й дати початок кар'єри тим людям, що не мають доступу до послуг дорогих студій звукозапису чи видавництв. Окремою проблемою також стає й викрадення інтелектуальної власності часто без змоги потім довести її дійсність.

На цьому тлі зростає потреба у платформах, які не лише забезпечують комфортну взаємодію між творцями та слухачами, а й враховують вимоги безпеки, крім цього ще й гнучкості та масштабованості. Адже кожен музичний проєкт — це потенційно глобальний продукт, і його цифрова присутність повинна бути готова до швидкого зростання аудиторії, обробки великих обсягів медіа та інтеграції з зовнішніми сервісами.

Успішне функціонування такої системи неможливе без врахування сучасних підходів до розробки програмного забезпечення, зокрема — модульності, автоматизації розгортання, безперервної інтеграції та тестування. Саме ці підходи дозволяють скоротити час між створенням нової функції та її появою у продуктивному середовищі, що критично важливо для конкурентоспроможних цифрових рішень. Крім того, застосування принципів DevOps та використання контейнеризації забезпечує не лише повторюваність середовищ розгортання, але й гнучкість у масштабуванні залежно від навантаження.

Також варто зазначити, що цифрові платформи, орієнтовані на роботу з медіа, повинні поєднувати високу продуктивність і низьку затримку в обробці контенту з потребою у збереженні конфіденційності та авторських прав. Це вимагає ретельного балансування між продуктивністю системи, витратами на інфраструктуру та функціональністю, яку очікують як користувачі, так і творці контенту. Підтримка саме таких сервісів передбачає активне використання хмарних технологій, що дозволяють швидко реагувати на зміну попиту й підтримувати географічно розподілену інфраструктуру.

Розробка веб-маркетплейсу для музичного контенту в таких умовах стає не лише технічно специфічним завданням, але й додає з точки зору архітектури, стійкості системи та забезпечення авторських прав. Саме тому в основі проєкту лежить використання сучасного технологічного стеку — продуктивного ядра на Go (Fiber), мікросервісів для обробки аудіо на Python (FastAPI), хмарної інфраструктури (AWS) та CI/CD-підходу із Jenkins. Така архітектура дозволяє швидко масштабувати систему, адаптувати її під зростаючі навантаження та впроваджувати нові функціональні блоки без переривання сервісу.

Окрему увагу приділено рішенню для захисту аудіофайлів — функціональності водяного знакування, що дозволяє додати приховану інформацію у звуковий сигнал і, таким чином, сприяти збереженню прав на інтелектуальну власність без шкоди для якості. Завдяки ізольованому мікросервісному підходу, цей модуль не лише зберігає автономність, а й легко інтегрується з іншими компонентами системи.

Відповідно, в даній роботі буде сфокусовано увагу на плануванні, проектуванні та імплементації вищеописаного програмного рішення, та в подальшому його розгортання й оновлення.

1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд конкурентів

Одним з найкращих сервісів музичної торгівлі на сучасному ринку є BeatStars. Це міжнародна онлайн-платформа, спрямована на торгівлю ліцензіями на музичний контент, такий, як біти, інструментальні та вокальні треки. Здебільшого вона є популярною серед музичних продюсерів, виконавців та авторів пісень, рідше великих студій.

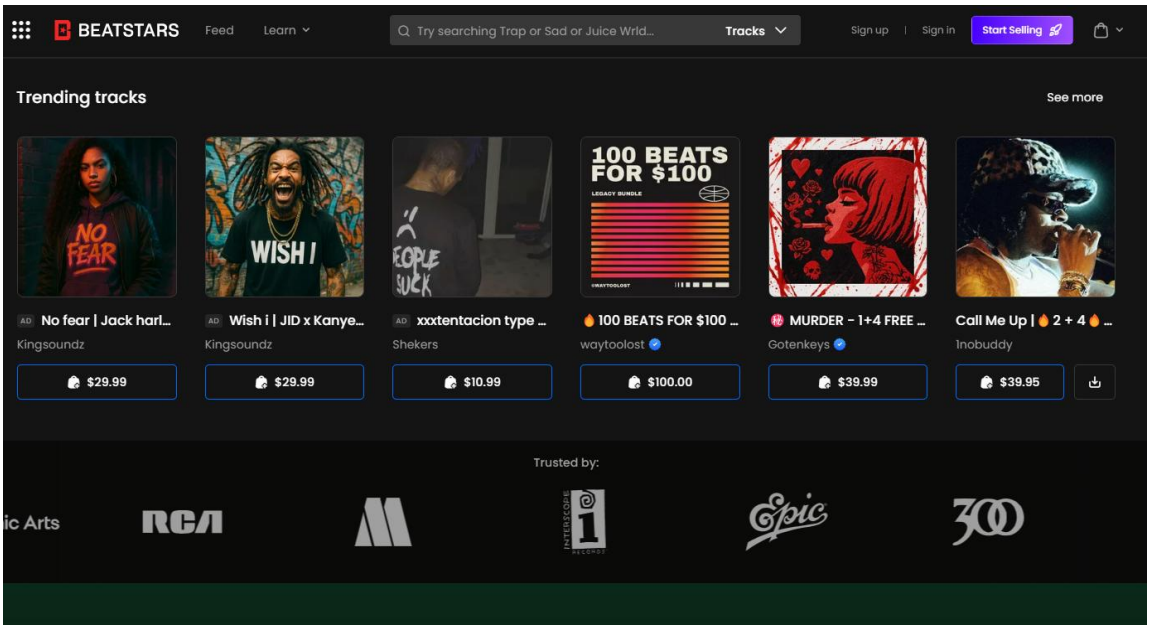

Рис. 1.1 – Зовнішній вигляд Beatstars [1]

Платформа дає змогу створювати персоналізовані сторінки, розміщувати контент з налаштуванням його цінової політики та керування ліцензій. Крім цього передбачено базовий функціонал для моніторингу статистики виконавця та комунікації між артистами. Доступним даний сервіс є вебсайт, адаптований під різні типи пристроїв, та мобільний додаток.

З додаткового функціоналу також є BeatID – система акустичних відбитків для відстежування аудіо продуктів, що перебувають під юрисдикцією даної платформи, яка використовується для захисту прав автора на інтелектуальну

власність, а також Seeds – функція, що допомагає продюсерам створювати музику, генеруючи короткі MIDI-патерни за допомогою технологій штучного інтелекту.

Ще одним популярним конкурентом є Airbit. Платформа дозволяє продюсерам завантажувати біти, створювати вітрини, керувати ліцензіями та моніторити продажі. Особливістю Airbit є орієнтація на автоматизацію процесів — завдяки інструментам автопродажів користувач може продавати контент без постійного ручного втручання. Інтерфейс Airbit доволі функціональний, однак може виглядати дещо технічно для новачків.

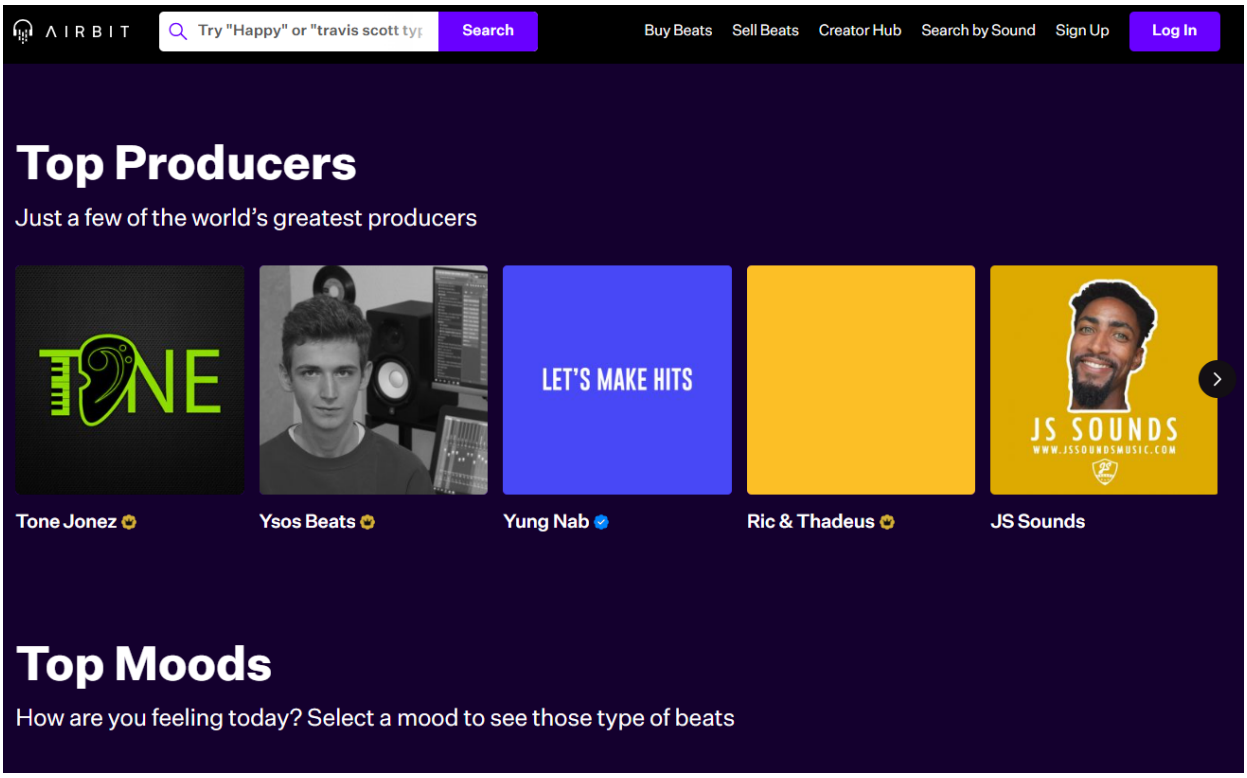

Рис. 1.2 – Зовнішній вигляд Airbit [2]

Платформа також підтримує аналітику, інтеграцію з PayPal та розширену статистику щодо слухачів. Незважаючи на широку функціональність, як і у випадку з BeatStars, фокус тут робиться на продажі та монетизації.

Обидва сервіси мають ряд власних переваг та недоліків. Для більш чіткого розуміння необхідно провести певний порівняльний аналіз, який дасть достатньо

інформації для подальшого визначення, чого варто уникати під час розробки, що варто врахувати, і який напрямок розробки буде в даного проекту.

У попередньому розділі ми розглянули два основних конкуренти для мого проєкту: BeatStars та Airbit. Для розуміння, в якому напрямку рухатися, у цьому розділі ми детально розглянемо переваги та недоліки цих платформ, як вони відповідають сучасним вимогам та відгуки користувачів. Порівняльний аналіз цих двох платформ дасть чітке розуміння, чого варто уникати при розробці платформи, яка буде їхнім конкурентом, та на які рішення варто рівнятися.

Почнемо з BeatStars. Серед його переваг можна виділити наступні:

1. Платформа пропонує широкий набір інструментів для продажу, ліцензування та монетизації музики.
2. Користувачі можуть самостійно налаштовувати типи ліцензій, ціни та умови використання бітів.
3. Спільнота: активна база користувачів, що полегшує мережеву взаємодію між продюсерами, виконавцями та авторами.
4. Інтеграція з YouTube Content ID та Facebook Rights Manager: захист контенту та монетизація на сторонніх платформах.
5. Простота у завантаженні, організації та відстеженні треків і ліцензій.
6. BeatStars легко вбудувати до власного сайту або соцмереж.

Недоліки платформи також варто врахувати:

1. Висока конкуренція: через велику кількість користувачів складно виділитися, особливо новачкам.
2. Дещо заплутаний інтерфейс: велика кількість функцій може ускладнювати роботу початківцям.
3. Комісії та підписка: щоб мати доступ до всіх функцій, необхідно оформити платну підписку; безкоштовна версія сильно обмежена.
4. Не завжди швидка підтримка: користувачі іноді скаржаться на повільні відповіді служби підтримки чи роботи сервісу загалом.

BeatStars сама по собі є досить потужною платформою для музичних продюсерів. Вона дає великі можливості в плані монетизації, гнучкості та

професійної присутності, однак для новачків чи тих, хто надає перевагу простоті, інтерфейс та кількість функцій та складність деяких процесів можуть бути радше неприйнятним фактором.

Тепер розглянемо Airbit. Основні його переваги наступні:

1. Платформа інтуїтивно зрозуміла навіть для нових користувачів.
2. Продюсери можуть побачити інформацію про прослуховування, завантаження та продажі в реальному часі.
3. Після покупки користувачі одразу отримують файли та ліцензії.
4. Вбудовані маркетингові інструменти: email-розсилки, купони, знижки та партнерські програми.
5. Присутня зручна система платежів для глобальної аудиторії.

Втім, Airbit також має свої недоліки:

1. Порівняно з BeatStars, платформа має обмежені інструменти для просування через соцмережі.
2. Обмежена спільнота: не настільки активна та велика база користувачів, як у BeatStars.
3. Дизайн магазину не завжди гнучкий: користувачі мають менше можливостей для унікального візуального оформлення.
4. Під час певних функціональних нововведень сервіс може переходити в режим maintenance на досить довгі терміни, і часто не всі функції працюють належно після перезапуску.

Як можна помітити, обидва конкуренти мають свої як переваги, так і недоліки. Ідеальним варіантом буде притримуватися золотої серединки між простотою та функціоналом. Особлива увага буде приділена роботі на базі хмарних технологій, варіативності продуктів і спрощеному керуванню контентом.

1.2 Обґрунтування вибору напрямку дослідження

У процесі розробки будь якого програмного забезпечення важливою складовою є визначення оптимального стеку технологій та архітектурного підходу, який би забезпечив масштабованість, стабільність, зручність у підтримці та швидкість розробки.

Проналізувавши основний функціонал потенційних конкурентів, а також з огляду на потреби у високій доступності та масштабованості, основою було обрано модульно монолітну архітектуру з елементами мікросервісів. Це такий собі гібрид, який бере переваги з обох архітектурних стилів (моноліт та мікросервіси), та утворює власний. Його суть полягає у наявності сервісу, який імплементує основну бізнес логіку та виконує певну взаємодію з додатковими “мікросервісами”, які виконують свою окрему, проте не менш важливу бізнес функцію. Даний підхід передбачає декілька плюсів, а саме:

1. Централізація: всі мікросервіси здебільшого взаємодіють суто з основним сервісом.
2. Модульність: попри наявність в даній архітектурі елементів монолітності, це не заважає додавати новий функціонал за потреби. У випадку, якщо відбудеться певний збій, таку систему легше обслуговувати.
3. Частково уникається складність підтримки міжсервісної комунікації.

Загалом по завершенню першої ітерації циклу розробки, готовий продукт повинен являти собою міні кластер з двох Docker контейнерів, один з яких є головним сервером, що містить основну бізнес логіку, а інший – API для модифікації об’єктів зі сховища. Обидва контейнери працюватимуть на базі дещо різних технологій та фреймворків, оскільки напрямки застосування інструментів всередині відрізняються концептуально.

Для реалізації основної серверної частини було обрано мову програмування Go (Golang). Це компільована, статично типізована мова з відкритим кодом, що

поєднує в собі простоту синтаксису (на кшталт Python) та високу продуктивність (як C/C++, наприклад) [3]. Однією з ключових причин вибору Go є його технічні переваги, серед яких висока швидкість виконання, досягнута за рахунок компіляції у машинний код без використання віртуальних машин; простий і лаконічний синтаксис, який знижує технічний борг та спрощує супровід коду; а також вбудовані інструменти керування залежностями, модульність, автоформатування та профілювання, що робить Go зручним для побудови масштабованих систем.

Також за основу було взято Fiber – веб-орієнтований фреймворк, побудований поверх високопродуктивної бібліотеки FastHTTP.

Рис. 1.3 – Логотип фреймворку Go Fiber

Даний фреймворк за своїм синтаксисом чимось подібний на Flask (аналог для Python), однак з деякими перевагами, такими, як краща продуктивність та більші масштаби застосування через його ефективність використання ресурсів. Вибір даного фреймворку зумовлений потребою у побудові компактного, стабільного та легко масштабованого вебдодатку, оптимізованого для розгортання у контейнеризованому середовищі [4].

Окрема частина функціональності — накладення цифрового водяного знаку на аудіофайли — винесена в мікросервіс, реалізований на мові програмування Python із використанням вебфреймворку FastAPI.

Python — популярна мова високого рівня з фокусом на читабельність коду та широкими можливостями для наукових обчислень, обробки сигналів та машинного навчання [5]. Вона має велику екосистему відкритих бібліотек, що значно прискорює процес розробки складної логіки. FastAPI, у свою чергу, є сучасним асинхронним фреймворком для створення веб-API, який базується на стандарті OpenAPI і підтримує автоматичну генерацію документації, типізацію на основі Python type hints і високу продуктивність завдяки використанню асинхронного сервера Uvicorn [6]. Для реалізації накладання водяного знаку у мікросервісі застосовувались бібліотеки numpy та scipy. Перша з них забезпечує ефективну роботу з багатовимірними масивами та векторами, що є основою для обробки аудіосигналу, друга — scipy — надає розширений набір інструментів для наукових обчислень, зокрема модуль scipy.fft дозволяє виконувати швидке перетворення Фур'є для роботи у частотній області. Це дасть змогу реалізувати базовий алгоритм цифрового водяного знаку через модифікацію частотних складових аудіофайлу, зберігаючи при цьому його якість для кінцевого користувача.

В контексті розробки музичного веб-маркетплейсу як основний інструмент для розгортання мікросервісної архітектури буде застосовуватися Docker. Це платформа контейнеризації, яка дозволяє створювати, запускати та керувати ізольованими середовищами для виконання програм, які називаються контейнерами. Ці контейнери дозволяють упакувати додаток разом з усіма його залежностями — бібліотеками, конфігураціями, середовищем виконання — в один уніфікований образ. Кожен компонент системи, наприклад, бекенд, окремий API для обробки аудіо, бази даних, об’єктні сховища (тільки для development та test) або інші інструменти — працюватиме у власному ізольованому контейнері [7].

Рис. 1.4 – Логотип Docker.

