

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Веб-технології»)

на тему:

«ВЕБСИСТЕМА ТА ДЕСКТОП-ДОДАТОК КОНТРОЛЮ ВЕРСІЙ МУЗИЧНИХ ПРОЄКТІВ»

Виконав студент групи ВТ-22-2
РЕГЕНЕЛЬ Тимофій Юрійович

Керівник роботи:
ФАНТ Микола Олександрович

Рецензент:
МОРОЗОВ Андрій Васильович

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: РЕГЕНЕЛЬ Тимофій Юрійович

Керівник роботи: ФАНТ Микола Олександрович

Тема роботи: «Вебсистема та десктоп-додаток контролю версій музичних проєктів», затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №92/с**

Вихідні дані для роботи: процес управління версіями та організації спільної роботи над музичними проєктами, що реалізується через розробку спеціалізованої системи контролю версій на базі веб- та десктоп-технологій для забезпечення цілісності та синхронізації даних.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Фант М.О.	02.03.2026р.	04.03.2026р.
2	Фант М.О.	10.04.2026р.	17.04.2026р.
3	Фант М.О.	15.05.2026р.	29.05.2026р.

РЕФЕРАТ

Пояснювальна записка до випускної кваліфікаційної роботи бакалавра містить 96 сторінок, 46 рисунків, 32 таблиці, 1 додаток та 30 використаних джерел.

Метою роботи є розробка алгоритмів та програмна реалізація комплексу, що складається з вебплатформи та віконного додатка, для управління, відстеження змін, синхронізації, централізованого збереження та контролю версій музичних проєктів.

У процесі роботи було спроектовано та реалізовано систему контролю версій, створено базову бібліотеку з її функціоналом, віконний додаток для взаємодії з локальними файлами та вебплатформу для керування проєктами і сховищами. Застосовано клієнт-серверну архітектуру для комплексу загалом, архітектуру «модульний моноліт» для серверної частини і MVVM для віконного додатка, механізми і протоколи для відновлюваної передачі даних (TUS, HTTP Range), а також сховище, адресоване за вмістом.

У результаті створено готовий програмний прототип, який дозволяє музикантам ефективно організовувати спільну роботу, зберігати версії проєктів, уникати конфліктів при злитті змін і синхронізувати дані на різних пристроях і сховищі проєкту, обраному користувачем.

КЛЮЧОВІ СЛОВА: ВЕБПЛАТФОРМА, ВІКОННИЙ ДОДАТОК, МУЗИЧНИЙ ПРОЄКТ, СИНХРОНІЗАЦІЯ, СИСТЕМА КОНТРОЛЮ ВЕРСІЙ.

					ІПЗ.КР.Б-121.26 - ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Вебсистема та десктоп-додаток контролю версій музичних проєктів	Літ.	Арк.	Аркуші
Розроб.		Регенель Т.Ю.						
Керівник		Фант М.О.					3	96
Реценз.		Морозов А.В.				ФІКТ Гр. ВТ-22-2		
Н. контр.								
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The explanatory note to the bachelor's thesis consists of 96 pages, 46 illustrations, 32 tables, 1 attachment, and 30 references.

The aim of this work is to develop algorithms and implement a software system – consisting of a web platform and a desktop application – for managing, tracking changes, synchronizing, centrally storing, and version controlling music projects.

During the project, a version control system was designed and implemented, a core library with its functionality was created, a desktop application for interacting with local files was developed, and a web platform for managing projects and repositories was built. A client-server architecture was used for the system as a whole, a «modular monolith» architecture for the server-side, and MVVM for the desktop application, along with mechanisms and protocols for resumable data transfer (TUS, HTTP Range), as well as a content-addressed storage system.

The result is a fully functional software prototype that enables musicians to effectively collaborate, save project versions, avoid conflicts when merging changes, and synchronize data across different devices and the project storage of their choice.

KEYWORDS: DESKTOP APPLICATION, MUSIC PROJECT, SYNCHRONIZATION, VERSION CONTROL SYSTEM, WEB PLATFORM.

		Регенель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				4
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ВСТУП.....	6
РОЗДІЛ 1. АНАЛІЗ НАПРЯМІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ КОНТРОЛЮ ВЕРСІЙ МУЗИЧНИХ ПРОЄКТІВ	8
1.1 Постановка задачі	8
1.2 Аналіз аналогів програмного продукту	11
1.3 Вибір архітектури для розроблюваної системи	17
1.4 Обґрунтування вибору засобів реалізації та вимоги до апаратного забезпечення	21
Висновки до першого розділу	24
РОЗДІЛ 2. ПРОЄКТУВАННЯ КОМПОНЕНТІВ СИСТЕМИ.....	25
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	25
2.2 Розробка бази даних системи.....	33
2.3 Проектування та реалізація алгоритмів роботи системи	47
2.4 Реалізація компонентів системи	57
Висновки до другого розділу	65
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБСИСТЕМОЮ ТА ВІКОННИМ ДОДАТКОМ.....	66
3.1. Порядок встановлення та налаштування системи	66
3.2. Структура інтерфейсів клієнтських додатків.....	69
3.3. Тестування роботи програмного комплексу	85
Висновки до третього розділу.....	88
ВИСНОВКИ	89
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	91
ДОДАТКИ	94

ВСТУП

Актуальність теми зумовлена браком інструментів та програмних продуктів, що спеціалізуються на контролі версій музичних проєктів, а також низькій пристосованості вже наявних систем контролю версій загального призначення до специфіки роботи з музичними проєктами.

Потреба у версіонуванні музичних проєктів виникає регулярно, та зростає зі збільшенням команди, масштабів проєктів та розвитком програмного забезпечення в цій сфері. На тематичних форумах та у запитах до розробників програмного забезпечення музиканти регулярно висловлюють потребу в функціоналі контролю версій для музичних проєктів.

Процес реалізації такої системи залучає низку аспектів, таких як структури даних та алгоритми над ними, оптимізація використання ресурсів, контентно-адресовані (content-addressed) сховища, хешування, відновлювану передачу даних мережею та масштабовану архітектуру програмних систем. Дослідження та напрацювання в цих напрямках позитивно впливають на якість кінцевих програмних продуктів.

Метою випускної кваліфікаційної роботи бакалавра є розробка системи контролю версій музичних проєктів, а також віконного додатка та вебплатформи на її основі.

Завдання роботи для досягнення поставленої мети до виконання визначено у вигляді наступного переліку:

- провести порівняльний аналіз існуючих рішень, виокремити ключовий функціонал для системи, що розробляється;
- здійснити аналіз та вибір засобів для реалізації;
- спроектувати архітектуру програмного забезпечення та взаємодії компонентів;
- розробити систему контролю версій. На її основі розробити віконний додаток, вебплатформу та інтегрувати їх між собою і набором підтримуваних сховищ.