У даному проєкті всі сервіси запускаються як окремі Docker-контейнери на віртуальному сервері в AWS EC2, що дозволяє централізовано керувати додатком, перезапускати окремі сервіси у випадку помилок, застосовувати оновлення без потреби зупиняти весь проєкт та легко масштабувати систему в подальших ітераціях.

Щодо Amazon EC2 (Elastic Compute Cloud), то це одна з ключових послуг Amazon Web Services (AWS) — хмарної платформи, яка пропонує широкий спектр інфраструктурних і платформних сервісів, що дозволяють розробляти, запускати й масштабувати вебдодатки без необхідності обслуговування фізичних серверів. Натомість використовуються віртуальні машини (інстанси), які можна гнучко налаштовувати відповідно до потреб проєкту. Зважаючи на те, що обчислювальні ресурси не є безкоштовними, AWS застосовує модель pay-as-you-go, тобто оплата здійснюється лише за фактично використані ресурси.

Рис. 1.5 – Логотип Amazon Web Services

У рамках даного проєкту використовуються три основні сервіси AWS: Amazon EC2, Amazon S3 та Amazon RDS.

Amazon EC2 — це сервіс для запуску віртуальних серверів, який забезпечує масштабоване обчислення в хмарі. У цьому проєкті EC2 використовується для хостингу всієї інфраструктури, розгорнутої у вигляді Docker-контейнерів. Контейнери можуть автоматично перезапускатися у разі збоїв, а їх оновлення можливе без зупинки всієї системи, що підвищує надійність і зменшує час простою [8].

Amazon S3 (Simple Storage Service) — це сервіс об’єктного зберігання, який використовується для зберігання будь-яких типів файлів, зокрема аудіофайлів, які завантажують користувачі платформи. S3 забезпечує високу надійність, гнучкі механізми контролю доступу, а також повну інтеграцію з іншими сервісами AWS. Завдяки цьому користувачі можуть безпечно зберігати, передавати та ділитися музичним контентом з мінімальними затримками [8].

Amazon RDS (Relational Database Service) — це керований сервіс реляційних баз даних, який підтримує основні СУБД, зокрема PostgreSQL, MySQL та SQL Server. У межах цього проєкту RDS використовується як основне реляційне сховище для метаданих: інформації про користувачів, музичні треки, ліцензії, рейтинги артистів тощо [8].

Для локальної розробки та тестування використовуватиметься MinIO — об’єктне сховище з відкритим кодом, повністю сумісне з протоколами Amazon S3. Таким чином це дозволить створювати і тестувати функціональність, пов’язану зі зберіганням та модифікацією аудіо, без підключення до реального хмарного середовища, що знижує витрати та прискорює цикл розробки. MinIO працює швидко, легко розгортається у Docker-контейнері та імітує всі основні можливості S3, включаючи API, політики доступу та версіонування.

Для автоматизації з відкритим кодом, який використовується для реалізації процесів безперервної інтеграції та безперервного розгортання (CI/CD) буде використовуватися Jenkins. Його головне призначення — автоматизувати всі етапи життєвого циклу розробки програмного забезпечення: від перевірки змін у коді до його збірки, тестування та доставки в продакшн-середовище.

У межах розробки маркетплейсу Jenkins забезпечуватиме автоматизоване оновлення окремих сервісів після внесення змін до їхнього коду, особливо це буде актуальним в подальших ітераціях життєвого циклу даного ПЗ. Після того, як розробник завантажує нову версію до репозиторію, Jenkins автоматично запускає відповідні пайплайни: виконується збірка Docker-образу, тестування, оновлення контейнера на сервері, і, в разі успішного проходження всіх етапів, деплой в хмарне середовище. Такий підхід дозволяє уникнути помилок, які можуть виникнути під час ручного оновлення, та суттєво пришвидшує цикл доставки нового функціоналу [9].

Крім цього, Jenkins дозволяє централізовано контролювати весь процес розгортання, що спрощує супровід та підтримку проєкту. Його гнучкість у конфігурації дозволяє налаштовувати пайплайни саме під потреби конкретної архітектури, що є критично важливим у випадку використання кількох мов програмування, середовищ та сервісів. Таким чином, Jenkins виступає ключовим інструментом у забезпеченні надійної, стабільної та швидкої доставки оновлень, що є необхідним для сучасного підходу до розробки хмарних вебдодатків.

1.3 Вибір методології розробки та керування кодовою базою

Під час проєктування та реалізації будь якого програмного продукту виникає низка певних технічних проблем, обумовлених складністю системи, багатокомпонентністю а також вимогами до масштабованості, стабільності та швидкої доставки оновлень. Для оптимізації та систематизації розробки таких, та й будь яких інших програмних продуктів було запроваджено спеціальні набори правил та принципів, які по іншому також називаються методологіями розробки.

Перш за все, такі правила надають передбачуваності розробці, полегшують менеджмент команди та зменшують ймовірність серйозних ризиків. У випадку відсутності чіткої системності та структурованості виникнуть проблеми з

низькою продуктивністю команди розробників, поганою якістю вихідного продукту, відсутністю нових клієнтів та втратою поточних, а також низкою інших абсолютно непердбачуваних проблем.

Відповідно, методології створені для того, щоб такого уникнути, тим самим приносячи користь як розробникам, полегшуючи їх роботу, збільшуючи ефективність, забезпечуючи точніші терміни виконання, та полегшуючи роботу з непередбачуваними змінами; так і клієнтам, полегшуючи їм можливість відстежування прогресу розробки, якщо такий надається.

Проте, з урахуванням кількості вже існуючих методологій, немає ідеального варіанту. Все залежить від типу та масштабу, специфіки команди, та контексту технічної реалізації.

З урахуванням специфіки даного проєкту, його масштабів, а також технічного та людського ресурсу, одним з найкращих варіантів методологій є Kanban — гнучка методологія управління робочим процесом, яка орієнтується на візуалізацію задач, обмеження одночасної роботи та забезпечення безперервного потоку виконання. Вона не нав’язує чіткої структури чи фіксованих ітерацій, як Scrum, а дозволяє адаптувати процес під власний стиль і ритм роботи [10].

Для одного розробника Kanban є особливо ефективним, і ось чому:

1. Методологія не потребує ролей, “бюрократії” чи складної структури. Не потрібно проводити щоденні стендапи, планування спринтів чи ретроспективи — достатньо мати дошку з задачами.
2. Завдання можна змінювати в будь-який момент – деатльно для умов, коли пріоритети часто змінюються або розробник сам приймає рішення, над чим працювати далі. На відміну від Scrum, не потрібно чекати завершення спринту для адаптації.
3. Kanban-дошка дозволяє бачити повний обсяг роботи, визначати вузькі місця та легко контролювати прогрес. Навіть одна особа може швидше зорієнтуватися у власному навантаженні та уникнути перевантаження.

4. Через обмеження на кількість задач у роботі (WIP-ліміти) Kanban дисциплінує не братися за нове, поки не завершено попереднє.
5. Досить легко моніторити прогрес, і на його основі подавати звітність замовнику.
6. Безперервне вдосконалення. Також Kanban дозволяє поступово вдосконалювати процес, спостерігаючи, як змінюється швидкість виконання задач, де виникають затримки, і де можна оптимізувати роботу.

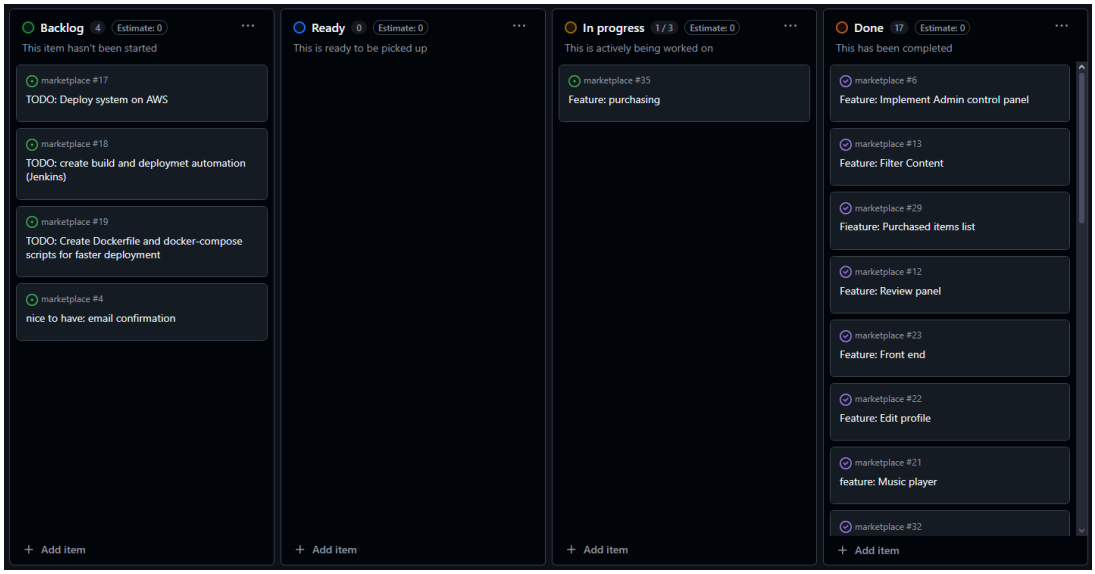

Рис. 1.6 – Візуалізація методології Kanban за допомогою GitHub Projects.

Загалом, Kanban чудово підходить для гнучкого планування, моніторингу, та, безпосередньо розробки, особливо з урахуванням технічних аспектів поточного проєкту та його масштабів.

Окрім методології розробки важливим аспектом також є підхід до керування кодовою базою. Ці підходи також є особливо важливо враховувати, оскільки вони забезпечують системність та надійність технічно, в той час, як методології розробки – фізично, себто людський фактор. Вони визначають те, як команда зберігає проєкт, інтегрує код та випускає оновлення. Чому це необхідно: незалежно від розміру команди та масштабу роботи, вже готовий код чи зміни до нього потрібно десь зберігати та мати змогу зробити бекап, зберігати всі внесені

зміни та версії. Відповідно для такого необхідно створити певний репозиторій, який би і виконував таку роботу.

Великі IT компанії мають власні сервери з об'єктними сховищами, де вони й зберігають вихідний код своїх розробок. Для малих команд, які не працюють над створенням великих проєктів достатньо звичайного носія з гарантією надійного зберігання. В нашому випадку це віддалений репозиторій. Це гарантує вищезазначену надійність зберігання даних, оскільки безпека залежатиме від постачальника послуг (в нашому випадку це GitHub), що мінімізує ризики втратити проєкт через можливі апаратні несправності, втрату носія, стихійні лиха та інших несприятливих ситуацій.

Щодо керування самою кодовою базою (репозиторієм), то для розроблюваного сервісу найбільше підійде Trunk Based Development. Перш за все даний підхід ідеальний для невеликих команд та соло-проєктів. Його суть полягає в тому, що всі розробники працюють в одній гілці, окремій від main, і виконують до неї часті малі коміти.

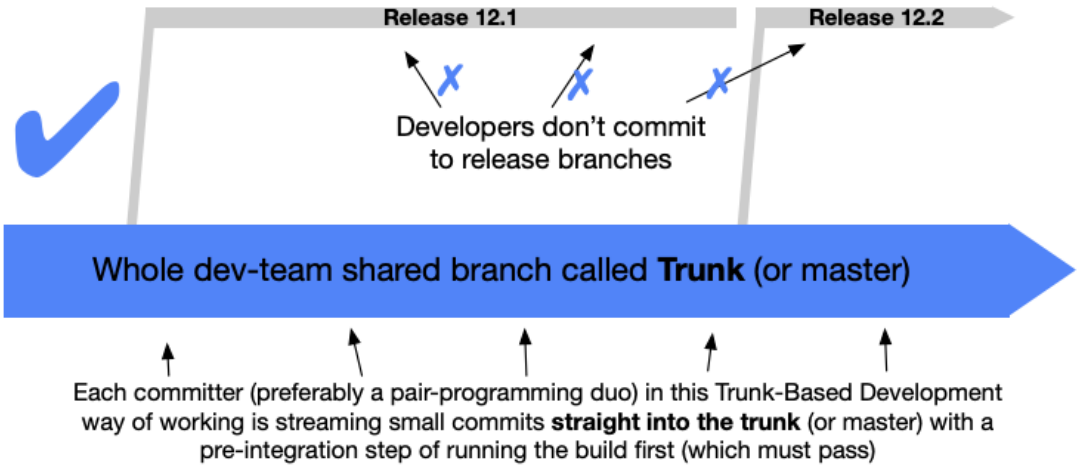

Рис. 1.7 – Принцип TBD, оптимізований під малі команди [11].

Такий підхід розрахований на те, щоб мінімізувати часовий розрив між моментом створення коду та його інтеграції у репозиторій, тим самим зменшуючи кількість конфліктів, роблячи процес стабільшіншим та потенційно прискорюючи

релізи готових версій продукту. Основні принципи Trunk Based Development (TBD) полягають в наступних аспектах:

1. Всі зміни потрапляють до спільної гілки. Усі гілки суто для короткотривалих задач;
2. Зміни надсилаються до репозиторію декілька разів на день, щоб підтримати кодову базу в актуальному стані.
3. Якщо є якийсь недоопрацьований функціонал, то він захований під feature-прапорцями. Це дозволяє розгортати новий функціонал без активації того, що ще не готовий.
4. CI/CD: trunk Based Development дуже тісно пов’язаний з автоматизацією, себто усі зміни повинні проходити автоматизоване тестування перед зливанням в основну гілку.
5. Оскільки розробники постійно інтегрують свої зміни, то великих конфліктів майже не виникає.

З переваг такого підходу варто виділити прискорений вихід оновлень, полегшений контроль над процесом, а також координацію команди та, як зазначено вище, зменшує ризик критичних конфліктів.

Загалом, попри те, що Trunk Based Development здебільшого орієнтований на розробку в “швидкому середовищі”, де потрібна швидка доставка (SaaS-продукти, до прикладу), його також можна без проблем застосовувати і тоді, коли проєкти не є дуже масштабними.

1.4 Формування вимог до системи

В одному з попередніх розділів було проаналізовано потенційних конкурентів розроблюваного сервісу, а саме Airbit та BeatStars. Обидві платформи мають досить просунутий функціонал, однак не без слабких сторін. Отриманих

результатів досліду та приблизного розуміння потреб сучасних користувачів в даній сфері достатньо для формування вимог.

Постанова завдання

Вихідний продукт призначений для наступних цілей:

- Комерційна діяльність (торгівля музичними цифровими товарами);
- Популяризація маловідомих виконавців;
- Захист аудіо продуктів спектральним шифруванням;

Функціональні вимоги

Для даного веб-маркетплейсу передбачено вимоги щодо наявності наступного функціоналу:

- Авторизація з сесійною системою (залишатися залогіненим протягом певного часу);
- Вихід з системи;
- Можливість адміністрування сервісом;
- Перегляд цифрових товарів та виконавців;
- Перегляд інформації про цифровий товар;
- Пошук товарів;
- Фільтрування товарів;
- Програвання цифрового товару (якщо аудіо);
- Керування списками відтворення;
- Додавання аудіо до списку відтворення та видалення звідти;
- Можливість залишати відгук про товар;
- Можливість ставити “вподобання” для підняття рейтингу виконавця;
- Додавання товару до кошика та видалення з нього;
- Покупка наявного в кошику;
- Можливість завантажити придбані товари;
- Завантаження власних товарів для продажу;
- Автоматичний захист аудіо водяним знаком (окремий мікросервіс);
- Зчитування водяного знаку;
- Перегляд власного профілю;

- Редагування профілю;
- Автоматизоване тестування та деплой;

Нефункціональні вимоги

Також нижче наведено перелік нефункціональних вимог:

- Безпека. Маркетплейс повинен бути захищений від зловмисного втручання. Чутливі дані мають зберігатися у форматі хешованих повідомлень, а поля введення захищені від потенційних SQL-ін’єкцій та інших загроз.
- Система має бути зрозумілою для використання на інтуїтивному рівні.
- Маркетплейс повинен мати змогу водночас обробляти до 1000 запитів водночас/
- Маркетплейс повинен мати інфраструктуру, яку можна буде в подальшому масштабувати, якщо така потреба появиться.

Деякі з вищевказаних вимог буде реалізовано в подальших ітераціях життєвого циклу програмного забезпечення з метою продемонструвати роботу автоматизації тестування та розгортання інфраструктури на хмарній платформі з метою подальшого її масштабування.

1.5 Проєктування відношень між акторами та прецедентами

Проектований сайт передбачає наявність в системі трьох акторів:

1. Користувач:
   - Має змогу авторизуватися на сайті;
   - Переглядати каталог;
   - Керувати списками відтворення;
   - Залишати коментарі та “вподобання” на сторінках товару;
   - Програвати аудіо;

- Купувати товари, що вже є в корзині та завантажувати їх;
- Редагувати профіль;
- Завантажувати власні товари;

2. Виконавець:
   - Може переглядати статистику свого облікового запису;
   - Керувати власними цифровими товарами;
3. Адміністратор:
   - Може здійснювати вхід в панель адміністратора сайту;
   - Переглядати статистику сайту;
   - Керування базами користувачів, товарів та плейлистів;

Отримавши всі необхідні зв’язки між акторами тепер потрібно структурувати цю інформацію у вигляді таблиці варіантів використання. Сумарно виявлено 12 прецедентів, з яких 7 належать звичайному користувачеві, 2 належать виконавцю та 3 – відповідно, адміністратору.

Таблиця 1 – Варіанти використання системи

Код	Актор	Назва варіанту використання	Опис
К1	Користувач	Реєстрація в системі	Користувач може зареєструватися в системі для подальшого входу в неї.
К2	Користувач	Вхід в систему	Відповідно до прецеденту К1 користувач, маючи обліковий запис, може увійти в систему.
К3	Користувач	Вихід із системи	Маючи обліковий запис, користувач може вийти з системи
К4	Користувач	Створювати плейлист	Користувач може створити плейлист аудіо для подальшого додавання до нього аудіо
К5	Користувач	Видалити плейлист	Якщо в користувача є плейлисти, які йому не потрібні, він може їх видалити.

Продовження таблиці 1

К6	Користувач	Додати до плейлиста	Користувач може додати до плейлиста аудіо, з можливістю подальшого прослуховування.
К7	Користувач	Видалити з плейлиста	Якщо є в плейлисті небажані треки, користувач може їх видалити.
К8	Користувач	Шукати продукт	Переглядаючи каталог користувач може використати панель пошуку для пошуку специфічного продукту.
К9	Користувач	Фільтрувати каталог	Користувач має змогу профільтрувати каталог для пошуку товарів за певним типом.
К10	Користувач	Додати до корзини	Користувач може додати до корзини певний товар.
К11	Користувач	Придбати додане до корзини	Користувач може придбати товари зі списку в корзині.
К12	Користувач	Завантажити придбане	Якщо товари є у власності користувача, то можуть бути завантажені.
К13	Користувач	Переглянути профіль виконавця	Користувач має змогу переглядати профілі виконавців (включно з каталогом)
К14	Користувач	Програти аудіо	Будь який користувач може програвати аудіо
В1	Виконавець	Перегляд статистики	Виконавець може переглядати статистику вподобань.
В2	Виконавець	Завантажити продукт	Виконавець може завантажувати власні товари для продажу ліцензії на них.
В3	Виконавець	Накласти захист на аудіо	Якщо виконавець завантажує на сервіс аудіо, то окремий API накладає на нього водяний знак

Продовження таблиці 1

B4	Виконавець	Видалити продукт	Виконавець може видалити товар, який він завантажив, з сервісу.
A1	Адміністратор	Вхід у панель адміністратора	Адміністратор має змогу відкрити панель адміністратора, звідки він отримує доступ до всіх даних сайту.
A2	Адміністратор	Перегляд статистики сайту	Адміністратор має змогу переглядати статистику сайту, а точніш загальну кількість зареєстрованих користувачів, продуктів та плейлистів.
A3	Адміністратор	Керування базами даних	Адміністратор має прямий доступ до баз даних користувачів, товарів та плейлистів та всіх дотичних до них таблиць.

Відповідно до Таблиці 1 варто створити діаграму варіантів використання для покращення візуального сприйняття та змоги ознайомитися з повним функціоналом системи, який потрібно імплементувати. Для цього на рисунку 1.8 зображено повну діаграму варіантів використання з усіма прецедентами.

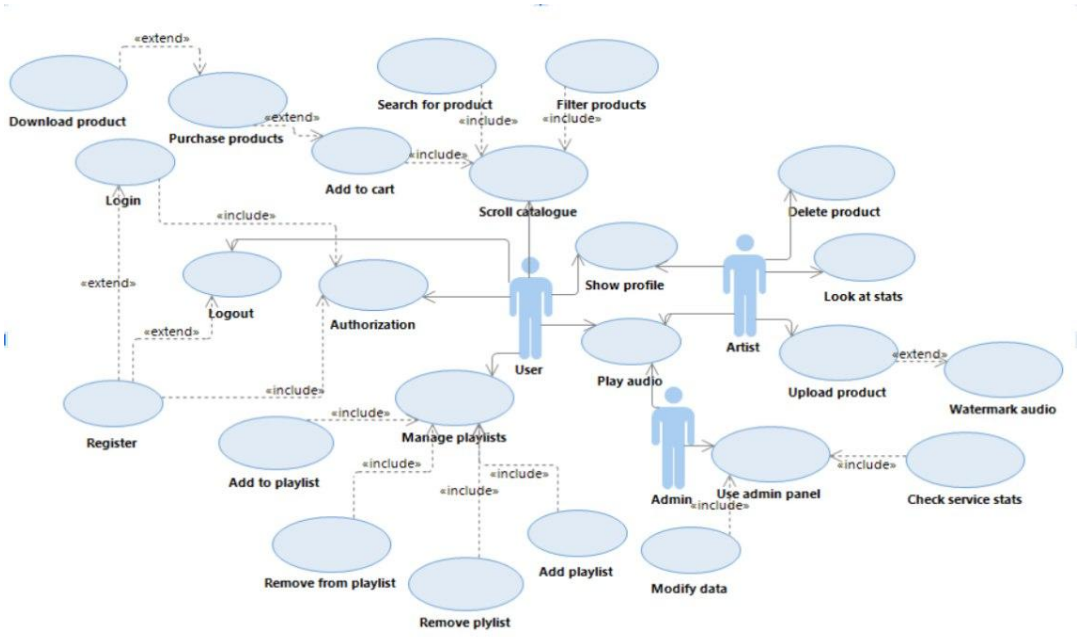

Рис. 1.8 – Повна діаграма варіантів використання системи.

У цьому розділі проведено огляд предметної області, визначено основні потреби користувачів та проаналізовано існуючі рішення на ринку. Виявлено їхні сильні та слабкі сторони, що дозволило окреслити ключові вимоги до системи. Також обґрунтовано вибір архітектури та формату реалізації — вебзастосунок, як оптимальне рішення для даного типу сервісу. Визначено технологічний підхід, що забезпечує масштабованість, гнучкість та ефективність розробки. Наостанок, було сформовано перелік функціональних і нефункціональних вимог, а також визначено основних користувачів системи. Побудовано діаграму варіантів використання, що відображає взаємодію між акторами та системою.

2. РОЗРОБКА МОДЕЛІ ТА ПРОГРАМНОГО КОМПЛЕКСУ

2.1 Проєктування архітектури маркетплейсу

Проєктування архітектури програмного забезпечення є критично важливим етапом, що визначає загальну структуру та логіку взаємодії між компонентами системи. Саме на цьому рівні закладаються основи масштабованості, продуктивності й надійності майбутнього рішення. Добре спроєктована структура дозволяє уникнути технічного боргу в майбутньому та забезпечує зрозумілу основу для командної розробки.

Ретельне планування взаємозв’язків між сервісами, модулями й інтерфейсами сприяє ефективнішій реалізації змін у кодовій базі без суттєвих ризиків для стабільності. Окрім того, продумана модель системи полегшує впровадження DevOps-практик, тестування та моніторинг. Таким чином, якісне проєктування зменшує ризики помилок у продакшн-середовищі та сприяє довгостроковій підтримуваності продукту.

2.1.1 Визначення архітектури системи

У сучасній розробці вебплатформ важливо забезпечити ефективну, стабільну та масштабовану архітектуру, що відповідає особливостям і вимогам системи. У випадку музичного маркетплейсу, що обробляє медіа-дані, виконує запити користувачів, керує акаунтами й каталогом контенту, доцільно реалізувати модульний моноліт із включенням окремих мікросервісів. Такий підхід дозволяє зберегти цілісність бізнес-логіки в межах одного бекенд-додатку, розробленого на Go з використанням Fiber, із можливістю відділення ресурсомістких чи спеціалізованих підсистем у незалежні компоненти, реалізовані на FastAPI.

Архітектура застосунку передбачає класичну клієнт-серверну модель, де взаємодія відбувається через HTTP/HTTPS-протоколи. Користувач працює з інтерфейсом, створеним за допомогою HTML, CSS та JavaScript, що дозволяє формувати запити до серверної частини, отримувати відповіді та оновлювати інформацію без необхідності повного перезавантаження сторінок. Такий підхід є гнучким і легким для підтримки, особливо у випадку поступової розробки функціоналу без повного перенавантаження фронтенду.

Серверна логіка ядра написана мовою Go, що забезпечує високу швидкодію, ефективне керування пам’яттю та швидкий час відгуку. Fiber у свою чергу дозволяє створити чисту модульну структуру застосунку, де кожен функціональний блок — наприклад, авторизація, управління медіа, каталогізація треків, історія замовлень — реалізований як окремий модуль у складі одного монолітного додатку, що досить непогано спростить розробку, пришвидшитть інтеграцію нових функцій та дозволить уникати проблеми різних зайвих міжпроцесних взаємодій, які притаманні повністю мікросервісним рішенням.

Окремі сервіси, що потребують високої продуктивності або специфічної логіки, реалізовані як мікросервіси на Python із FastAPI. Наприклад, обробка аудіо-файлів для вбудовування або зчитування водяних знаків винесена в окремий компонент. Комунікація між ядром і мікросервісом здійснюється через REST API, а іноді — з використанням черг повідомлень або асинхронних викликів. Така побудова забезпечує масштабованість і дозволяє оновлювати або перезапускати окремі частини системи без впливу на основний застосунок.

Зв'язок між користувачем і серверною частиною забезпечується мережею Інтернет або локальною мережею. Веб-браузер виконує роль клієнта, що надсилає HTTP-запити та отримує у відповідь HTML-документи або дані у форматі JSON. Завдяки розділенню логіки між фронтендом та сервером, користувач бачить тільки кінцевий результат — адаптовані сторінки, списки контенту, аудіоплеєр і т.і.

Перевага модульного моноліту з елементами мікросервісів полягає у можливості масштабування системи частково, уникаючи повного переходу до

мікросервісної архітектури. Основні бізнес-функції залишаються в одному кодовому просторі, що полегшує розробку, дебаг, CI/CD, а також пришвидшує роботу над єдиним репозиторієм. У той же час, незалежні сервіси можуть розроблятися паралельно, маючи власні життєві цикли та технічні вимоги.

2.1.2 Проєктування компонентів системи

З урахуванням особливостей обраного технічного стеку, найкращим варіантом серед існуючих парадигм програмування стане модульне. Ця методологія, полягає в розбитті програми на незалежні, логічно завершені частини — модулі. Кожен модуль відповідає за окрему частину функціональності, має чітко визначений інтерфейс і може розроблятись, тестуватись і повторно використовуватись окремо від інших.

Однак попри відсутність суто об’єктного підходу, його використання (структури, інтерфейси та пакети є повними замінниками класів в Go) все ще може бути актуальним, оскільки часто з'являється потреба в певних специфічних типах даних. У випадку великої кількості вебзастосунків вони використовуються для представлення особливих моделей даних, які своїм структуруванням інформації полегшують організацію та взаємодію з базами даних, підтримку цілісності інформації та масштабування проєкту в подальшому.

Відповідно до поставлених вимог, та специфік проєкту було створено декілька основних моделей сутностей для відображення інформації та зберігання їх у базах даних.

Однією з базових таких є account. Дана сутність представляє собою відображення облікового запису користувача з основною інформацією про нього. Вона містить такі поля:

- Id, себто унікальний ідентифікатор користувача для подальших взаємодій з ним.

- Username відповідно до своєї назви є іменем користувача, який використовується для його ідентифікації та логіну.
- Email є додатковим полем. В майбутньому використовуватиметься для підтвердження ідентифікації та зворотного зв’язку.
- Bio – загальна інформація про користувача, його біографія.
- Profile_pic – лінк на фото профілю користувача. За його відсутності відображається перша літера його юзернейму.
- Is_admin перевіряє, чи має користувач права адміністартора.
- is_artist відповідно до Is_admin перевіряє привілеї користувача, але в даному випадку це стосується того,чи є користувач виконавцем, або тим, хто постачає цифровий товар.
- Is_logged_in потрібне для перевірки, чи користувач залогінений. Так як в нашому випадку використовується JWT автентифікація для перевірки такої інформації, дане поле виконує дещо іншу роль. Воно необхідне для змін станів веб сторінки.

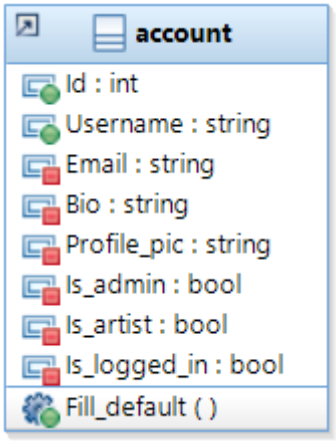

Рис. 2.1 – Модель account.

Наступною моделлю є comments, вона відповідає за відображення коментарів на сторінці продуктів. Серед її полів є наступні:

- UserID – зовнішній ID користувача. Використовується для підв’язування коментаря власнику.
- Username – ім’я користувача, що коментує.

- ProductID – ідентифікатор продукту, до якого додають коментар.
- Comment – основне наповнення коментаря.
- CreatedAt – час, коли коментар було додано.
- Profile_pic – посилання на фото профілю користувача.

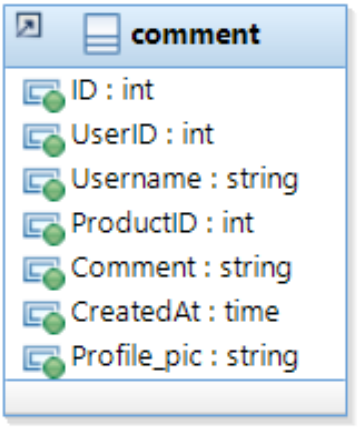

Рис. 2.2 – Модель comment.

Наступним на черзі йде product. Ця модель призначена для зберігання інформації про певний цифровий продукт. Нижче наведено інформацію про її поля.

- ItemID – Ідентифікатор пов’язаного ресурсу (наприклад, аудіофайлу або продукту).
- Name – Назва продукту (наприклад, "Chill Piano Loops" або "808 Bass Pack").
- Price – Ціна продукту.
- Type – Тип продукту: audio, midi або sample.
- Description – Опис продукту (характеристики, особливості, стиль тощо).
- Owner – Ім’я користувача, який виставив продукт на продаж.
- Genre – Музичний жанр продукту (наприклад, "lo-fi", "trap", "ambient").
- ImageURL – Посилання на зображення (наприклад, обкладинку продукту).

- CreatedAt – Дата та час створення продукту в системі.

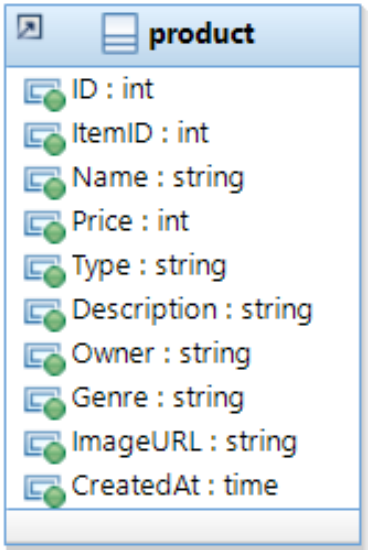

Рис. 2.3 – Модель product.

Для того, щоб інформацію виводити користувачеві на екран, також передбачається окрема сутність-шаблонізатор. У випадку даного проекту такий екземпляр називається RenderTemplate, і має наступні елементи:

Атрибути:

- tplCache: map[string]*pongo2.Template – асоціативний масив, де ключем є назва шаблону, а значенням – скомпільований шаблон Pongo2; використовується для кешування шаблонів і зменшення витрат на їхнє повторне зчитування з диска при кожному запиті
- tplCacheMtx: sync.RWMutex – м’ютекс з розділенням прав на читання і запис, дозволяє безпечно працювати з tplCache у багатопотоковому середовищі, забезпечуючи паралельне читання і виключення на час запису
- TemplateData: map[string]interface{} – словник даних, що передаються в шаблон при рендерингу; використовується як контекст, у якому шаблон шукає змінні (наприклад, назви, ціни, повідомлення тощо)

Методи:

- RenderTemplate(c *fiber.Ctx, templateName string, data TemplateData): error – головна функція для рендерингу HTML-сторінки: спочатку перевіряє, чи шаблон вже є в кеші; якщо ні – читає його з файлової системи і додає в кеш; потім виконує шаблон із переданим контекстом (data) і повертає згенерований HTML як відповідь клієнту через Fiber; у випадку помилки повертає статус 500 з текстом помилки

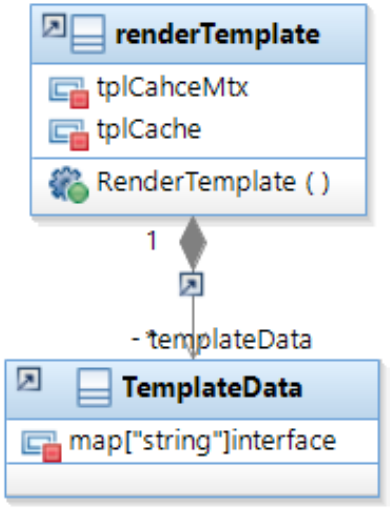

Рис. 2.4 – Шаблонізатор RenderTemplate.

З урахуванням модульної парадигми, яка використовується для написання маркетплейсу, зв’язки між її системними елементами дещо відрізняються від типових об’єктно-орієнтованих, і імплементуються на трохи вищому рівні абстракції, а саме на рівні компонентів. Щоправда невелика взаємодія між екземплярами класів усе ще передбачена. Вона передбачає взаємодію моделей з базами даних та шаблонізатором.

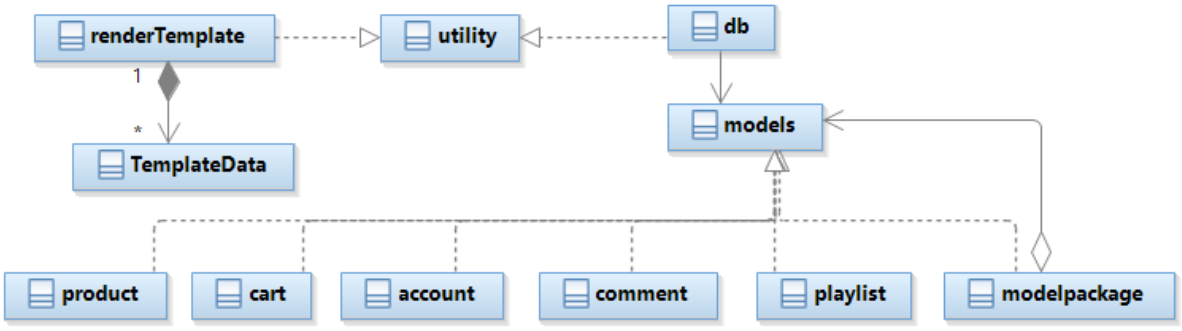

Рис. 2.5 – Загальна діаграма представлення класів в системі.