		Регенель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				6
Змн.	Арк.	№ докум.	Підпис	Дата		

Методами аналізу та інформаційною базою дослідження є метод порівняльного аналізу існуючих аналогів та метод об'єктно-орієнтованого проєктування, наукові публікації, офіційна технічна документація та стандарти проєктування розподілених систем.

Об'єктом дослідження є автоматизація процесів збереження, відстеження змін та відновлювана передача даних.

Предметом дослідження є архітектурні рішення та програмні засоби реалізації гібридної системи контролю версій для музичних проєктів.

Практичне значення отриманих результатів. Створена система може бути використана незалежними музикантами, командами, студіями та медіаконцернами для організації роботи та збереження вмісту проєктів. Напрацювання та елементи кінцевого програмного забезпечення також можуть бути використані в інших системах для уніфікації інтерфейсів взаємодії різних сховищ, відновлюваної передачі даних, проєктування масштабованих систем та крос-платформних додатків. Також кінцеве програмне забезпечення і дані, зібрані при його використанні, можуть бути використані в дослідженнях в областях структурного та семантичного порівняння, виявлення розбіжностей та злиття.

Апробація результатів роботи. За матеріалами кваліфікаційної роботи опубліковано 2 наукові праці (1 стаття у фаховому виданні України та 1 теза):

1. Регенель Т. Ю., Фант М. О., Савіцький Р. С. Розробка системи контролю версій для музичних проєктів. Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки. 2025. Том 36 (75) № 3. С. 328-333.

2. Регенель Т. Ю., Кошелева М. В. Purpose, application and tradeoffs of Dual-DB approach. Тези доповідей науково-практичної онлайн-конференції здобувачів вищої освіти і молодих учених, присвяченої Дню науки (11–15 травня 2026 р., м. Житомир), 2026. С. 379-382.

		Регенель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ НАПРЯМІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ КОНТРОЛЮ ВЕРСІЙ МУЗИЧНИХ ПРОЄКТІВ

1.1 Постановка задачі

Аналіз предметної області показує, що за десятиліття розвитку технологій цифрової обробки звуку та роботи з музичними проєктами спеціалізоване програмне забезпечення й досі продовжує в корені видозмінювати робочі процеси інженерів, саунд-дизайнерів, музикантів, продюсерів тощо. Обслуговуючи не лише безпосередньо музичну індустрію, а й дотичні, такі як телебачення, кінематограф, бізнес та інші сфери, цей сегмент програмного забезпечення передбачає не єдину цифрову звукову робочу станцію для одного користувача, а обслуговування цілих команд і медіаконцернів з великими проєктами, робота над якими передбачає співпрацю учасників на різних етапах виробництва і робочих процесах.

Переважає більшість програмного забезпечення в цьому сегменті оперує файлами: проєкти DAW, семпли, мультитреки, файли сцен, налаштування тощо. Кожному учаснику на всіх етапах роботи важливо мати доступ до актуальних версій цих файлів, швидко і зручно надавати їх іншим учасникам, мати доступ до їх історії або організувати їх централізоване зберігання, наприклад, на FTP-сервері, що є стандартною практикою на великих студіях. За функціональними вимогами, найближчим видом програмного забезпечення для вирішення такої проблеми є системи контролю версій, зокрема, гібридні.

В інших сферах вже існують програмні продукти такого типу, що застосовуються багатьма фахівцями і вважаються стандартами своєї індустрії. При цьому кожен з цих продуктів адаптовано та оптимізовано під сферу, в якій він застосовується. Так, наприклад, в сфері розробки програмного забезпечення, найпоширеніша система контролю версій – Git, а

		Регенель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

також платформи для зберігання репозиторіїв в хмарі – GitHub, GitLab, BitBucket та інші. В сфері комп'ютерних ігор – PerForce та Diversion.

Проте, в музичній сфері спостерігається нестача рішень, що надавали б такий функціонал. До цього часу фахівці користувалися переважно лише альтернативними шляхами, на кшталт простого версіонування вбудованого у файлообмінники, онлайн-платформ для колаборації або ручного збереження та організації історії файлів проєктів.

Таким чином, завданням для кваліфікаційної роботи є розробка повноцінної вебсистеми контролю версій, функціонал якої розподілено між декількома ключовими компонентами (табл. 1.1).

Таблиця 1.1

Компоненти системи та їх призначення

Компонент	Призначення
Система контролю версій	Містить основний функціонал для версіонування файлів музичних проєктів: збереження версій та перемикання між ними, порівняння станів проєкту.
Клієнтський додаток	Для взаємодії користувача з основними операціями системи контролю версій на локальних проєктах, а також синхронізації їх актуального стану з вебплатформою та пристроями інших користувачів.
Вебплатформа	Для керування проєктами, доступом користувачів до них та хмарними сховищами, в тому числі для синхронізації між пристроями та роботи декількох користувачів над спільними проєктами.

Оскільки робота з музичними проєктами передбачає великі розміри файлів, для системи контролю версій критичною характеристикою є

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				9
Змн.	Арк.	№ докум.	Підпис	Дата		

ефективність використання дискового простору. Механізми зберігання версій вмісту мають бути розроблені з використанням оптимальних структур даних та алгоритмів порівняння та стискання [21].

При цьому для зберігання вмісту проєктів важлива підтримка користувацьких сховищ замість власного сховища, запропонованого платформою. Потреба у такій функції зумовлена декількома факторами.

Першим фактором є полегшення міграції. При переході на використання запропонованої вебсистеми користувач, імовірно, вже матиме придбане сховище в хмарних провайдерів на кшталт Google або DropBox, а в більших команд і компаній нормальною практикою є застосування власних FTP-серверів або хмарних сховищ для централізованого зберігання робочих матеріалів. Підтримка системою використання цих сховищ для збереження вмісту проєктів робить перехід на неї дешевшим та швидшим, а також може зменшити вартість інфраструктури зі сторони розробників вебсистеми [11].

Другим фактором є авторське право та безпека даних. Місцезнаходження сховища та доступ до нього пов'язані із законодавством, яке може варіюватися в залежності від країни. Особливо для великих компаній і студій, контроль над тим, де зберігається вміст, захищений авторським правом, комерційною таємницею, NDA тощо.

Крім, безпосередньо, вмісту, з версіями проєкту та його життєвим циклом тісно пов'язані елементи менеджменту та організації роботи. В більшості випадків, команди для цього використовують фізичні маркерні дошки, або цифрові інструменти для менеджменту. Втім, оскільки це напряду пов'язане з організацією історії версій проєкту, доцільно інтегрувати подібний функціонал в вебплатформу.

Результатом виконання поставленого завдання є програмний комплекс, за допомогою якого користувачі можуть керувати власними проєктами, історіями їх версій та збереженням їх вмісту локально або на сховищах.