Як зазначено раніше, основна функціональна складова сайту імплементована на рівні компонентів (модулів, якщо враховувати мовні стандарти Go). В таком випадку використовується відповідна діаграма – компонентна. Це вид UML-діаграми, який використовується для візуалізації структури програмної системи на рівні її складових частин. Вона показує, з яких компонентів складається система, як вони пов’язані між собою та через які інтерфейси взаємодіють. У такій діаграмі кожен компонент зазвичай представляє окрему логічну частину (модуль, пакет, бібліотека, мікросервіс або будь-який інший елемент, що виконує певну функцію в межах системи). Для зручного відтворення інформації про кожен компонент системи наведено в таблиці 2.

Таблиця 2 – компоненти системи

№	Компоненти аналізу	Компоненти проектування	Опис
1	Керуючий компонент	Main	Головний компонент системи. Використовується, як пункт входу в програму перед запуском.
2	Менеджер сервісів	ServiceWorker	Ініціалізує всі основні обробники в системі та керує ними.

Продовження таблиці 2

3	Автентифікатор	Auth	Система керування автентифікацією. Виступає водночас і системою логіну, реєстрації, а також Middleware, які перевіряють привілеї користувачів.
4	Каталог	Catalogue	Керує виведенням всієї інформації про продукти та виконавців.
5	Панель адміністратора	Admin	Спеціальний компонент, який надає доступ до всіх баз даних, статистики та об’єктів користувачам з привілеєю адміністратора.
6	Уподобання	Likes	Керує системою вподобань користувачами продуктів та частково рейтингами їх постачальників.
7	Система керування продуктами	ProductManagement	Це спеціальний компонент, призначений для керування продуктами та інформацією про них, а також виступає допоміжною ланкою для каталогу.
8	Керування плейлистами	Playlist	Керує списками відтворення користувачів.
9	Керування файлами	FileTransfer	Спеціальний компонент, який реалізує взаємодію користувачів з об’єктним сховищем через посередництво сайту
10	Кошик	Cart	Взаємодіє з кошиками користувачів, додає та видаляє з нього.

Продовження таблиці 2

11	Аудіо плеєр	Player	Якщо завантаженим продуктом є аудіо, то дозволяє його програвати, транслюючи його з об’єктного сховища.
12	Шаблонізатор	RenderTemplate	Рендерить вебсторінки відповідно до запитів користувача.
13	Бази даних	db	Компонент, який дозволяє системі зв’язуватися з базами даних.
14	Моделі	Models	Інкапсулює різні моделі відображення для полегшення взаємодії з базами даних та структуризації коду.
15	API для водяних знаків	WatermarkAPI	Окремий від сайту мікросервіс, який накладає на завантажені аудіо спеціальним стеганографічним методом водяний знак для їх захисту.

Майже всі робочі компоненти, які відповідають за основну логіку, окрім каталогу та плеєра працюють за своєрідним посередництвом, як зазначено в таблиці. Для того, щоб запобігти несанкціонованому доступу до відносно чутливого функціоналу, передбачається запровадження специфічних middleware-валідаторів, таких, як LoginRequired та AdminRequired, відповідно для функцій, які потребують того, щоб користувач був залогінений, та мав привілеї адміністратора.

Також увесь основний функціонал, себто ті компоненти, які імплементують обробники, майже постійно використовують додаткові допоміжні utility та

інтерфейси, такі, як взаємодія з базою даних, рендеринг чи мікросервіси, якщо такі є в наявності.

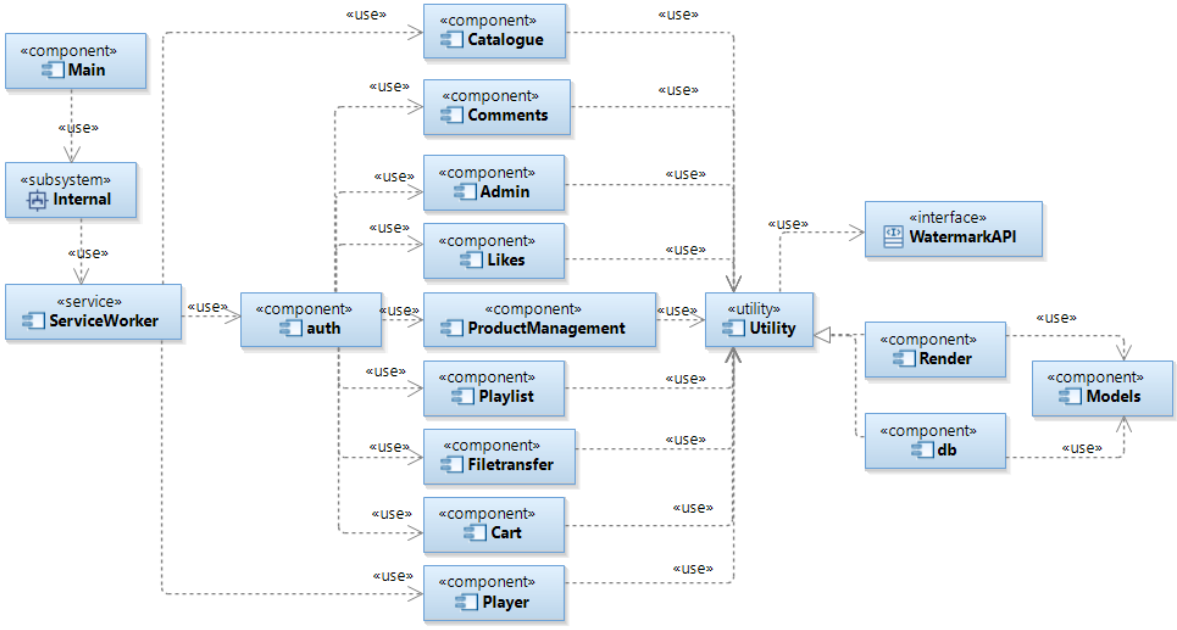

Рис. 2.6 – Діаграма компонентів системи.

Як можна побачити по діаграмі, система спроєктована, як типовий модульний моноліт, щоправда з можливістю також за потреби масштабувати його мікросервісами. Ключовою перевагою такої архітектури є її модульність. І цьому є низка причин:

- Якщо раптом зламається якийсь один компонент, то, стає легше відслідкувати та виправити певний баг чи вразливість.
- Стає набагато легше імплементувати новий функціонал без потреби кардинально переписувати вже наявний.
- Додаткові мікросервіси беруть на себе частину навантаження, яке може потенційно серйозно сповільнити основний сервер.

Завдяки такому підходу забезпечується висока гнучкість у розробці та підтримці системи. Можна відносно легко працювати над окремими модулями незалежно, що скорочує час на впровадження змін. Крім того, перехід за потреби до повноцінної мікросервісної архітектури в майбутньому не вимагатиме суттєвих перебудов.

2.1.3 Проєктування інфраструктури та розгортання системи

Для того, щоб маркетплейс виконував свою роботу повноцінно, перш за все необхідно його розгорнути на публічному хостингу. У випадку даного проєкту передбачається його розгортання на хмарному середовищі, утворивши певну своєрідну віртуальну інфраструктуру. Загалом у плані розгортання бере участь декілька вузлів, а саме:

- Локальне середовище – будь яка локальна робоча машина розробника, з якої розробляється проєкт та вносяться до нього зміни.
- CI/CD сервер – вузол, відповідальний за автоматичне збирання, тестування та розгортання готового програмного рішення у production.
- Artifact Repo – репозиторій, в якому зберігаються всі версії зібраних артефактів, готові для повноцінного розгортання.
- Cloud Instance – хмарний віртуальний сервіс, всередину якого розгортаються готові зібрані артефакти.
- Backend Container – конетйнер, що містить в собі основний компонент програмного рішення, а саме зібраний та запущений в production артефакт.
- WatermarkAPI Container – як і контейнер для бекенду, це вкладений вузол у хмарну віртуальну машину, який виконує роль додаткового мікросервісу. Також може слугувати прикладом інших мікросервісів, які можуть потенційно додатися в майбутньому.
- DB Instance та Object Repo Instance – додаткові вузли, які необхідні для роботи розгорнутого сервісу (для підтримки бази даних та об’єктного сховища). В залежності від стадії можуть змінювати свій стан (на стадії розробки працює одна база даних, на стадії тесту інша, і на стадії деплою ще інша, наприклад).

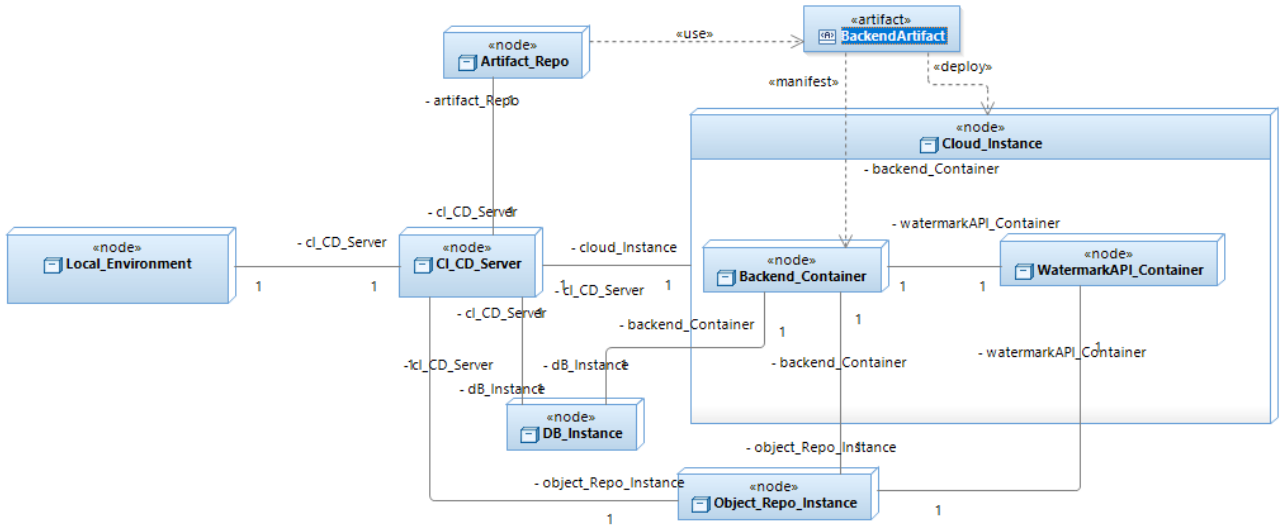

Рис. 2.7 – Діаграма розгортання сервісу.

Зважаючи на діаграму, зображену на рис. 2.7, можна приблизно передбачити сам процес розгортання та попередній варіант розгорнутої інфраструктури. З урахуванням масштабів даного проєкту, передбачено для початку розгортання невеликого кластеру на хмарному інстансі. Передбачається наступний порядок: розробник надсилає “коміти” на репозиторій, сервер CI/CD автоматично тригериться на ці зміни, та завантажує їх собі локально, проводить тести, за успішного завершення яких збирає знімок контейнера та надсилає його у сховище, після чого віддалено завантажує цей знімок на сервер, де й розгортає вже для повноцінної роботи.

Описаний вище підхід є достатнім для реалізації мінімально необхідної інфраструктури проєкту. В подальшому майбутньому хорошим варіантом буде розгортання більш ніж одного такого кластеру для забезпечення масштабованості сервісу, та підвищення його надійності.

2.2 Конструювання макретплейсу

Відповідно до спроектованої інфраструктури вебсайту тепер необхідно реалізувати її у програмному рішенні. Слідуючи умовам імплементації такої нестандартної архітектури, як модульний моноліт з елементами мікросервісів, перша версія такого сервісу буде поділена на два службові компоненти, а саме основний бекенд сервер, та API для водяних знаків. Процес конструювання відбуватиметься в контейнеризованому режимі, тобто кожна компонентна одиниця матиме власне ізольоване середовище, та окремо спілкуватиметься з рештою за допомогою TCP та HTTP протоколів.

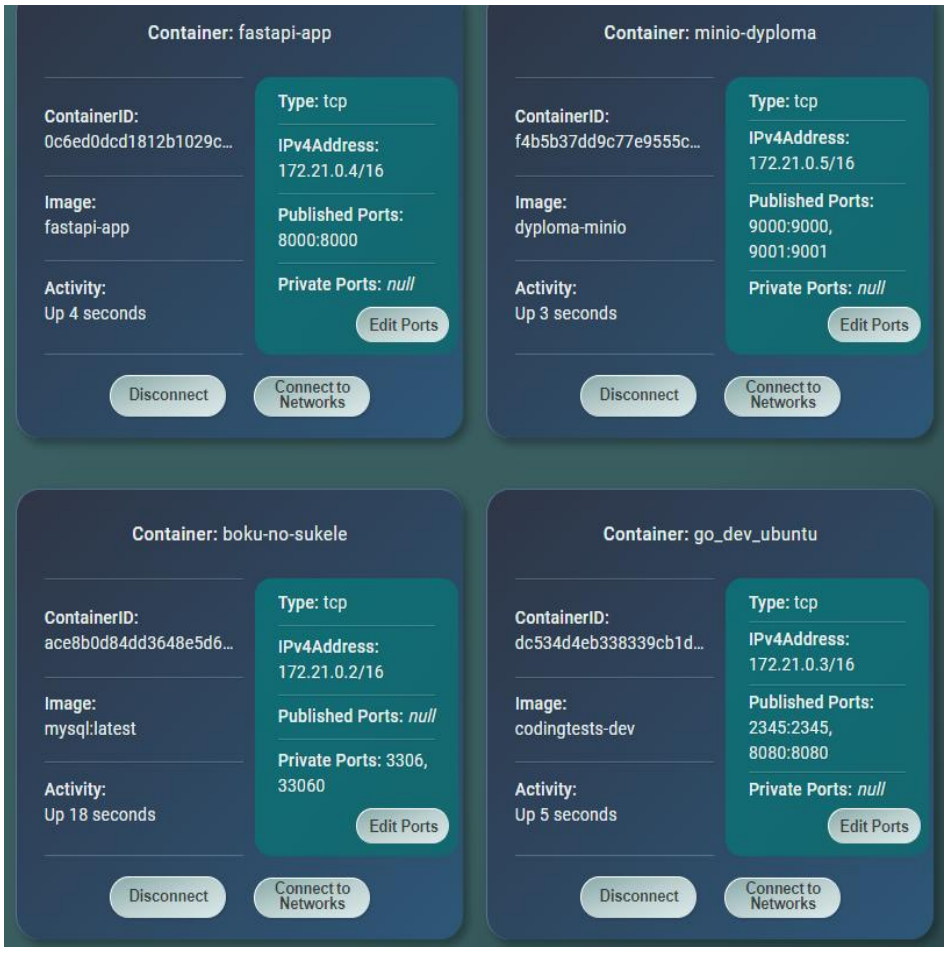

Рис. 2.8 – Локальна інфраструктура проєкту.

Як можна побачити на рис. 2.8, середовище взаємодії буде працювати за допомогою контейнеризації, а точніш із використанням Docker-контейнерів. На етапі конструювання таку взаємодію можна реалізувати повністю локально без потреби використовувати хмарні обчислювальні ресурси чи сховища. З наведених інфраструктурних елементів тут присутні два контейнери, які виконують роль сховищ (точніш об’єктного, такого, як MinIO, якщо локально, чи S3, якщо в хмарі, та бази даних, а саме MySQL), також контейнери з бекендом та мікросервісом, які мають тісно пов’язаний функціонал. З урахуванням того, що основними функціональними елементами є два останні, то саме їх імплементація і є основним завданням.

Для зберігання коду було використано Git Та Github. Робочий процес організовувався за підходом Trunk-Based Development, який передбачає роботу з однією основною гілкою (main) та короткоживучими відгалуженнями. Для кожного завдання спочатку створювалося issue, де описувалась суть роботи. Від цього issue створювалася окрема гілка з відповідною назвою (наприклад, feature/назва-задачі). У новоствореній гілці реалізовувались зміни, в які далі вносилися коміти з певними короткими оновленнями. Після завершення розробки проводилося локальне тестування.

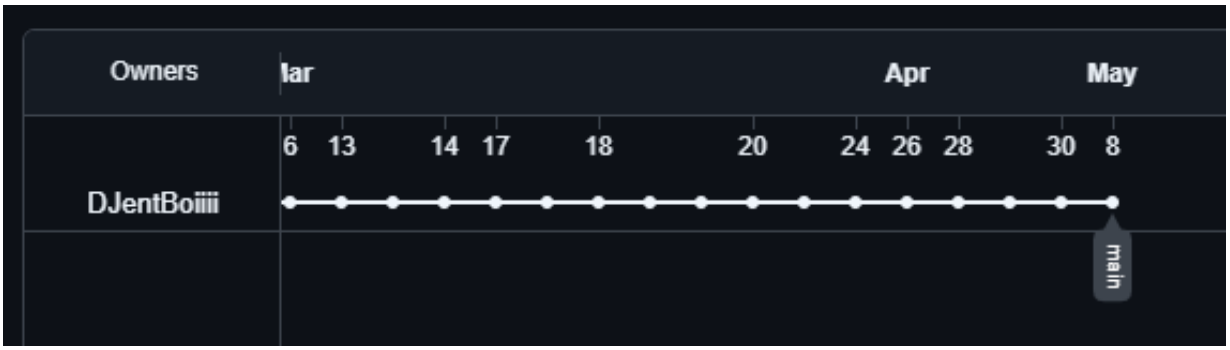

Рис. 2.9 – графічне представлення змін проєкту з підходом Trunk based development у GitHub.

Якщо по внесених змінах сайт працював належним чином, то далі створювався pull request у гілку main, де ще раз переглядалися зміни перед злиттям. Як тільки зміни верифікуються, відбувається злиття гілок в основну, при тому гілка, в яку передавалися зміни з локальної машини (див. рис. 2.9), видаляється.

2.2.1 Конструювання серверної частини

Основною мовою програмування для розробки бекенд складової було обрано Go з використанням фреймворку Fiber. Частково такий підхід чимось схожий на імплементацію вебсервісів за допомогою Flask чи Django для Python, однак з певними власними особливостями. Враховуючи те, що Go не застосовується широко для написання застосунків на кшталт вебсайтів, її технічні можливості ідеально підходять для цього, по кількох причинах, а саме:

- Go є компільованою мовою програмування, що робить виконання нею завдань набагато швидшим.
- Синтаксис цієї мови є строго структурованим та легко читабельним.
- Ключовою особливістю фреймворку Fiber є його швидкість, яка майже нічим не поступається Express.js.
- За замовчуванням всі маршрути у Fiber працюють на основі “горутин”, себто своєрідної легковагової паралельності. В такому разі єдиними обмежувачами стають запити до зовнішніх сервісів.

Завдяки загальноприйнятим стандартам Go, структура проєкту на базі цієї мови вже сама по собі просуває модульність, як парадигму. Кожна її архітектурна одиниця на рівні модулів є незалежним елементом, який, однак, тісно пов’язаний з усією архітектурою.

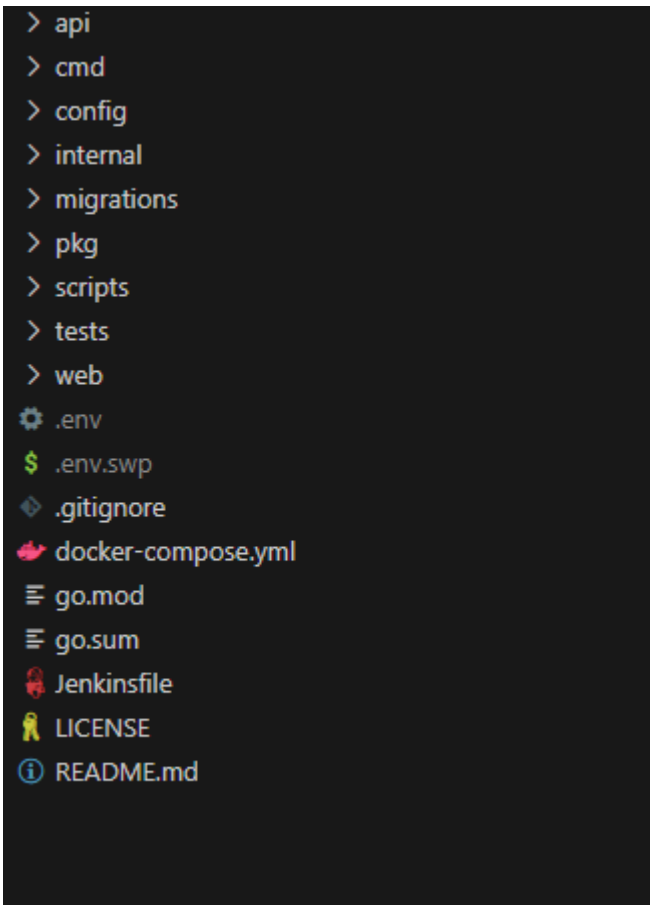

Рис. 2.10 – Структура проєкту на мові Go [12].

В цілому структура проєкту на цій мові складається з кількох основних елементів, чітко розподілених відповідно до виконуваних ними завдань. Серед таких структурних одиниць є наступні:

- api. Це директорія, яка імплементує API складову проєкту, та містить логіку для взаємодії інших сервісів з цим на, за допомогою, власне API інтерфейсу.
- cmd відповідає за керуючу логіку проєкту. Такою логікою може бути ініціалізація запущеного сайту, його запуск з певною особливою конфігурацією, чи просто запуск.
- config містить в собі усю конфігураційну інформацію, як, наприклад, змінні середовища, конфігурація баз даних і т.і.
- В internal міститься вся основна логіка проєкту. У випадку даного проєкту тут знаходяться всі підмодулі, які відповідають за обробку запитів користувачів та комунікацію з мікросервісами. Вся логіка, що

знаходиться всередині internal є приватною, і може використовуватися тільки в межах проєкту.

- pkg виконує ту саму роль, тільки зосереджений на публічному використанні, тобто може бути імпортований іншими проєктами.
- scripts потрібен для використання автоматизованих скриптів для, наприклад, розгортання системи, запису логів і т.і.
- web використовується в практиках, коли до проєкту потрібен зовнішній вигляд. В контексті даного проєкту він використовується для зберігання HTML, CSS та JavaScript складової.

В даному випадку ключова логіка розташована в пакеті `internal` яка, відповідно до архітектури компонентів (рис. 2.6), являє собою підсистему всіх основних модулів, які в свою чергу імплементують основні обробники запитів.

Найголовнішим таким модулем є service.go − центральна точка входу для налаштування та запуску сервісу. Цей модуль забезпечує основну інфраструктуру для роботи всього додатку, включаючи ініціалізацію вебдодатку, налаштування обмежень, підключення обробників маршрутів та запуск сервісу. Він є критично важливим для забезпечення стабільної роботи системи. Його функціонал включає:

- Створення HTTP-сервер за допомогою фреймворку Fiber, що дозволяє обробляти запити з високою продуктивністю.
- Встановлення ліміту на розмір тіла запиту (50 MB), що запобігає перевантаженню сервера великими запитами.
- Додавання маршруту /static, який дозволяє доступ до статичних ресурсів, таких як CSS, зображення та JavaScript.

Також service.go відповідає за обробку статичних файлів, що дозволяє зменшити навантаження на сервер, оскільки ці файли обслуговуються безпосередньо.

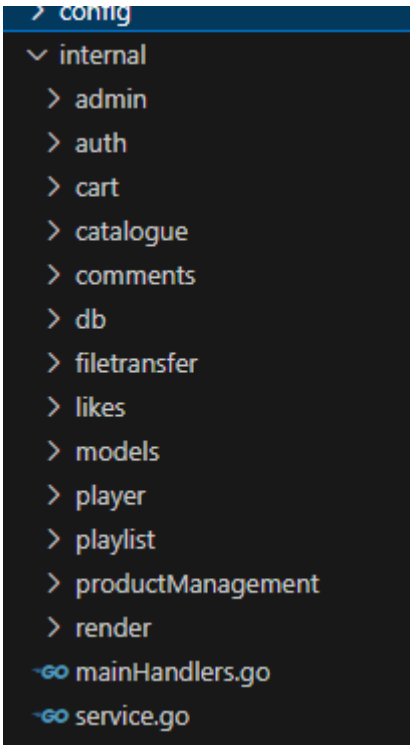

Рис. 2.11 – Структура підсистеми internal.

Пакет auth відповідає за автентифікацію та авторизацію користувачів. Він забезпечує безпеку додатку, дозволяючи лише авторизованим користувачам доступ до певних функцій. Також включає функціонал для роботи з профілями користувачів та управління доступом. Серед його основних функцій:

- Реєстрація та логін користувачів. Після того, як користувач залогінився, система створює та надає йому тимчасовий JWT токен з певним терміном дії. По його завершенню автоматично відбувається logout, себто вихід із системи.
- Профіль користувача. дозволяє переглядати та редагувати профіль, включаючи зміну пароля та інших особистих даних.
- Спеціальні middleware-валідатори, такі, як LoginRequired та AdminRequired забезпечують захист від небажаного доступу до основного функціоналу та панелі адміністратора. Саме через них проходить більша кількість маршрутів.

Модуль auth інтегрується майже з усіма іншими всередині підсистеми, оскільки доступність функціоналу залежить насамперед від нього.

```
hashed_password := hash_pwd(password)

if dbPassword != hashed_password {
    return c.Status(400).SendString("Неправильний пароль")
}

token := jwt.NewWithClaims(jwt.SigningMethodHS256, jwt.MapClaims{
    "user_id":  id,
    "username": username,
    "email":    dbEmail,
    "is_admin": IsAdmin,
    "exp":      time.Now().Add(time.Hour * 72).Unix(),
})

tokenString, err := token.SignedString([]byte(config.JWT_SECRET))
if err != nil {
    return c.Status(500).SendString("Помилка генерації токена")
}

c.Cookie(&fiber.Cookie{
    Name:  "jwt",
    Value: tokenString,
})

fmt.Println("User logged in")
return c.Redirect("/")
```

Рис. 2.12 – Код, що реалізує JWT токени для зберігання логіну.

Наступним на черзі є admin – модуль, що надає функціонал для адміністраторів, дозволяючи їм контролювати ключові аспекти системи. Він включає інструменти для управління користувачами, продуктами та перегляду статистики.

- Управління користувачами: дозволяє адміністраторам видаляти, оновлювати та переглядати список користувачів.
- Управління продуктами: забезпечує можливість видалення продуктів та перегляду їх списку.
- Статистика: надає адміністраторам доступ до інформаційної панелі зі статистикою, що дозволяє аналізувати активність користувачів та ефективність системи.

Даний модуль використовується окремо, і виключно для менеджменту наповненням сайту. Отримати права адміністратора можна виключно зміною значень в базі даних. За замовчуванням існує тільки один обліковий запис адміністратора.

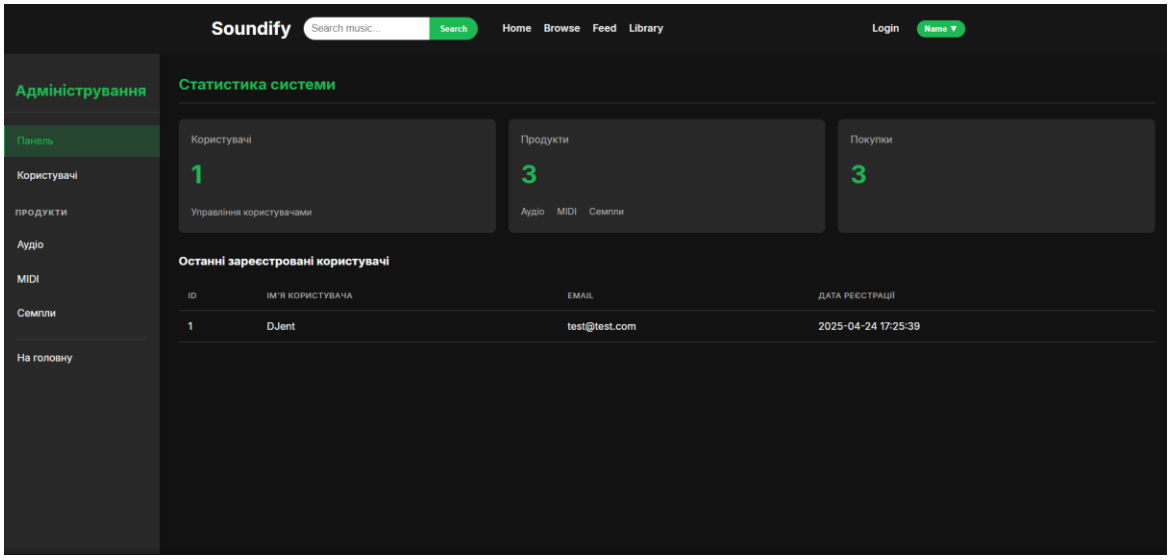

Рис. 2.13 – Панель адміністратора з використанням відповідного модулю.

Модуль playlist відповідає за функціонал, пов’язаний із плейлистами, дозволяючи користувачам створювати, редагувати та переглядати плейлисти, а також додавати або видаляти елементи з них. На етапі першої версії сайту його функціонал передбачає наступне:

- Створення та видалення плейлистів. Відповідно, дозволяє створювати та видаляти плейлисти.
- Додавання контенту до плейлистів та видалення з нього, надаючи змогу керувати списками відтворення.
- Можливість переглядати список плейлистів та їх деталі.

Списки відтворення повністю реалізують свій функціонал, коли це стосується аудіо товарів, інтегруючи сюди також програвач (в системі як player) для відтворення мультимедійного контенту.

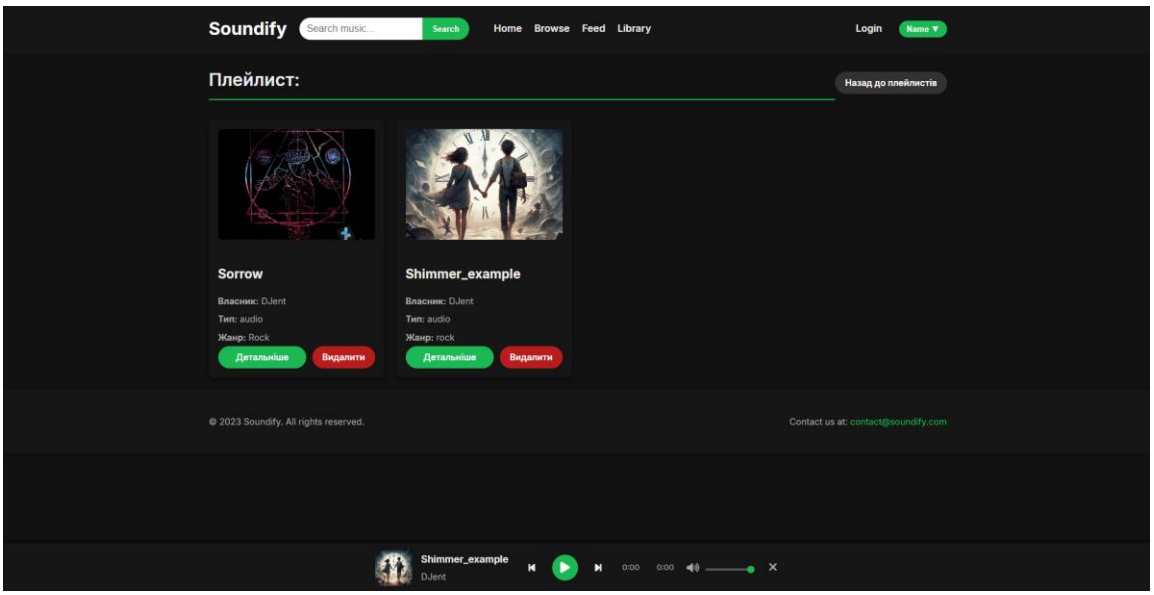

Рис. 2.14 – Імплементація плейлистів із вбудованим динамічним програвачем.

Для програвання треків спершу отримується інформація з об’єктного сховища, і в процесі отримання одразу передає потік користувачеві. В майбутніх версіях для зменшення навантаження на бекенд хорошим варіантом буде використання CDN, яке напряму через посилання з об’єктного сховища надсилатиме користувачеві інформацію, при тому не ризикуючи безпекою сховища[].

У випадку з маркетплейсом аудіо програвач буде особливо корисним, оскільки перед покупкою ліцензії на товар користувач матиме змогу прослухати його. Додавання елементів стрімінгових сервісів а згодом і рейтингів дозволить крім прибутку також збільшувати популярність виконавців. Ризики щодо потенційної крадіжки було враховано заздалегідь, тому ті треки, які програються, заздалегідь захищені спектральним водяним знаком, принцип роботи якого буде продемонстровано разом з описом конструювання зовнішнього API.

Наступним іде cart, який реалізує функціонал кошика, дозволяючи користувачам додавати товари до нього, переглядати їх, видаляти та завершувати покупки. Він є важливим компонентом для реалізації комерційної частини додатку. Як і для будь якого іншого кошика, він реалізує такий функціонал:

- Додавання продуктів: дозволяє користувачам додавати товари до кошика, зберігаючи їх у сесії або базі даних. Підтримує перевірку наявності товарів у наявності.
- Видалення продуктів: забезпечує можливість видалення товарів із кошика, оновлюючи загальну вартість у реальному часі.
- Перегляд кошика: відображає список товарів, які користувач додав до кошика, включаючи їх кількість, ціну та загальну суму.
- Завершення покупки: обробляє транзакції для завершення покупки, включаючи інтеграцію з платіжними системами (Stripe).

Cart інтегрується з такими модулями, як productManagement та auth для перевірки доступності товарів і авторизації користувачів. Також підтримує функціонал збереження кошика для незареєстрованих користувачів.

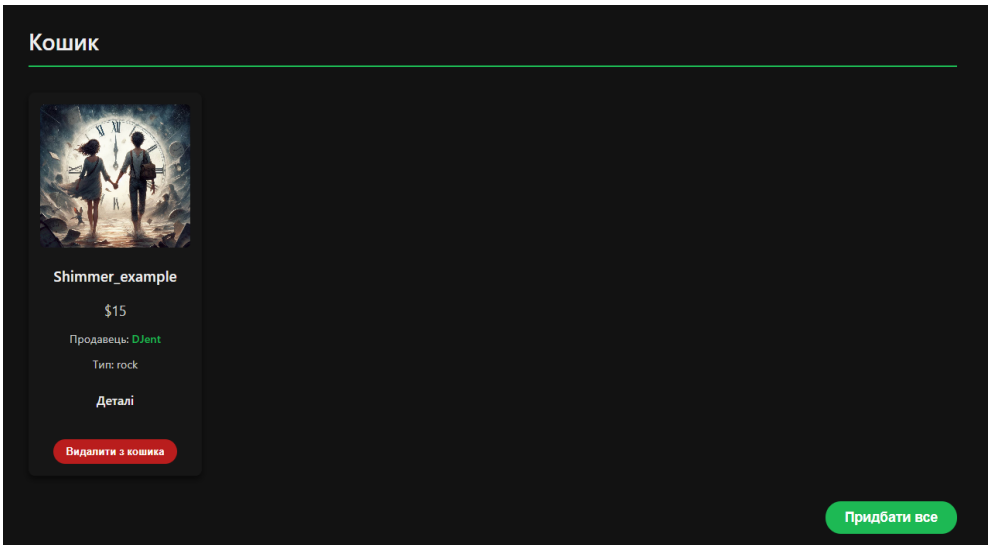

Рис. 2.15 – Зовнішній вигляд готової імплементації кошика.

Модуль catalogue відповідає за відображення каталогу продуктів. Цей пакет забезпечує зручну навігацію по продуктах, дозволяючи користувачам знаходити потрібні товари за різними критеріями. Він є основним інструментом для перегляду асортименту.

- Показ усіх продуктів: відображає товари, згруповані за типами, артистами чи продавцями. Підтримує пагінацію для зручності перегляду великої кількості товарів.
- Форматування назв: забезпечує зручне відображення назв типів продуктів, використовуючи стандартизовані шаблони.
- Фільтрація продуктів: дозволяє користувачам знаходити товари за певними критеріями, такими як ціна, категорія або популярність. Підтримує комбіновані фільтри для точного пошуку.

Catalogue інтегрується з модулями productManagement та likes для відображення популярних товарів і рекомендацій. Також підтримує функціонал сортування за рейтингом, ціною та новизною.