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

1.2 Аналіз аналогів програмного продукту

Розглянемо існуючі програмні продукти, які можна вважати аналогом або частковою альтернативою розробці цієї кваліфікаційної роботи. На розгляд обрано виключно самостійні рішення, оскільки частковий функціонал історії версій, вбудований, наприклад, в ProTools або в хмарні сховища на кшталт DropBox є обмеженими у можливостях і напряду залежать від продукту, в який входять.

З прямих аналогів вдалося знайти один продукт – систему контролю версій DAWLab. Проєкт перебуває на стадії розробки, і наразі доступна лише інформація, надана на офіційній вебсторінці [9].

Згідно демонстрацій та інфографіки, для користувача проєкт фокусується на збереженні історії версій, ефективному використанні трафіку і простору на диску та хмарі, та спільній роботі декількох користувачів (рис. 1.1).



Рис. 1.1. Інфографіка ключових функцій та переваг DAWLab

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

Щодо наявності вебінтерфейсу інформації не надано, проте є фрагменти користувацького інтерфейсу віконного додатка (рис. 1.2). Йому характерний простий дизайн без ускладнених панелей керування та діалогових вікон, що робить продукт доступним для використання новим користувачам без попереднього досвіду.

Продемонстровано можливості контролю доступу до проєктів (публічний/приватний, надання доступу окремим користувачам), історію версій проєкту з підтримкою гілок та їх злиття, а також синхронізацію внесених змін з хмарою.

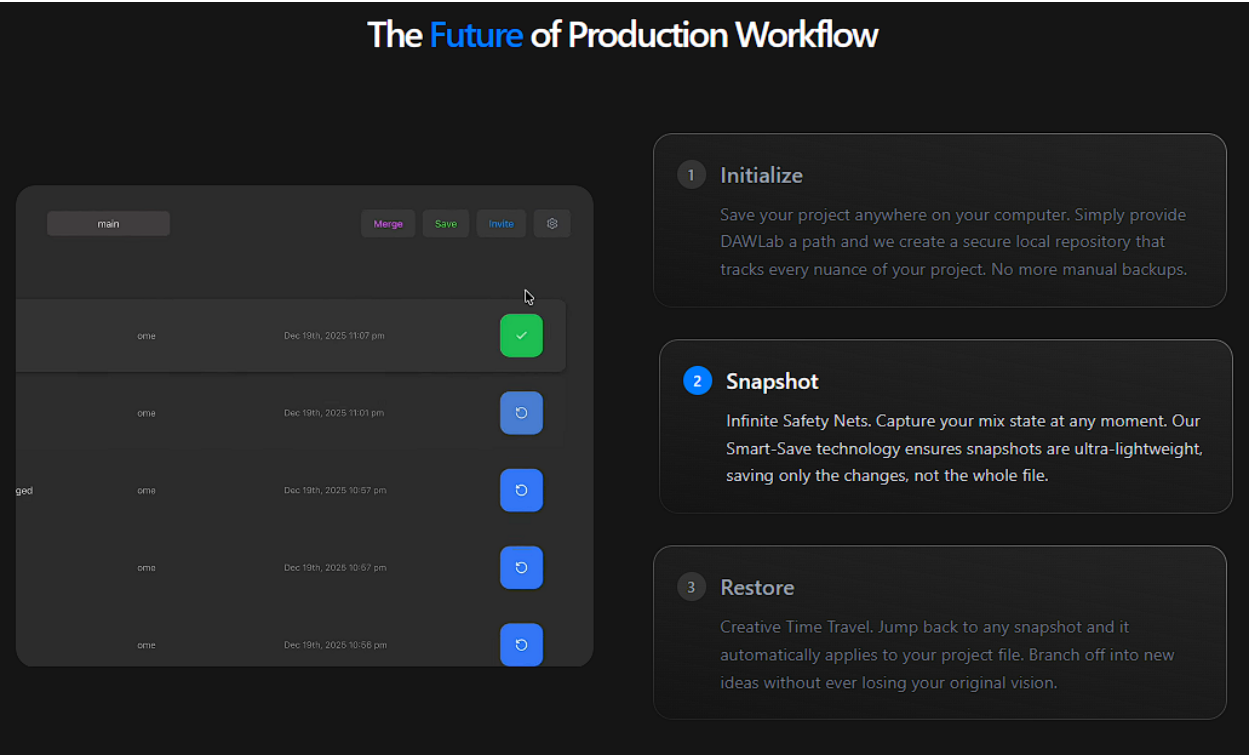


Рис. 1.2. Демонстрація інтерфейсу віконного додатка DAWLab

Водночас не зазначено, які операційні системи підтримуються віконним додатком, чи підтримує система власні користувацькі сховища та FTP-сервери для централізованого збереження вмісту проєктів, як саме відбувається процес злиття гілок та вирішення конфліктів між версіями, та не наведено конкретних метрик та результатів тестувань ефективності продукту.

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

Іншим корисним для аналізу аналогом є платформа GitHub, яку побудовано на основі системи контролю версій Git [7]. В своїй індустрії – інформаційних технологіях – ця платформа є стандартом світового рівня, та пропонує величезну кількість функціоналу та інтеграцій.

Основна взаємодія користувача з GitHub відбувається через вебінтерфейс (рис. 1.3). Через нього можна здійснювати керування проєктами, редагувати їх вміст, налаштовувати CI/CD автоматизації та інтеграції, переглядати різного роду аналітику проєктів, здійснювати менеджмент робочих процесів (Pull-request, Issues, GitHub Projects) тощо.

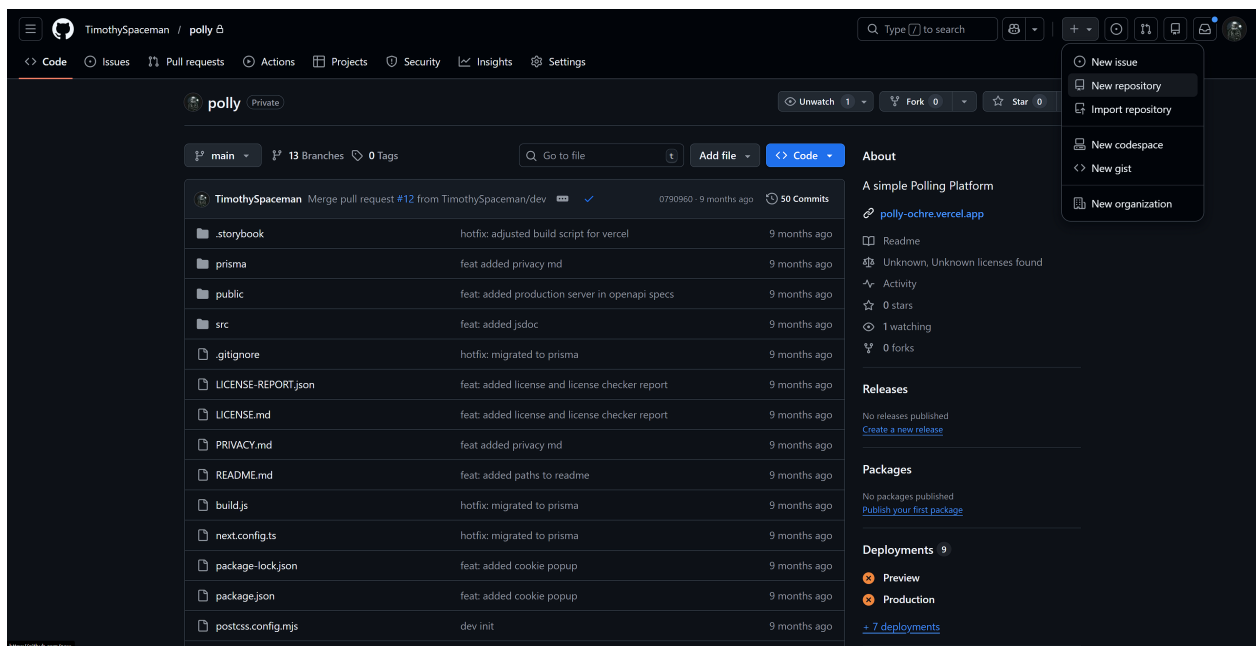


Рис. 1.3. Вебінтерфейс платформи GitHub

Для колаборації та організації проєктів платформа пропонує широкий набір можливостей. Зокрема, крім надання контрольованого доступу до репозиторію іншим користувачам, можна об'єднувати декілька репозиторіїв в організації, публікувати релізи, та навіть розгортати статичні сторінки за допомогою GitHub Pages.

Стосовно версіонування, оскільки GitHub використовує Git як систему контролю версій, підтримуються всі відповідні механізми і концепти: коміти, теги, гілки, злиття гілок, порядкові порівняння змін і вмісту файлів тощо.

		Регенель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				13
Змн.	Арк.	№ докум.	Підпис	Дата		

Платформа пропонує власний віконний додаток GitHub Desktop (рис. 1.4). Додаток дозволяє переглядати локальні та збережені на вебплатформі проєкти, редагувати вміст, керувати змінами в них та їх станом.

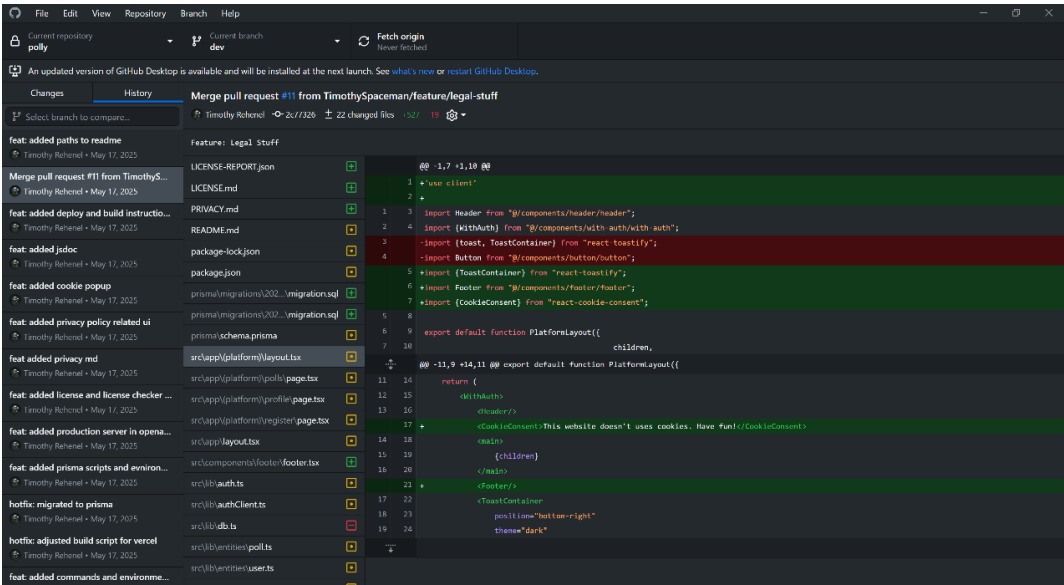


Рис. 1.4. Інтерфейс додатка GitHub Desktop

Для мобільних пристроїв з операційною системою iOS або Android, платформа пропонує власний додаток GitHub Mobile (рис. 1.5). Цей додаток не передбачає роботу з вмістом проєктів на локальному сховищі пристрою, проте дозволяє керувати проєктами та іншими сутностями на платформі.

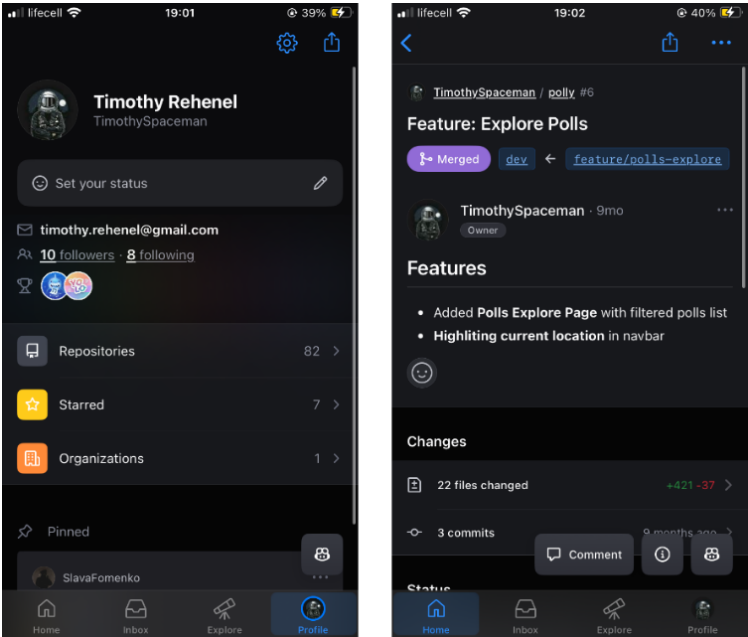


Рис. 1.5. Інтерфейс додатка GitHub Mobile

Третій аналог для аналізу – схожа платформа GitLab, розроблена українськими розробниками в 2011 році. Платформа схожа на GitHub за основним функціоналом, але крім готового хмарного рішення пропонує також можливість розгортання власних екземплярів веб-платформи. Наявний власний віконний додаток, а інтерфейс веб-платформи за характером та можливостями схожий до GitHub (рис. 1.6).