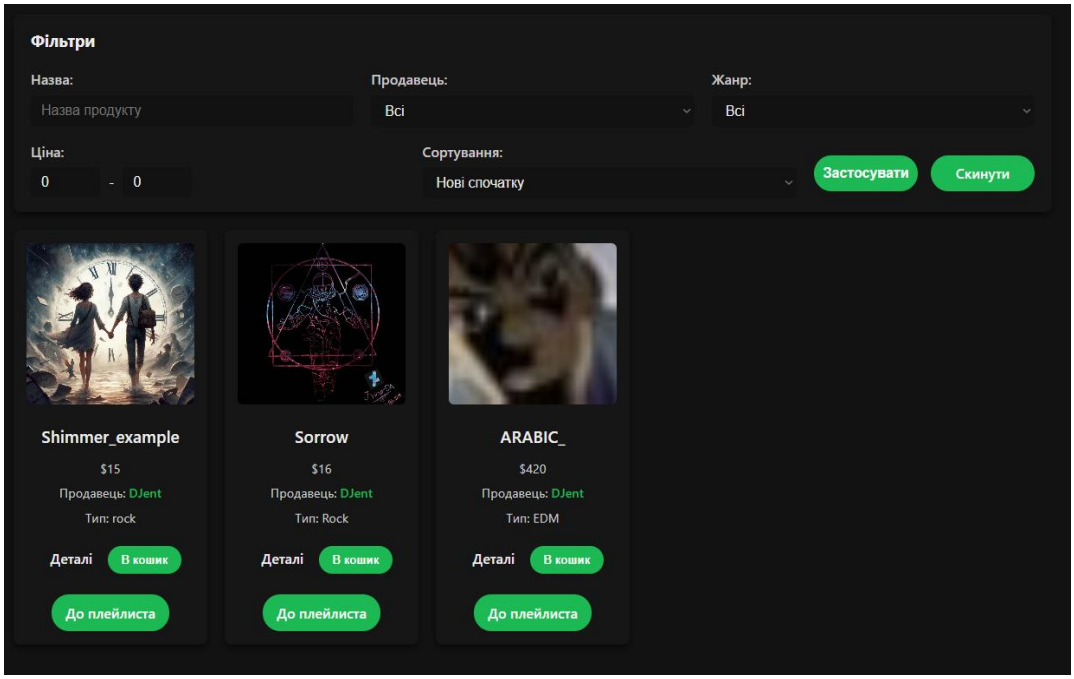

Рис. 2.16 – Готова імплементація каталогу.

Для обробки коментарів створено відповідний модуль – comments. Використовується тільки на сторінці інформації про продукт, де і надає свій основний функціонал, а саме:

- Додавання коментарів: дозволяє користувачам залишати відгуки до продуктів або контенту. Підтримує перевірку наявності нецензурної лексики.
- Видалення коментарів: забезпечує можливість видалення небажаних коментарів, як користувачами, так і адміністраторами.
- Отримання списку коментарів: відображає всі коментарі до певного продукту або контенту, включаючи дату, автора та рейтинг.

Модуль likes реалізує функціонал уподобань та разом з цим і статистики. Як і коментарі, система вподобань загалом працює суто в розділі детальної інформації про продукт, і реалізує наступний функціонал.

- Додавання та видалення уподобань: дозволяє користувачам оцінювати продукти, додаючи чи прибираючи уподобання. Підтримує миттєве оновлення кількості уподобань на сторінці.
- Кількість уподобань: показує загальну кількість уподобань для продукту, що дозволяє користувачам оцінити його популярність.

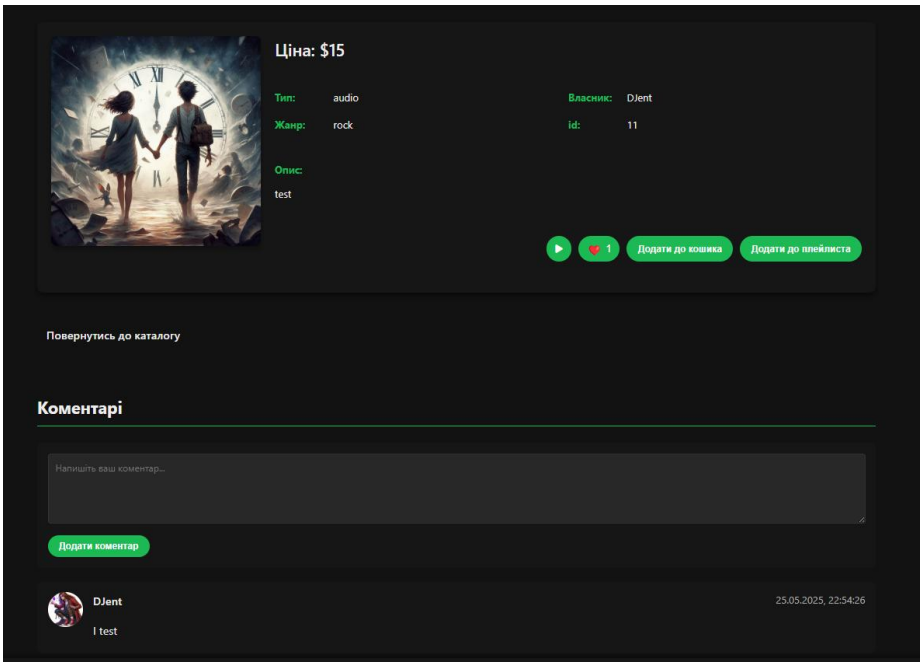

Рис. 2.17 – Сторінка деталей про продукт з коментарями та уподобаннями.

Одним з найважливіших компонентів системи є fileTransfer. Це спеціальний модуль, який реалізує взаємодію за посередництвом між об’єктним сховищем та користувачем.

```
exists, _ := MinioClient.BucketExists(c.Context(), bucketName)
if !exists {
    err := MinioClient.MakeBucket(c.Context(), bucketName, minio.MakeBucketOptions{})
    if err != nil {
        return c.Status(500).SendString("Не вдалося створити бакет")
    }
}

_, err = MinioClient.PutObject(
    c.Context(),
    bucketName,
    objectPath,
    bytes.NewReader(fileData.Bytes()),
    int64(fileData.Len()),
    minio.PutObjectOptions{ContentType: file.Header["Content-Type"][0]},
)
if err != nil {
    return c.Status(500).SendString("Помилка завантаження файлу")
}
filename := strings.TrimSuffix(file.Filename, filepath.Ext(file.Filename))
extension := filepath.Ext(file.Filename)

_, err = DB.Exec(
    "INSERT INTO Products (name, type, price, description, vendor, image_url, Extension, genre) VALUES (?, ?, ?, ?, ?, ?, ?, ?)",
    filename, typeVal, price, description, user.Username, imgDBPath, extension, genre,
)
```

Рис. 2.18 – Сторінка Upload, як приклад використання компоненту fileTrnasfer

Саме він відповідає за комунікацію з API для водяних знаків, організовує директорії у сховищі та відповідає за менеджемент, власне, самих файлів. З урахуванням особливостей предметної області, для нього передбачено наступне:

- Завантаження файлів користувачем. Система перевіряє тип цих файлів, і в залежності від нього міняє свою поведінку. Для MIDI та семплів вона організовує перевірку архівів на наявність зайвих файлів, в той час, як у випадку з аудіо вона надсилає його до сховища, та формує запит до зовнішнього API на накладання водяного знаку.
- Якщо користувач придбав відповідний продукт, то матиме змогу його завантажити.
- Адміністратор або користувач має змогу видалити завантажений ним на сервер файл. В такому разі він видалиться з баз даних, однак не з

об’єктного сховища, і люди, які придбали даний товар, мають все ще змогу його завантажувати, залежно від ліцензійних умов.

- Валідація файлів: перевіряє файли на відповідність вимогам, таким як розмір і формат

Останнім, та не менш важливим є productManagement, який обробляє функціонал, пов’язаний із продуктами. Цей модуль є основою для роботи з товарами в системі, забезпечуючи доступ до даних про продукти, їх фільтрацію та управління.

- Отримання даних про продукти: відображає інформацію про товари, включаючи їх опис, ціну та наявність.
- Фільтрація продуктів: дозволяє користувачам знаходити товари за певними критеріями, такими як ціна, категорія або популярність.
- Випадкові продукти: відображає випадкові товари для рекомендацій користувачам.
- Управління продуктами: забезпечує доступ до товарів за типами чи продавцями, дозволяючи адміністраторам і продавцям керувати своїми товарами.

Всі вищезгадані компоненти разом реалізують основний функціонал міжмодульної взаємодії всередині бекенду, однак у проєкті також передбачається взаємодія між мікросервісами, одним з яких є watermarkingAPI.

2.2.2 Конструювання API мікросервісу.

Технічний стек API відрізняється від основного бекенд компоненту, оскільки він більше зосереджений на обробці аудіо, а точніш на накладанні захисту на нього. Як тільки аудіо повністю завантажене на сервер, на API надходить запит на вбудовування повідомлення за допомогою методів аудіостеганографії. Для імплементації такого захисту було використано Python та

до нього фреймворк FastAPI, однією з ключових переваг якого є легкість створення швидких мікросервісів. Також було використано бібліотеку sciру, а точніш її модуль fft, який відповідає за швидкі перетворення Фур’є – спеціальний алгоритм , який швидко обчислює дискретне перетворення Фур'є (DFT). Воно перетворює сигнал із часової області (де можна побачити, як змінюється сигнал у часі) в частотну область (де видно, з яких частот складається сигнал). Якщо коротко, цей алгоритм дозволяє дізнатися, які частоти присутні в сигналі і з якою інтенсивністю, що дуже корисно для аналізу, обробки та модифікації сигналів, таких як звук чи зображення [14].

В даному випадку FFT (Fast Fourier Transform) необхідний для розкладання аудіо на частоти з метою модифікувати їх у подальшому за допомогою методу амплітудної модуляції, який в свою чергу представляє покращену версію методу LSB, оскільки є набагато стійкішим до модифікацій та стискання.

Створюваний API містить в собі 2 ендпойнта, а саме embed та extract, з відповідними обробниками. Для першого це embed_repeating_watermark. Він вбудовує водяний знак у аудіосигнал, працюючи з його частотними компонентами. Спочатку водяний знак, представлений у вигляді бітів, повторюється кілька разів за допомогою функції repeat_bits. Наприклад, якщо вхідний рядок бітів \u2014 "101", то після повторення з коефіцієнтом repeat_factor=3 він перетворюється на "111000111".

Аудіосигнал розбивається на блоки фіксованого розміру block_size, наприклад, 2048 семплів. Кількість блоків визначається як num_blocks = len(audio) // block_size. Водяний знак розподіляється по цих блоках, і кількість повторень водяного знаку в аудіо визначається як num_repeats = num_blocks // watermark_block_count, де watermark_block_count \u2014 це кількість бітів у повторюваному водяному знаку.

Для кожного блоку виконується швидке перетворення Фур'є (FFT), щоб перейти до частотної області. У спектрі блоку вибираються дві частотні компоненти, наприклад, bin_a=50 і bin_b=100. Їх амплітуди порівнюються, і залежно від значення біта водяного знаку (0 або 1) амплітуди змінюються. Якщо

біт дорівнює 1, то амплітуда bin_a збільшується відносно bin_b за допомогою коефіцієнта alpha. Якщо біт дорівнює 0, то навпаки, амплітуда bin_b збільшується.

```
if bit == 1 and a < b:
    spectrum[bin_a] = spectrum[bin_b] * alpha
elif bit == 0 and a > b:
    spectrum[bin_b] = spectrum[bin_a] * alpha
```

Лістинг 2.1 – Модифікація спектру відповідно до бітів.

Після цього зберігається симетрія спектру, щоб сигнал залишався реальним. Наприклад:

```
spectrum[-bin_a] = np.conj(spectrum[bin_a])
spectrum[-bin_b] = np.conj(spectrum[bin_b])
```

Лістинг 2.2 – Збереження симетрії спектру.

Модифікований спектр перетворюється назад у часову область за допомогою зворотного перетворення Фур'є (IFFT), і результат записується у відповідний блок аудіосигналу. Цей процес повторюється для всіх блоків, поки не буде вбудовано весь водяний знак. У результаті повертається модифікований аудіосигнал з вбудованим водяним знаком.

Функція починається з підготовки водяного знаку. Вхідний текст перетворюється у бінарний формат за допомогою функції text_to_bits. Наприклад:

```
text = "Hi"
watermark_bits = text_to_bits(text)
print(watermark_bits)  # Виведе: "0100100001101001"
```

Лістинг 2.3 – Конвертація повідомлення в біти для подальшого вбудування.

Потім кожен біт повторюється кілька разів для підвищення стійкості до спотворень:

```
repeated_bits = repeat_bits(watermark_bits, repeat=3)
```

```
print(repeated_bits)
#Виведе: "00011100011100000011111111100000111111111
```

Лістинг 2.4 – Первинний захист від спотворень.

Далі аудіосигнал розбивається на блоки фіксованого розміру. Це дозволяє працювати з кожним блоком окремо, вбудовуючи у нього частину водяного знаку. Наприклад, якщо розмір блоку дорівнює 2048 семплів, то сигнал довжиною 20480 семплів буде розбитий на 10 блоків:

```
block_size = 2048
audio = np.random.rand(20480)
num_blocks = len(audio) // block_size
print(num_blocks)  # Виведе: 10
```

Лістинг 2.5 – Розбиття сигналу на блоки.

Для кожного блоку виконується швидке перетворення Фур'є, що дозволяє перейти до частотної області. У цій області можна маніпулювати амплітудами частотних компонент. Наприклад:

```
block = audio[0:block_size]
spectrum = fft(block)
print(spectrum[:10])  # Виведе перші 10 частотних компонентів
```

Лістинг 2.6 – Перетворення блоків у частотні області за допомогою швидкого перетворення Фур’є.

Вибираються дві частоти, наприклад, bin_a=50 і bin_b=100. Залежно від значення біта водяного знаку змінюється співвідношення амплітуд цих частот. Якщо біт дорівнює 1, то амплітуда частоти bin_a збільшується відносно bin_b. Якщо біт дорівнює 0, то навпаки, амплітуда частоти bin_b збільшується. Це робиться так, щоб зміни були непомітними для слухача, але їх можна було легко витягти під час зворотного процесу.

Після маніпуляцій спектр перетворюється назад у часову область за допомогою зворотного перетворення Фур'є. Модифікований блок записується у вихідний аудіосигнал:

```
modified_block = np.real(ifft(spectrum))
audio[0:block_size] = modified_block
```

Лістинг 2.6 – Зворотнє перетворення Фур’є блоків та вмонтовування їх у аудіо.

Цей процес повторюється для всіх блоків, поки не буде вбудовано весь водяний знак. У результаті виходить аудіосигнал, який містить водяний знак, але при цьому звучить майже так само, як і оригінал (загалом різниця по відчуттях мінімальна).

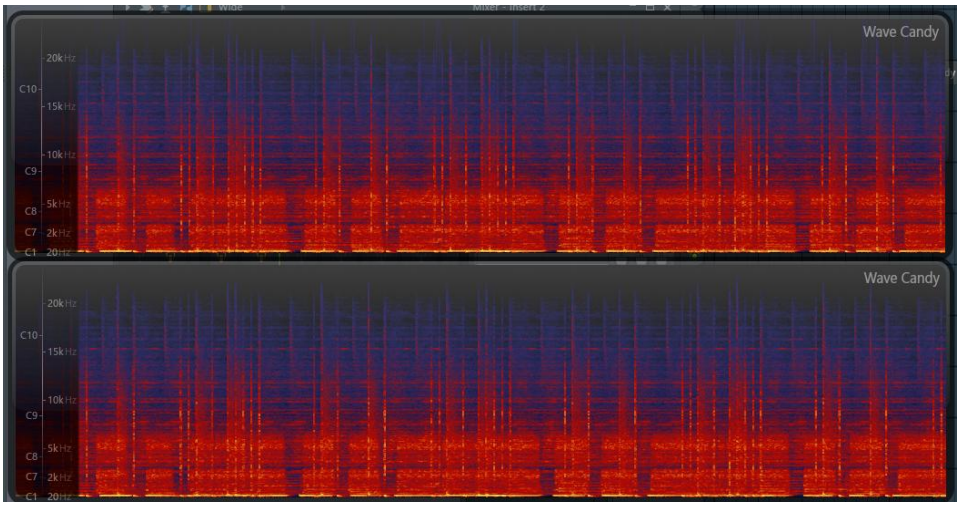

Рис. 2.19 – Спектрограми захищеного (зверху) та оригінального (знизу) аудіо.

Як можна побачити із спектрограм, водяний знак здебільшого накладається на частотах, які не є чутними людському вуху (16-20kHz), і може бути легко пошкоджений при спробі його прибрати, однак при шкоді водяному знаку нанесеться і шкода самому аудіофайлу, який в такому разі перестане бути придатним до споживання. Однак такий метод досить стійкий до стискання, при якому з WAV до MP3 (10 до 1), наприклад, відсоток того, наскільки буде пошкоджений водяний знак, не перевищуватиме 40% [15].

Рис. 2.20 – Отримання повідомлення водяного знаку з MP3 файлу.

Загалом, цього функціоналу достатньо для накладання первинного унікального захисту на аудіо та зчитування його сигнатури, проте не виключаючи подальшого покращення даної системи (до використання методів split spectrum чи спектральної графіки, наприклад). Ця інфраструктурна одиниця працюватиме окремо від основного серверу, і спілкуватиметься з ним лише у випадках обробки запитів на клонування та модифікацію аудіо продуктів.