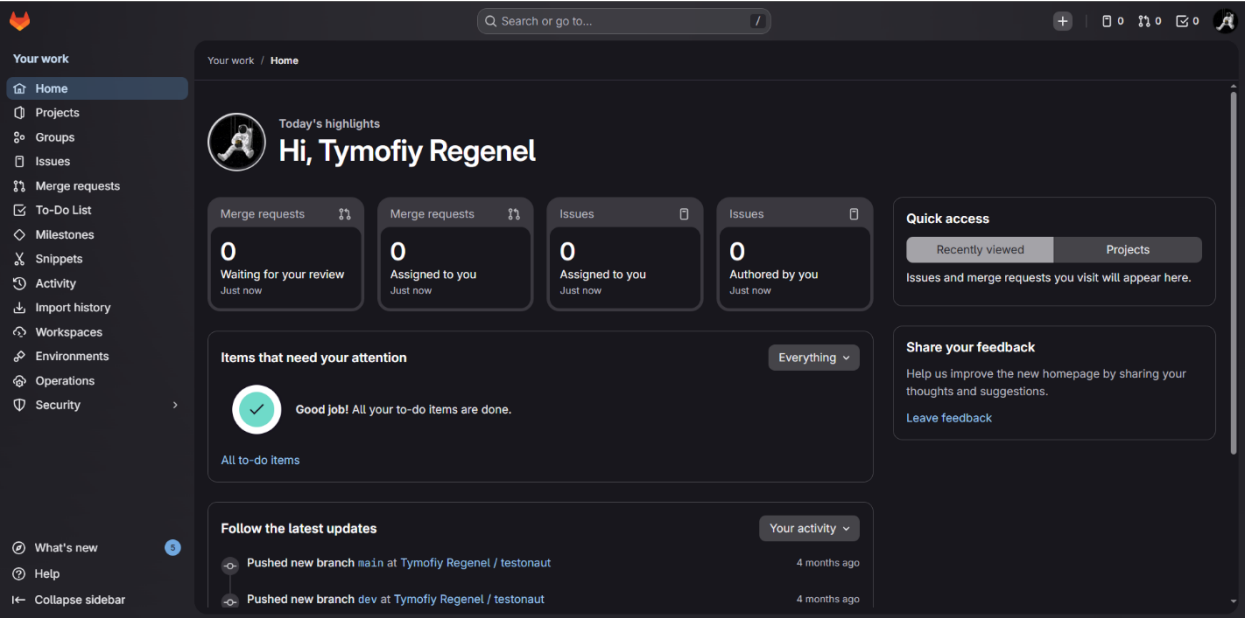


Рис. 1.6. Інтерфейс веб-платформи GitLab

При цьому як GitHub, так і GitLab використовують Git як систему контролю версій, яка в свою чергу призначена для версіювання змін у текстових файлах, і оптимізована відповідним чином. Музичним проєктам характерна робота з аудіо файлами великого розміру, які змінюються хаотично, а не по рядках як текстові файли. Система Git погано пристосована до роботи з такими даними та їх змінами, в зв’язку з чим і створюються альтернативи на кшталт Git LFS, Mercurial тощо [15]. Самі ж платформи в свою чергу мають ліміт на максимальний розмір окремого файлу, і цей ліміт менший, ніж потенційні розміри файлів музичних проєктів.

Для розглянутих платформ, відштовхуючись від поставленої задачі, підібрано перелік основних функцій та критеріїв для порівняння. Результати порівняння наведено в порівняльній таблиці 1.2.

		Регенель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

Порівняльна таблиця аналогів

Критерій	DAWLab	GitHub	GitLab
Контроль версій проєкту	+	+	+
Керування доступом інших користувачів	+	+	+
Віконні або інші клієнтські додатки	+	+	+
Можливість використання платформи з користувацьким сховищем	-	-	+-
Елементи менеджменту і організації роботи	-	+	+
Пристосованість системи контролю версій для роботи з музичними проєктами	+	-	-

Розглянуті аналоги або не пристосовані до роботи з великими бінарними файлами музичних проєктів, або, як у випадку DAWLab, перебувають на стадії розробки і не надають вичерпної технічної інформації для повноцінного порівняння.

Проаналізувавши та порівнявши існуючі аналоги, до розробки в межах цієї кваліфікаційної роботи було сформульовано такі вимоги. Система повинна мати зручний та інтуїтивно зрозумілий інтерфейс як віконного додатка, так і веб-платформи, а також підтримувати спільну роботу декількох користувачів над одним проєктом. Передбачається можливість під'єднання користувацьких сховищ та інтеграція елементів менеджменту й керування робочими процесами. Окрему увагу приділено пристосованості системи контролю версій до роботи з музичними проєктами, що передбачає ефективне використання дискового простору та мережевого трафіку.

1.3 Вибір архітектури для розроблюваної системи

Функціонал системи контролю версій потрібен як на серверній стороні (наприклад, для отримання переліку файлів на момент певної версії, або для отримання самої історії версій), так і на стороні віконного додатка безпосередньо для взаємодії з проєктами на диску користувача. Можливість перевикористання програмного коду, який реалізує цей функціонал, значно скоротить час, необхідний на розробку та підтримку обох компонентів системи.

Для цього було створено бібліотеку з таким програмним кодом, а серверний та віконний додатки використовуватимуть цю бібліотеку як залежність [8]. Нижче описано особливості її архітектури (рис. 1.7).