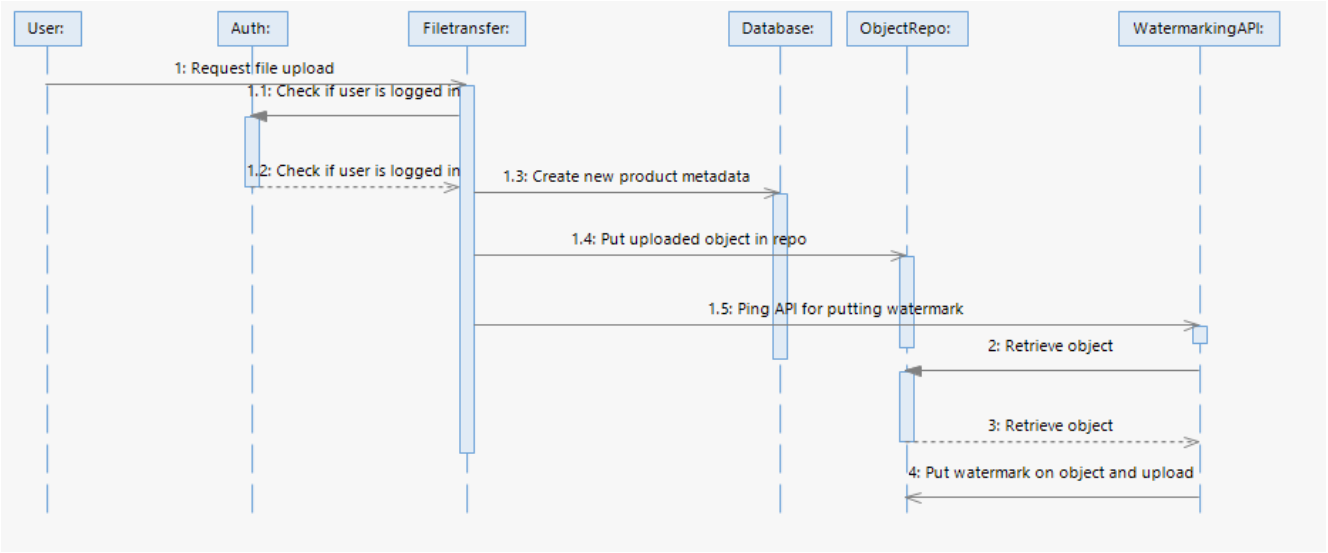

Рис. 2.21 – Послідовність виконання завантаження аудіо користувачем.

Як приклад виконання, можна взяти завантаження користувачем продукту в базу маркетплейсу. Відповідно до рис. 2.21 можна чітко побачити приклад взаємодії різних компонентів для отримання необхідного результату. Перед використанням будь якої функції сайту система спершу перевіряє, чи користувач

авторизований, після чого або користувач перенаправляєтсья на сторінку логіну, або продовжується виконання програми. В даному випадку після виконання користувачем запиту відбувається комунікація між модулями. Модуль, який відповідає за переміщення файлів створює метадані нового продукту та зберігає їх в базу даних, а сам завантажений об’єкт у репозиторій. Далі надсилається сигнал зовнішньому мікросервісу, з вимогою накласти прихований водяний знак у аудіо, після чого захищена версія файлу зберігаєтсья у спеціально виділеній директорії, звідки і транслюється на загал.

2.3 Тестування та розгортання системи.

По завершенню етапу конструювання, у життєвому циклі програмного забезпечення настає етап тестування. Цей етап є особливо важливим, оскільки саме на ньому перевіряється відповідність характеристикам сконструйованої системи до поставлених вимог. У випадку маркетплейсу використовуваними тестатим стали інтеграційні. Їхня перевага полягає в тому, що вони дозволяють перевірити правильність взаємодії між окремими модулями системи, що дає змогу виявити помилки, які не можна виявити при ізольованому тестуванні окремо кожного модуля. На етапі конструювання першої версії сайту було передбачено три основні категорії тестів, а саме авторизація, робота з продуктами та плейлистами.

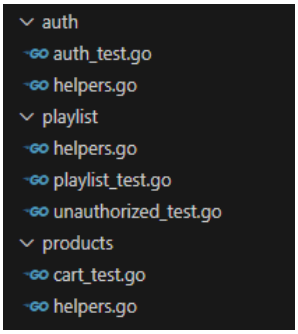

Рис. 2.22 – Структура тестів в модулі tests проєкту.

Тестування є важливим етапом розробки програмного забезпечення, що гарантує його стабільність, відповідність функціональним вимогам та якість. У цьому проєкті реалізовано широкий спектр тестів, які охоплюють ключові аспекти функціональності системи. Нижче наведено детальний опис усіх тестів, створених для перевірки модуля роботи з плейлистами.

Першим таким тестованим компонентом буде система аутентифікації. Її коректна робота є критичною для забезпечення безпеки всієї платформи, оскільки вона відповідає за контроль доступу користувачів до персоналізованого контенту, облікових записів, завантаження та купівлі аудіотреків. Далі наведено основні тести, реалізованої для поточної ітерації маркетплейсу.

1. Імітація запитів:

Опис компонентів, що взаємодіють: Тест TestLoginPage перевіряє взаємодію між маршрутом /login, HTML-відповіддю та елементами інтерфейсу.

Сценарії, які перевіряються:

- Імітація GET-запиту до сторінки входу.
- Перевірка наявності форми входу, поля для пароля та кнопки відправки.
- Аналіз HTML-відповіді для перевірки коректності структури сторінки. Очікувана поведінка системи: Сторінка входу містить усі необхідні елементи, такі як форма входу, поле для пароля та кнопка відправки.

Очікувана поведінка системи: Сторінка входу містить усі необхідні елементи, такі як форма входу, поле для пароля та кнопка відправки.

Очікуваний результат: Користувач бачить сторінку входу з усіма необхідними елементами.

2. Взаємодія з базами даних:

Опис компонентів, що взаємодіють: TestLoginSuccess перевіряє взаємодію між маршрутом /login, базою даних, системою авторизації та логікою обробки запитів.

Сценарії, які перевіряються:

- Імітація POST-запиту з обліковими даними користувача.
- Перевірка, чи система правильно ідентифікує користувача в базі даних.
- Перевірка, чи створюється сесія або токен для авторизованого користувача.
- Перевірка перенаправлення користувача на відповідну сторінку після успішного входу.

Очікувана поведінка системи: Користувач успішно авторизується, створюється сесія, а користувач перенаправляється на відповідну сторінку.

Очікуваний результат: Користувач потрапляє на головну сторінку після успішного входу.

3. Обробка недійсних полів.

Опис компонентів, що взаємодіють: Тест TestLoginFail перевіряє взаємодію між маршрутом /login, базою даних та логікою обробки помилок.

Сценарії, які перевіряються:

- Імітація POST-запиту з неправильними обліковими даними користувача.
- Перевірка, чи система правильно обробляє відсутність запису в базі даних.
- Аналіз HTML-відповіді для перевірки повідомлення про помилку. Очікувана поведінка системи: Система повертає статус 500, а у відповіді міститься повідомлення "Невірний логін або пароль".

Очікуваний результат: Користувач отримує повідомлення про помилку входу.

4. Керування власними полями.

Опис компонентів, що взаємодіють: Тест TestProfile перевіряє взаємодію між маршрутом /profile/{username}, базою даних, JWT-токеном та HTML-відповіддю.

Сценарії, які перевіряються:

- Імітація GET-запиту до профілю користувача з валідним JWT-токеном.
- Перевірка, чи система правильно отримує дані користувача з бази даних.
- Аналіз HTML-відповіді для перевірки коректності відображення профілю, включаючи ім'я користувача та біографію.

Очікувана поведінка системи: Система повертає коректну HTML-сторінку профілю з відображенням імені користувача, біографії та інших деталей.

Очікуваний результат: Користувач бачить свій профіль із правильною інформацією.

Наступною на черзі йде перевірка системи плейлистів. Попри те, що це окрема структурна одиниця, принцип її роботи подібний до більшості імплементованих компонентів, таких, як користувачі, коментарі та інші елементи, принцип дії яких зав’язаний на створенні списків, і додаванні до них (взаємодія “продукт до коментарів”, наприклад). До тестів, пов’язаних з плейлистами входить наступне:

1. Маніпуляція списками:

Опис компонентів, що взаємодіють: Тести TestRemoveFromPlaylist та AddToPlaylist взаємодію між маршрутом /playlist/remove, базою даних та логікою обробки запитів.

Сценарії, які перевіряються:

- Імітація POST-запитів на додавання або видалення елементів з плейлистів.
- Перевірка, чи система правильно ідентифікує елемент у базі даних.
- Перевірка, чи видаляється елемент із бази даних.
- Перевірка, чи відбувається перенаправлення користувача на сторінку плейлиста після успішного видалення.

Очікувана поведінка системи: Елемент успішно видаляється з таблиці, а користувач перенаправляється на сторінку плейлиста, або ж елемент додається до плейлиста, і користувача перенаправляє на сторінку каталогу.

2. Перевірка роботи прав доступу.

Опис компонентів, що взаємодіють: Тест TestPlaylistUnauthorized перевіряє взаємодію між маршрутом плейлистів, системою авторизації та логікою обробки запитів.

Сценарії, які перевіряються:

- Імітація спроби перегляду плейлиста, який належить іншому користувачу.
- Імітація спроби додавання елемента до плейлиста іншого користувача.
- Імітація спроби видалення елемента з плейлиста іншого користувача.
- Імітація доступу до списку плейлистів без автентифікації.

Очікувана поведінка системи: У всіх випадках система повертає статус 403 (Forbidden) для заборонених дій або перенаправляє користувача (статус 302) для неавторизованих запитів.

3. Покупка товарів.

Опис компонентів, що взаємодіють: Тест `TestPurchase` перевіряє взаємодію між системою кошика, базою даних, та логікою обробки замовлень.

Сценарії, які перевіряються:

- Імітація POST-запиту для оформлення покупки з даними про товари в кошику.
- Перевірка, чи система правильно обробляє дані з кошика та створює замовлення в базі даних.
- Перевірка взаємодії з платіжною системою для підтвердження транзакції.
- Перевірка очищення кошика після успішної покупки.

Очікувана поведінка системи: Система створює замовлення, підтверджує транзакцію та очищає кошик.

Очікуваний результат: Користувач отримує підтвердження успішної покупки, придбаний продукт з'являється в нього у відповідній базі з можливістю його завантажити без накладеного захисту.

```
ok      github.com/DjentBoiiii/marketplace/tests/playlist       (cached)
=== RUN   TestViewCart
--- PASS: TestViewCart (0.00s)
=== RUN   TestAddToCart
--- PASS: TestAddToCart (0.00s)
=== RUN   TestRemoveFromCart
--- PASS: TestRemoveFromCart (0.00s)
=== RUN   TestPurchase
--- PASS: TestPurchase (0.00s)
PASS
```

Рис. 2.23 – Результат виконання інтеграційних тестів в локальному середовищі.

Як можна побачити на рис. 2.23, швидкість виконання тестів займає мінімальну кількість часу (менше сотої секунди на тест), що вказує на високу швидкодію фреймворку. З цього можна зробити висновок, що єдиними чинниками, які потенційно дійсно сповільнюватимуть роботу сервісу, може бути або дуже великий трафік, або погано спланована архітектура з слабкою підтримкою (моніторинг та розгортання відбуваються суто з використанням людського ресурсу, наприклад). Для того, щоб запобігти таким ризикам, хорошим варіантом буде автоматизувати етапи тестування та розгортання.

Автоматизація процесів тестування та розгортання є ключовим етапом у забезпеченні стабільності, надійності та ефективності роботи програмного забезпечення, оскільки вона дозволяє мінімізувати людський фактор, автоматизувати рутинні завдання та забезпечити швидке виявлення і виправлення помилок.

Після пушу змін у гілку release автоматично тригериться Jenkins, який є потужним інструментом для автоматизації CI/CD процесів, що дозволяє інтегрувати, тестувати та розгортати зміни в автоматичному режимі. Jenkins підключається до репозиторію, отримує останні зміни та зберігає проєкт локально на своєму сервері, що забезпечує ізольоване середовище для виконання подальших етапів. Далі відбувається запуск процесу тестування, який включає виконання всіх тестів, що дозволяє перевірити коректність роботи кожного окремого компонента, їхню взаємодію між собою та відповідність системи загальним вимогам.

У разі успішного проходження тестів Jenkins переходить до наступного етапу, який передбачає створення Docker-знімків, що є контейнеризованими версіями серверної частини та API. Docker-знімки створюються з урахуванням усіх необхідних залежностей, конфігурацій та коду. Після створення знімків Jenkins завантажує їх у репозиторій артефактів, який слугує централізованим сховищем для зберігання всіх версій знімків.

Наступним етапом є розгортання сервісу на інстансі в AWS. Через Secure Shell (SSH) Jenkins підключається до віддаленого сервера, де виконується розгортання створених Docker-контейнерів. Для цього використовуються заздалегідь підготовлені скрипти, які забезпечують зупинку попередніх версій, очищення ресурсів, необхідних для їхньої роботи, та запуск нових контейнерів.

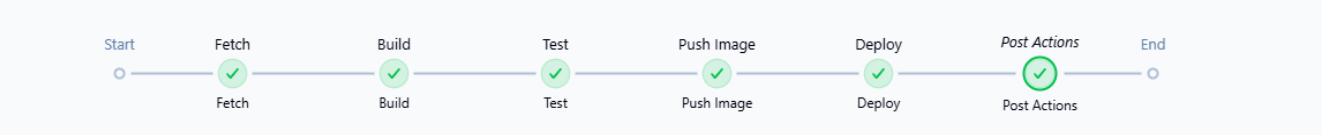

Рис. 2.24 – Приклад роботи Jenkins pipeline.

Після успішного розгортання Jenkins виконує фінальну перевірку доступності сервісу, яка включає тестові запити до API, перевірку відповіді сервера та аналіз логів для виявлення можливих проблем, що дозволяє переконатися в коректності роботи системи.

3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Таксонометрія небезпек

Таксономія небезпек – це класифікація та систематизація явищ, процесів, інформації, об'єктів, які здатні завдати шкоди [16, с. 11]. Для приведення прикладу таксономії небезпек можна зробити наступний поділ:

- за походженням (природні, техногенні, соціально-політичні, комбіновані);
- за локалізацією (космічні, атмосферні, літосферні, гідросферні);
- за наслідками (захворювання, травми, пожежі, загибелі, різні забруднення);
- за шкодою (соціальні, технічні, екологічні);
- за сферою прояву (у побуті, на виробництві, дорожньо-транспортні);
- за часом проявлення (кумулятивні, імпульсні);
- за характеристиками дії на людину (активні і пасивні).

Найбільш поширеним варіантом класифікації небезпек є ті, що діляться за походженням, які можна поділити на природні, техногенні, соціально-політичні, та комбіновані, перші три з яких належать людського оточення, а четверта до комбінацій різних елементів життєвого середовища.

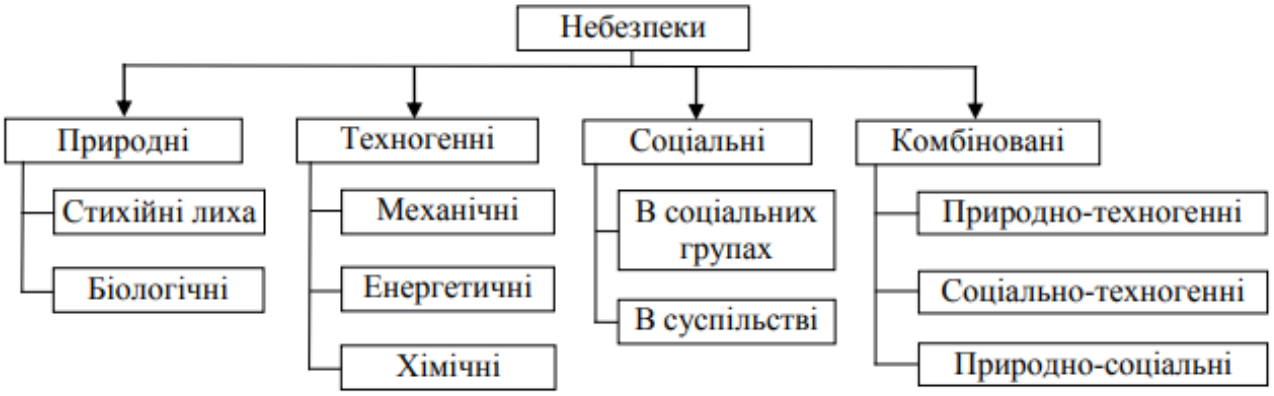

Рис. 3.1.1 – Поділ небезпек за джерелами походження [16, с. 13].

До природних джерел небезпеки належать будь які елементи природи, на кшталт стихійних лих чи інших природних явищ, які потенційно становлять небезпеку для життя чи здоров’я людини. Прикладом таких явищ є повені, вулкани, лавини, шторми, урагани, град, небезпечна фауна, різні заразні хвороби, бактерії чи віруси)

Техногенні небезпеки стосуються використання транспортних засобів, експлуатації підйомно-транспортного обладнання, різних легкозаймистих, горючих і вибухонебезпечних речовин. Також це стосується підприємств, що займаються діяльністю, пов’язаною з підвищеними температурами, тиском, електроенергією, концентрацією хімічних реочовин та випромінювання різних видів, таких, як радіація, електромагнітне випромінювання та ін.

Соціальні небезпеки напряму пов’язані з діяльністю людини, низьким соціальним, моральним, культурним та духовним рівнем суспільства. Такими проявами небезпеки може бути алкоголізм, тютюнопаління чи проституція. Причинами таких небезпек можуть бути погані умови проживання, конфлікти на різних рівнях чи страйки.

Вирізняються також політичні небезпеки. До них належать конфлікти на різних рівнях (міжнаціональний, міждержавний), гноблення, тероризм, геноцид, збройні конфлікти та війни.

Щодо комбінованих небезпек, то це можуть бути прояви небезпек, чинниками яких є поєднання декількох небезпек. До таких можуть входити:

- природно-техногенні (кислотні дощі, пустелі, зсуви, або інші явища, які так чи інакше пов’язані з людською діяльністю);
- природно-соціальні (інфекційні епідемії, венеричні хвороби, наркоманія);
- соціально-техногенні (порушення здоров’я, викликані виробничою діяльністю, вплив ЗМІ) [16, с. 14].