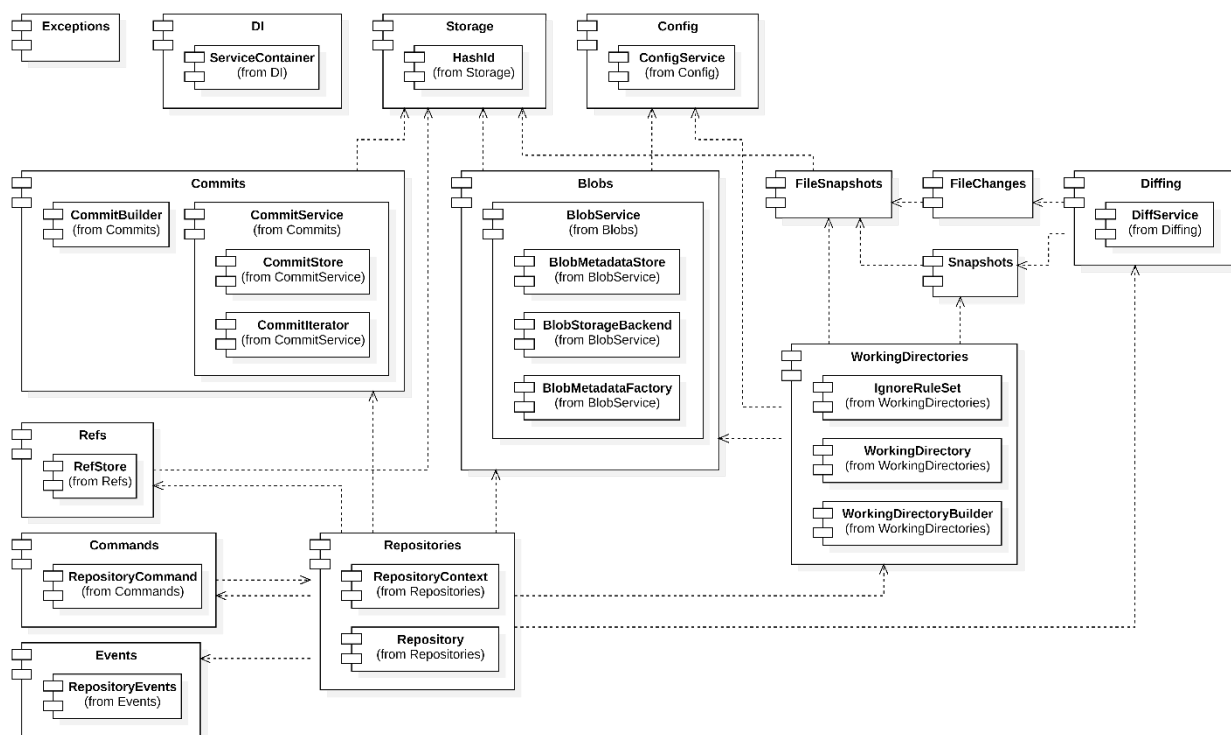


Рис. 1.7. Діаграма компонентів бібліотеки системи контролю версій

Реалізації деяких компонентів бібліотеки можуть відрізнятися для різних середовищ. Так, наприклад, зміна файлів у робочій директорії для віконного додатка реалізована безпосередньо змінами файлів на диску. Проте для серверного середовища такий підхід є неефективним, тож для серверного додатка робоча директорія може зберігати структуру файлів в оперативній

пам'яті. У такий самий спосіб будь-який додаток може впроваджувати власну конфігурацію, керувати потоками виконання операцій тощо.

Щоб кожен додаток міг використовувати власні реалізації таких компонентів без втручання в код бібліотеки, в бібліотеці впроваджено патерн «ін'єкція залежності»: всі компоненти спираються на інтерфейси, а конкретний додаток при ініціюванні надаватиме власні реалізації цих інтерфейсів [2]. До того ж, такий підхід є природним, оскільки платформи для цих додатків вже підтримують ін'єкцію залежності і пропонують її власні реалізації.

Щодо віконного додатка, архітектурним патерном для нього обрано MVVM, або Model-View-ViewModel (рис.1.8). Цьому патерну характерна відокремленість інтерфейсу (представлення), бізнес-логіки та даних (моделі), та проміжного шару, який визначає взаємозв'язки попередніх двох, виконує необхідні перетворення даних тощо (ViewModel) [13]. Також цей патерн підтримується «з коробки» такими платформами як .NET MAUI.

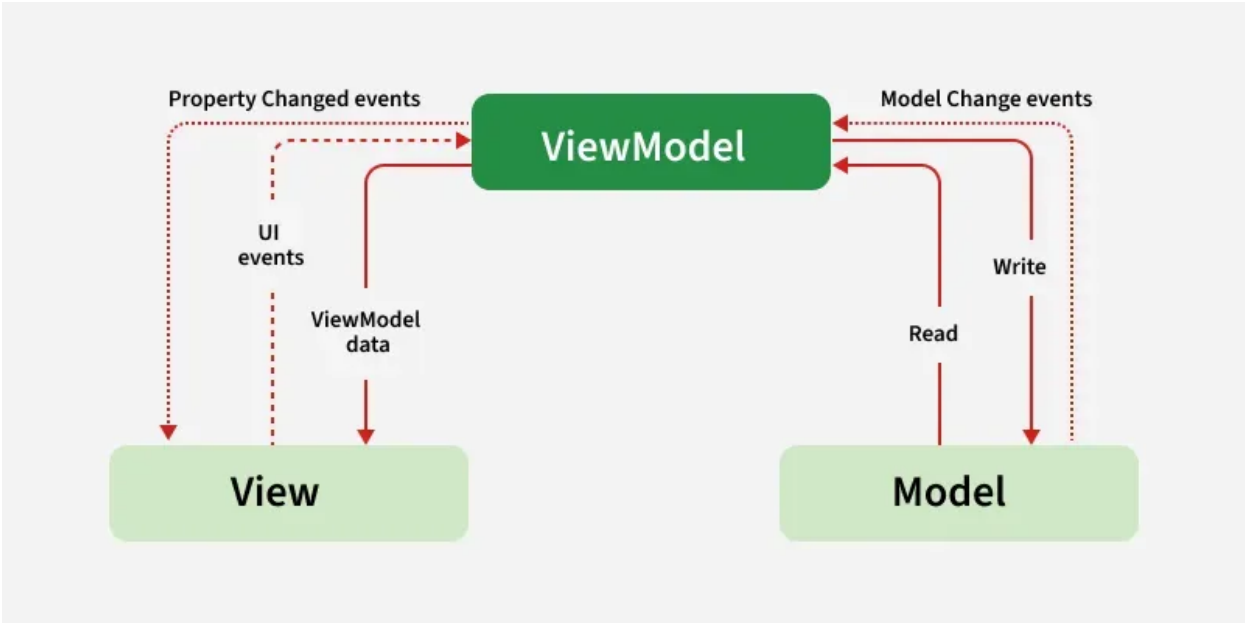


Рис. 1.8. Діаграма архітектурного патерну MVVM

Для віконних додатків важливим аспектом є можливість двостороннього зв'язування бізнес-логіки та даних з представленням – елементами інтерфейсу. В бібліотеці впроваджено ще два патерни, які

вирішують цю задачу. Патерн «Команда» дозволить прив'язувати виконання бізнес-логіки до елементів керування в інтерфейсі. Патерн «Спостерігач» працюватиме в зворотному напрямі: можна прив'язати елементи інтерфейсу для відображення даних до змін в цих самих даних.

При виборі архітектури для серверної частини важливо врахувати масштабованість системи і можливість ефективного використання та розширення інфраструктури при збільшенні навантаження і трафіку [19]. Мікросервісній архітектурі притаманна така модульність, проте вона вимагає додаткових ресурсів і часу для розробки, розгортання і підтримки.