3.2 Проведення інструктажів з охорони праці

Під час прийняття на роботу та в процесі трудової діяльності працівники, а також учні, курсанти, слухачі й студенти під час проходження трудового або професійного навчання проходять на підприємстві за рахунок роботодавця інструктажі, навчання та перевірку знань з питань охорони праці, надання домедичної допомоги постраждалим у разі нещасних випадків, а також щодо правил поведінки у випадку виникнення аварійної ситуації [17, п. 3.1].

Інструктажі з охорони праці проводяться на всіх підприємствах, в установах і організаціях незалежно від їхнього профілю, підпорядкування чи форми власності. Їх основна мета — навчити працівників безпечному виконанню своїх трудових обов’язків без ризику для себе та оточення. Залежно від часу проведення та цільового призначення, інструктажі поділяються на такі види: вступний, первинний, повторний, позаплановий і цільовий.

Вступний інструктаж проводить інженер з охорони праці або інша відповідальна особа, призначена наказом. На підприємстві інструктаж організовується в кабінеті охорони праці або у спеціально облаштованому приміщенні відповідно до програми, складеної з урахуванням специфіки діяльності підприємства. Його проходять також працівники, які прибули у відрядження, а також учні та студенти, направлені на виробничу практику. Відомості про проведення інструктажу обов’язково фіксуються в спеціальному журналі. Під час вступного інструктажу працівникам пояснюють значення виробничої дисципліни, ознайомлюють з особливостями майбутньої роботи, основами трудового законодавства, вимогами до безпеки праці та правилами поведінки на робочому місці. Також розглядаються основні безпекові правила, порядок складання актів про нещасні випадки, вимоги до утримання робочих зон, правила використання індивідуальних засобів захисту, санітарні та гігієнічні норми, а також основи пожежної безпеки [18, с. 38].

Первинний інструктаж проводиться безпосередньо на робочому місці до початку виконання обов’язків. Його проходять: новоприйняті працівники (на

постійну чи тимчасову роботу), особи, переведені в інший підрозділ або на нову для них роботу, а також відряджені співробітники, які залучаються до виробничого процесу. Також первинний інструктаж проходять учні, студенти, курсанти й слухачі перед початком навчальної практики або перед виконанням кожного завдання, пов'язаного з використанням обладнання, інструментів чи матеріалів. Даний інструктаж проводиться індивідуально або для групи осіб однієї спеціальності на основі чинних інструкцій з охорони праці, відповідно до характеру роботи [18, с. 38].

Повторний інструктаж проводиться на робочому місці індивідуально або з групою працівників, які виконують однотипні роботи, з метою оновлення знань і навичок безпечного виконання трудових обов'язків. Він охоплює той самий обсяг питань, що й первинний інструктаж, і здійснюється з певною періодичністю залежно від характеру виконуваних робіт [18, с. 39].

Позаплановий інструктаж проводиться тоді, коли виникає потреба в додатковому навчанні працівників з питань охорони праці. Це може бути пов'язано зі змінами в нормативних документах, технологічному процесі чи обладнанні, або у разі порушення правил, що призвело до аварій чи травм. Також позаплановий інструктаж проводиться після тривалих перерв у роботі. Його зміст визначається індивідуально, залежно від конкретної ситуації, і може проводитися як з окремим працівником, так і з групою [18, с. 39].

У випадку ліквідації аварії або стихійного лиха чи проведенні робіт, на які оформляється певний наказ або розпорядження, проводиться цільовий інструктаж. Проводиться він і певною окремою групою працівників, чи окремим працівником. Обсяг і зміст інструктажу повністю залежить від виду виконуваної роботи [18, с. 39].

Первинний, повторний, позаплановий та цільовий інструктажі проводяться безпосереднім керівником робіт — начальником структурного підрозділу, майстром або фізичною особою, яка наймає працівників. Після завершення інструктажу обов'язково проводиться перевірка знань — у формі усного опитування чи з використанням технічних засобів. Також оцінюються практичні

навички щодо безпечного виконання робіт, що здійснює той, хто проводив інструктаж. Якщо результати перевірки знань, умінь або навичок після первинного, повторного чи позапланового інструктажу виявляються незадовільними, то протягом десяти днів необхідно провести додатковий інструктаж і повторну перевірку. У випадку, коли працівник не проходить перевірку знань після цільового інструктажу, він до виконання робіт не допускається, та не має права на повторну перевірку [19, с. 105].

Записи про проведення первинного, повторного, позапланового та цільового інструктажів, а також про допуск до роботи, здійснює особа, яка проводила інструктаж, у спеціальному журналі реєстрації. Обов'язковими є підписи як того, кого інструктують, так і того, хто проводить інструктаж. Журнал повинен бути пронумерований, прошитий та завірений печаткою [19, с. 105].

Проведення первинного, повторного, позапланового та цільового інструктажів із питань охорони праці фіксується в спеціальному журналі. Запис здійснює відповідальна особа, яка проводить інструктаж, із обов'язковим зазначенням підписів як працівника, що проходив інструктаж, так і особи, яка його проводила. Журнал має бути прошитим, пронумерованим і засвідченим печаткою. Якщо робота потребує оформлення наряду-допуску, запис про цільовий інструктаж робиться безпосередньо в цьому документі, а внесення його до журналу не є обов'язковим. Роботодавець визначає перелік посад та професій, які не підлягають повторному інструктажу. До таких можуть входити працівники, діяльність яких не пов'язана безпосередньо з використанням обладнання, машин, механізмів, інструментів, а також із зберіганням або обробкою матеріалів і сировини. Керівник підрозділу або сам роботодавець повинен надати працівникові інструкцію з охорони праці за його посадою або розмістити її безпосередньо на робочому місці [19, с. 105].

ВИСНОВКИ

У рамках цієї кваліфікаційної роботи бакалавра було реалізовано повноцінну архітектурно-технологічну основу для музичного веб-маркетплейсу, орієнтованого на сучасні вимоги цифрового ринку. Проєкт охопив усі ключові етапи створення системи — від дослідження предметної області до безпосередньої розробки та тестування функціонального рішення. Особлива увага приділялася тому, щоб створена платформа не лише задовільняла базові потреби музичної індустрії, а й мала високий потенціал до масштабування, адаптації та подальшого розвитку.

Під час аналізу ринку було вивчено поточну ситуацію в сфері онлайн-розповсюдження музики, проаналізовано проблеми піратства та недосконалості існуючих платформ у контексті захисту прав авторів. Це дозволило сформувати обґрунтовані вимоги до функціональності системи — з акцентом на модульність, безпеку, продуктивність і гнучкість інтеграції. Серед ключових компонентів: можливість аудіо-водяного знакування контенту, адаптивна система управління завантаженнями та надійна робота з великими обсягами медіа.

Архітектура застосунку була побудована за принципом модульного моноліту з елементами мікросервісної інтеграції. Основний бекенд створено за допомогою мови Go та фреймворку Fiber, що забезпечило високу швидкодію та стабільність. Для окремого мікросервісу обробки аудіо використано FastAPI на Python, зокрема для імплементації механізму цифрового водяного знаку. Завдяки такій структурі система легко масштабується як горизонтально, так і функціонально.

Важливою частиною проєкту стала побудова інфраструктури за допомогою хмарних рішень — AWS EC2, RDS, S3 (з підтримкою MinIO на етапі розробки) — та налаштування CI/CD-процесів через Jenkins, що гарантує стабільність і контроль якості на кожному етапі розгортання. Також було застосовано методологію Kanban для організації процесу розробки, яка дозволила гнучко реагувати на зміни та підтримувати постійний темп прогресу.

Результатом проєкту стала система, що може служити базисом для повноцінного музичного маркетплейсу, здатного до масштабування як для потреб окремих незалежних артистів, так і для реалізації комерційних моделей у рамках більших лейблів чи дистриб’юторів. Система вже сьогодні демонструє готовність до розширення функціональності та інтеграції нових сервісів — зокрема платіжних систем, інструментів аналітики та просування.

Завдяки сучасному технічному стеку, розробка показала практичну реалізацію навичок у сфері розробки вебзастосунків, проектування архітектури, безпеки, хмарної інфраструктури та мікросервісних підходів. В результаті було досягнуто основної мети — створити ефективне, стійке та масштабоване рішення для актуальних потреб музичної індустрії в умовах цифрової трансформації.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Beatstars official page. URL: https://www.beatstars.com (дата звернення: 13.06.2025).

2. Airbit official page. URL: https://airbit.com (дата звернення: 13.06.2025).

3. Go Documentation. Overview. URL: https://go.dev/doc/ (дата звернення: 13.06.2025).

4. Go Fiber docs. URL: https://docs.gofiber.io (дата звернення: 13.06.2025).

5. The Python Tutorial. Python Documentation. URL: https://docs.python.org/3.11/tutorial/index.html (дата звернення: 13.06.2025).

6. Learn FastAPI. Triangolo. URL: https://fastapi.tiangolo.com/#license (дата звернення: 13.06.2025).

7. Getting started with Docker. Introduction. URL: https://docs.docker.com/get-started/introduction/ (дата звернення: 13.06.2025).

8. Amazon Web Services documentation. URL: https://docs.aws.amazon.com/?nc2=h_ql_doc_do (дата звернення: 13.06.2025).

9. Octopus Deploy. What Is Jenkins and How Does it Work? Intro and Tutorial. Codefresh. URL: https://codefresh.io/learn/jenkins/ (дата звернення: 13.06.2025).

10. What is Kanban?. Atlassian. URL: https://www.atlassian.com/agile/kanban (дата звернення: 13.06.2025).

11. Trunk Based Development. URL: https://trunkbaseddevelopment.com (дата звернення: 13.06.2025).

12. Standard Go project structure. GitHub. URL: https://github.com/golang-standards/project-layout (дата звернення: 13.06.2025).

13. О. Дзюма, І. Мудрик. Дослідження систем тестування на основі розробленого інтернет сервісу потокового аудіо. Матеріали X науково-технічної конференції „Інформаційні моделі, системи та технології “Тернопільського національного технічного університету імені Івана Пулюя, ТНТУ, 2022. С.113.

14. Fast Fourier Transformation FFT - Basics. NTI Audio. URL: https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft (дата звернення: 13.06.2025).

15. Макоїд О. М. Застосування принципів LSB та Split Specter для захисту аудіо та шифрування даних всередині них. Природничі та гуманітарні науки. Актуальні питання, Тернопіль, Україна, 24–25 квіт. 2025. Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2025. С. 391.

16. В.В.Зацарний, Н.А.Праховнік, О.В.Землянська. Безпека життєдіяльності: конспект лекцій / НТУУ «КПІ». Київ : НТУУ «КПІ», 2016. 93 с.

17. Типове положення про порядок проведення навчання та перевірки знань з питань охорони праці: НПАОП 0.00-4.12-05 / затв. наказом Держнаглядохоронпраці України від 26.01.2005 р. № 15; із змінами згідно з наказом Держпраці України від 30.01.2017 р. № 140. – Офіц. вид. – Київ: Держпраці України, 2017. – 44 с.

18. Охорона праці : посібник / В. П. Кучерявий та ін. Львів : Оріяна-Нова, 2007. 369 с.

19. Мелех Л. В. Безпека життєдіяльності та охорона праці : навчальний посібник. Львів : Львівський державний університет внутрішніх справ, 2022. 219 с.

20. Методичні вказівки до виконання дипломної роботи освітнього рівня - бакалавр студентами усіх форм навчання для напряму підготовки 121 – Інженерія програмного забезпечення/ Укладачі : Петрик М.Р.,Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2016 – 28 с.

21. Дистанційний курс «Кваліфікаційні роботи бакалаврів» сайту дистанційного навчання ТНТУ [Електронний ресурс]. – Режим доступу: URL: https://dl.tntu.edu.ua/bounce.php?course=5329

ДОДАТКИ

Додаток А: Dockerfile для збирання знімку серверу

```
# syntax=docker/dockerfile:1

# Етап побудови
FROM golang:1.21.6-alpine AS builder

WORKDIR /app

# Кешування модулів
COPY go.mod go.sum ./
RUN go mod download

# Копіюємо решту коду
COPY . .
RUN go build -o /marketplace-server main.go

# Етап запуску
FROM alpine:latest

WORKDIR /app
COPY --from=builder /marketplace-server .

EXPOSE 3000

CMD ["./marketplace-server"]
```

Додаток Б: Dockerfile для знімку API

```
# Базовий образ з Python 3.10 та FFmpeg
FROM python:3.10-slim-bookworm

# Встановлення системних залежностей
RUN apt-get update && \
    apt-get install -y --no-install-recommends ffmpeg ca-
certificates tzdata && \
    rm -rf /var/lib/apt/lists/*

# Робоча директорія
WORKDIR /app

# Копіювання залежностей
COPY requirements.txt .

# Встановлення Python-залежностей
RUN pip install --no-cache-dir -r requirements.txt

# Копіюємо весь проєкт
COPY . .

# Відкриття порту
EXPOSE 8000

# Запуск сервера
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Додаток В: Публікація у науковій конференції

УДК 621.326
Макоїд О. – ст. гр. СП-42
Тернопільський національний технічний університет імені Івана Пулюя

ЗАСТОСУВАННЯ ПРИНЦИПІВ LSB ТА SPLIT SPECTER ДЛЯ ЗАХИСТУ АУДІО ТА ШИФРУВАННЯ ДАНИХ ВСЕРЕДИНІ НИХ

Науковий керівник: док.філософ. Мудрик І. Я.

Makoid O.
Ternopil Ivan Puluj National Technical University

APPLYING LSB AND SPLIT SPECTER PRINCIPLES FOR AUDIO PROTECTION AND DATA ENCRYPTION INSIDE AUDIO SIGNALS

Supervisor: Ph.D. Mudryk I.

Ключові слова: стеганографія, аудіозахист, спектральне кодування, приховування даних
Keywords: steganography, audio protection, spectral coding, data hiding

Аудіофайли є мало не найбільш вразливим видом цифрових даних, що часто піддається незаконному копіюванню, поширенню та дистрибуції. Одним з найбільш надійних варіантів захисту прав інтелектуальної власності для такого типу продуктів є застосування аудіостеганографії. Аудіостеганографія є процесом приховування інформації в аудіосигналі так, щоб її не можна було помітити неозброєним оком на спектрумі чи почути, але можливо витягнути при наявності відповідного ключа або алгоритму [1].

Одним з таких методі є LSB (Least Significant Bit), що змінює найменш значущий біт у 16-бітному PCM-сигналі. Таке втручання настільки незначне, що не сприймається слухом. У стереотреці з частотою 44.1 кГц за секунду можна приховати кілька кілобайтів даних без втрати якості. Метод ефективний для вбудовування ID-треків, цифрових підписів та DRM-токенів. Щоправда, даний принцип має великий недолік — чутливість до обробки сигналу, наприклад, стиснення, фільтрації або перекодування (перетворення WAV аудіо в MP3 чи FLAC, наприклад), що може знищити приховану інформацію або сильно спотворити її [2].

З метою подолання таких обмежень застосовується принцип Split Specter (розщеплення спектра), що передбачає перехід аудіосигналу в частотну область через швидке перетворення Фур'є (FFT). На відміну від LSB тут дані вбудовуються у малочутливі діапазони спектра — зазвичай інфразвук (20–200 Гц) або ультразвук (понад 16 кГц) Це забезпечує вищу непомітність та стійкість до обробки — фільтрації, перекодування, нормалізації тощо. Алгоритм охоплює:

зчитування WAV-файлу, сегментацію на фрейми, застосування FFT, модифікацію амплітуд або фаз цільових частот згідно з бітами повідомлення (наприклад, з використанням псевдовипадкової послідовності), зворотне перетворення (IFFT) і збереження результату. Такий підхід дозволяє інтегрувати дані, що зберігаються навіть після частих перетворень сигналу [3].

Гібридний алгоритм застосування LSB та Split Spectrum може експотенційно збільшити захист, оскільки це не тільки збільшення можливого об'єму прихованої інформації, але й стійкості до спотворень. Принцип такого шифрування передбачає:

1. Попереднє шифрування повідомлення за допомогою алгоритму AES або RC4;
2. Поділ за важливістю: службова інформація вбудовується в LSB, додаючи декілька біт між заголовком аудіо файлу та самим його наповненням;
3. Основне повідомлення (водяні знаки, ліцензії) у формі двійкового коду вставляється в спектр аудіо через спектральне кодування;
4. На завершення трек перевіряється на збереження прозорості звучання та стійкість до обробки;

Таке поєднання дозволяє ставити індивідуальні відбитки на кожен екземпляр файлу, легко ідентифіковувати та усунути джерело витоку, а також є надійним способом передавати конфіденційну інформацію, не привертаючи зайву увагу, тим самим гарантуючи ефективний первинний захист чутливим даним та інтелектуальній власності.

Однак варто врахувати, що накладання захисту на аудіо та його зчитування зазвичай займає деякий час для обробки, що , наприклад, для веб орієнтованих систем, є важливим фактором ефективності. В такому разі варто розглянути варіанти його масштабування. Одним із таких є використання хмарних технологій для зберігання аудіо (Amazon S3) та розподіленої асинхронної обробки через Redis Queue. Дані інструменти ідеально підходять для масштабування задач такої складності, особливо, коли це стосується великої кількості даних для обробки.

Список використаних джерел:

1. Sumit Kumar A. Audio Steganography : The art of hiding secrets within earshot. *Medium*. 17.06.2018. URL: https://sumit-arora.medium.com/audio-steganography-the-art-of-hiding-secrets-within-earshot-part-2-of-2-c76b1be719b3 (дата звернення: 09.04.2025).
2. *International Journal of Computer Trends and Technology (IJCTT)*. 2014. Т. 13, вип. 2. С.
3. M. Petryk, M. Bachynskyi, V. Brevus, I. Mudryk, D. Mykhalyk. Analysis technology of neurological movements considering cognitive feedback influences of cerebral cortex signals. ITTAP CEUR Workshop Proceedings 3309 (2022): 45–54.
4. Nadiia, Y., Nadiia, M., Lesia, O., & Mudryk I., (2023). RADIAL-BASIS NEURAL NETWORKS FOR ENTERPRISES ACTIVITY PREDICTION. European Science, 3 (sge17-03), 42–48. https://doi.org/10.30890/2709-2313.2023-17-03-012

Додаток Г: Посилання на репозиторій серверу

Додаток Д: Диск