Як альтернативу, яка пропонує баланс між складністю розробки і масштабованістю, обрано архітектуру «модульний моноліт» (рис. 1.9). Для розробки це дає чітку структуру проекту з відокремленням модулів один від одного [3]. З точки зору масштабування, весь проект все ще можна розгорнути як моноліт на ранніх стадіях, і, за потреби, виділяти додаткові ресурси або виокремлювати на додаткову інфраструктуру найбільш навантажені модулі, без кардинальних змін в коді додатка [28].

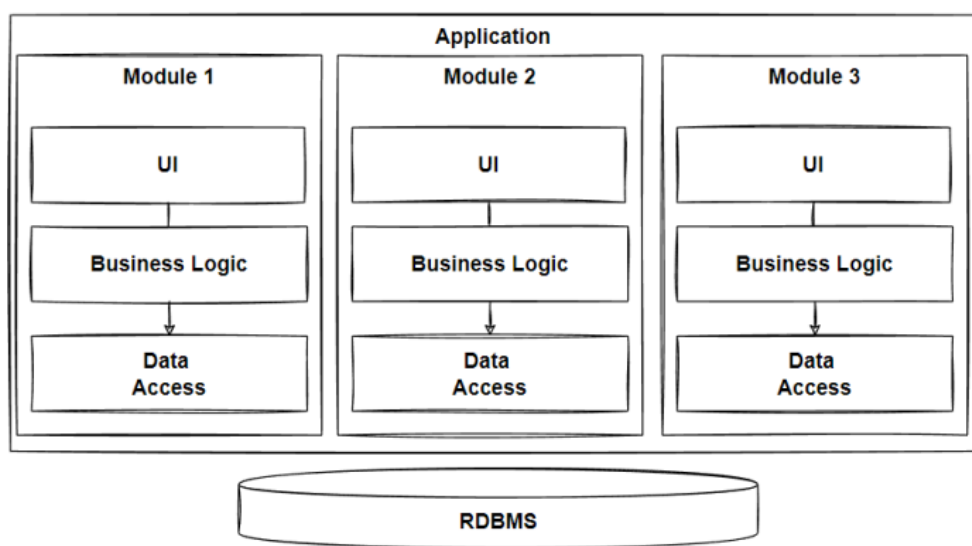


Рис. 1.9. Діаграма архітектурного патерну «модульний моноліт»

Для взаємодії з веб-платформою, було розроблено окремий веб-додаток, який надаватиме користувачеві графічний інтерфейс для взаємодії з функціями системи, не доступними через віконний додаток. Комунікація між

віконним та серверними додатками відбуватиметься за допомогою протоколу HTTPS і технологій REST API [26].

Нижче наведено діаграму розгортання для варіанту програмного комплексу, в якому основну частину серверного функціоналу і модуль, що відповідає за вміст проєктів, розгорнуто на окремій інфраструктурі (рис. 1.10).

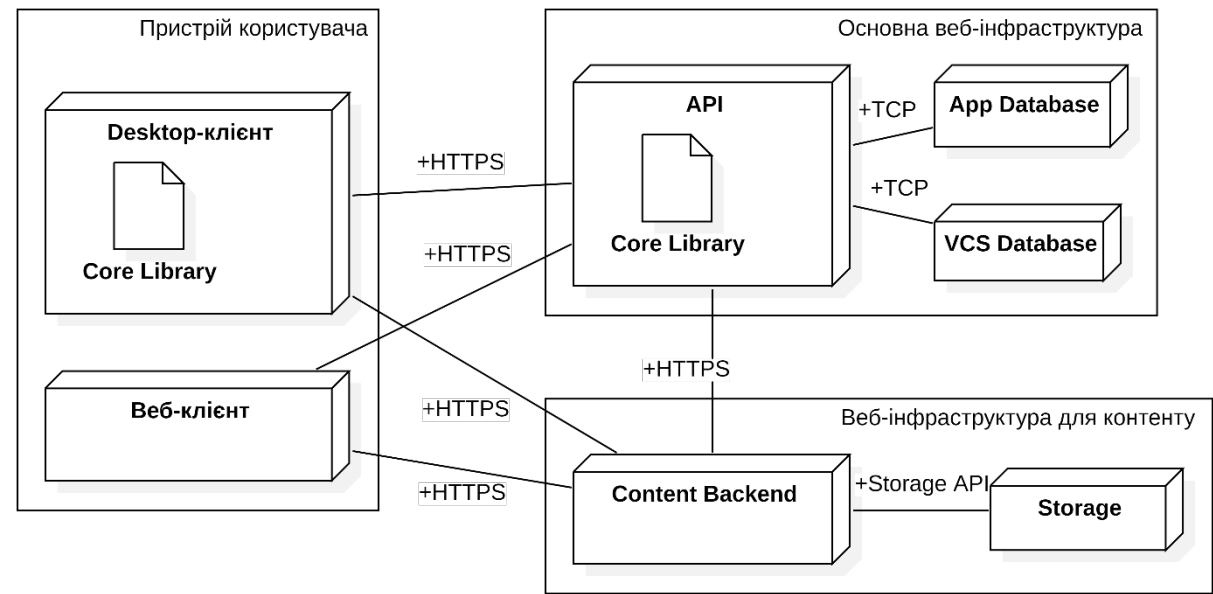


Рис. 1.10. Діаграма розгортання для варіанту програмного комплексу

Оскільки навіть проєкти, які зберігаються на користувацькому сховищі, можуть бути публічними, виникає задача з'єднати віконний додаток стороннього користувача з іншим сховищем. Здійснювати з'єднання віконного додатка зі сховищем напряму є небезпечним, оскільки для деяких протоколів (наприклад, SFTP) це передбачає передачу даних для автентифікації та авторизації. До того ж, виникає потреба у реалізації підтримки всіх необхідних протоколів на стороні віконного додатка.

Щоб вирішити цю проблему, однією з ролей серверного додатка є передача даних між сховищем і віконним додатком. Таким чином, дані для автентифікації не виходять за межі серверного додатка, а для віконного додатка передача даних здійснюється через єдиний інтерфейс незалежно від сховища, на якому зберігається проєкт.

1.4 Обґрунтування вибору засобів реалізації та вимоги до апаратного забезпечення

В системі передбачається обробка файлів великого розміру, застосування алгоритмів стиснення, порівняння тощо. Це підвищує вимоги до швидкодії та ефективності обраних технологій.

При цьому вимагається можливість написання спільної бібліотеки для серверного і віконного додатків. Таким чином, для обраної технології мають існувати інструментарії для написання обох видів додатків.

Враховуючи ці вимоги, для написання віконного і серверного додатків обрано технологію .NET. Для написання серверної частини використовуватиметься фреймворк ASP.NET Core, через високу ефективність, діючу підтримку розробників та спільноти, а також велику екосистему інструментарію [5]. Для написання віконного додатка використовуватиметься платформа .NET MAUI, через діючу підтримку розробників та можливість написання додатків для різних ОС [24]. На основі цього, можна сформулювати вимоги до пристрою користувача (табл. 1.3).

Таблиця 1.3

Технічні вимоги до пристрою користувача

Компонент	Мінімальні вимоги
ОС	Windows 10 (1809+) / MacOS 13+
ЦП	x64 або ARM64, 1 ГГц+
ОЗП	256 МБ
Дисковий простір	200 МБ (без урахування проєктів)

Для написання користувацького інтерфейсу веб-платформи обрано фреймворк NextJS. До його переваг можна віднести підтримку технологій Server-Side Rendering та Static Site Generation для покращення швидкодії додатка на стороні користувача і кращої SEO оптимізації [22]. Обрано TypeScript як засіб типізації для покращення якості вихідного коду і зменшення кількості помилок пов'язаних з типами даних, Tailwind CSS для

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				21
Змн.	Арк.	№ докум.	Підпис	Дата		

швидкої верстки завдяки великому набору попередньо генерованих класів та бібліотеку ShadCN для використання готових компонентів з гнучкою конфігурацією та відповідністю стандартам доступності, і в основі – React для написання інтерактивних інтерфейсів.

Для збереження даних про сутності хорошою опцією є реляційні СУБД. Дані про сутності в межах веб-платформи (користувачів, проєкти тощо) і дані про конкретний проєкт (коміти, гілки, метадані тощо) доцільно зберігати в різних базах даних, навіть в межах одного екземпляру СУБД. На роль СУБД для цього проєкту обрано PostgreSQL через надійність, швидкодію, хорошу підтримку ACID, розширюваність та активну підтримку зі сторони спільноти [20]. Для тимчасових даних (коди верифікації пристрою, інформація про завантаження тощо) використовуватиметься Redis [6].

Для комунікації серверного додатка з СУБД робота з SQL-запитами напряду не є ефективною та має низку проблем, зокрема в контексті типізації результатів запитів, небезпеки SQL ін'єкцій тощо. Обраною альтернативою є застосування ORM (Object-Relational Mapping), а саме Entity Framework Core через швидкодію та модульність.

Для розгортання серверного додатка та дотичних сервісів використовується засіб контейнеризації Docker з інструментами Docker Compose для оркестрації багатоконтейнерного середовища та Docker Swarm для масштабування системи. Це дозволяє забезпечити відтворюваність середовища розгортання, спростити конфігурацію залежностей між сервісами та ізолювати компоненти системи один від одного [29].

Як зворотній проксі-сервер використовується Nginx, який забезпечує маршрутизацію вхідних запитів, балансування навантаження, а також підтримку HTTPS [23]. Для керування міграціями баз даних використовується Flyway, який забезпечує контроль версій схеми та автоматизує її оновлення під час розгортання.

На відміну від даних про сутності, вміст проєкту не обов'язково зберігатиметься на власній інфраструктурі вебсистеми. Тому потрібно, щоб

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

система підтримувала найбільш розповсюджені конфігурації сховищ, що передбачає використання декількох технологій. Зокрема, стандартним сховищем для системи обрано AWS S3, який є одним із найнадійніших і найдоступніших сховищ у галузі, і також пропонує власні набори для розробки програмного забезпечення (SDK) для більшості існуючих мов програмування, що пришвидшує розробку [4].

Для підтримки FTP серверів застосовуватимуться протоколи FTP та SFTP. В перспективі підтримки довільного пристрою як сховища передбачається використання HTTPS протоколу для обміну даними між серверним додатком і спеціалізованим додатком на пристрої-сховищі. Між серверним і віконним додатками використовуватиметься протокол HTTPS з технологією HTTPS Streaming, протоколом TUS та використанням Range-запитів для ефективного обміну великим обсягом даних [30].

Отже, обраний набір технологій відповідає вимогам продукту: використовуються рішення з високою швидкістю та ефективним використанням ресурсів, порядок комунікації між вузлами обрано з огляду на безпеку даних, а всі технології мають діючу підтримку розробників і спільноти, що забезпечує перспективу подальшого розвитку системи. На основі цього сформовано вимоги до середовища для розгортання серверного додатка (табл. 1.4).

Таблиця 1.4

Технічні вимоги до середовища для серверного додатка

Компонент	Мінімальні вимоги
ЦП	2 ядра
ОЗП	2 ГБ
Дисковий простір	20 ГБ (без урахування сховища проєктів)
ОС	Linux (Ubuntu 22.04 / Debian 12)
Програмне забезпечення	Docker Engine 24+, Docker Compose v2, увімкнено Docker Swarm

Висновки до першого розділу

У першому розділі проведено аналіз предметної області та обґрунтовано доцільність розробки спеціалізованої системи контролю версій для музичних проєктів. Встановлено, що наявні рішення загального призначення – зокрема GitHub та GitLab – погано пристосовані до роботи з бінарними аудіофайлами великого розміру та не підтримують користувацькі сховища, тоді як єдиний прямий аналог у цій ніші – DAWLab – перебуває на стадії розробки і не надає вичерпної технічної інформації. За результатами порівняльного аналізу сформовано вимоги до системи, що розробляється: підтримка спільної роботи, під'єднання користувацьких сховищ, інтеграція елементів менеджменту та ефективне використання дискового простору і мережевого трафіку.

Визначено загальну архітектуру системи, в центрі якої спільна бібліотека з основним функціоналом контролю версій, що використовується як серверним, так і віконним додатком на основі патерну ін'єкції залежності. Для віконного додатка обрано архітектурний патерн MVVM, для серверного – модульний моноліт. Як технологічний стек обрано .NET та ASP.NET Core для серверної частини, .NET MAUI для віконного додатка, NextJS для вебплатформи, PostgreSQL та Redis для зберігання даних.

Для ефективного обміну великим обсягом даних між серверним і віконним додатками обрано протокол HTTPS у поєднанні з технологією HTTPS Streaming, протоколом TUS для завантаження та Range-запитами, що є критично важливим з огляду на великі розміри файлів музичних проєктів. Для розгортання серверної інфраструктури обрано Docker з оркестрацією через Docker Swarm, Nginx як зворотний проксі-сервер та Flyway для керування міграціями баз даних. Визначено мінімальні вимоги до апаратного забезпечення як середовища для серверного додатка, так і пристроїв кінцевих користувачів віконного додатка.

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

РОЗДІЛ 2. ПРОЄКТУВАННЯ КОМПОНЕНТІВ СИСТЕМИ

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

В першу чергу, сценаріями використання системи контролю версій в тій чи іншій формі є керування проєктами (репозиторіями) та версіонування їх вмісту. Зокрема, розроблювана система включатиме відстеження змін у файлах проєкту, вирішення конфліктів, збереження, перегляд та перемикавання версій, а також створення, налаштування та перегляд проєктів (рис. 2.1).

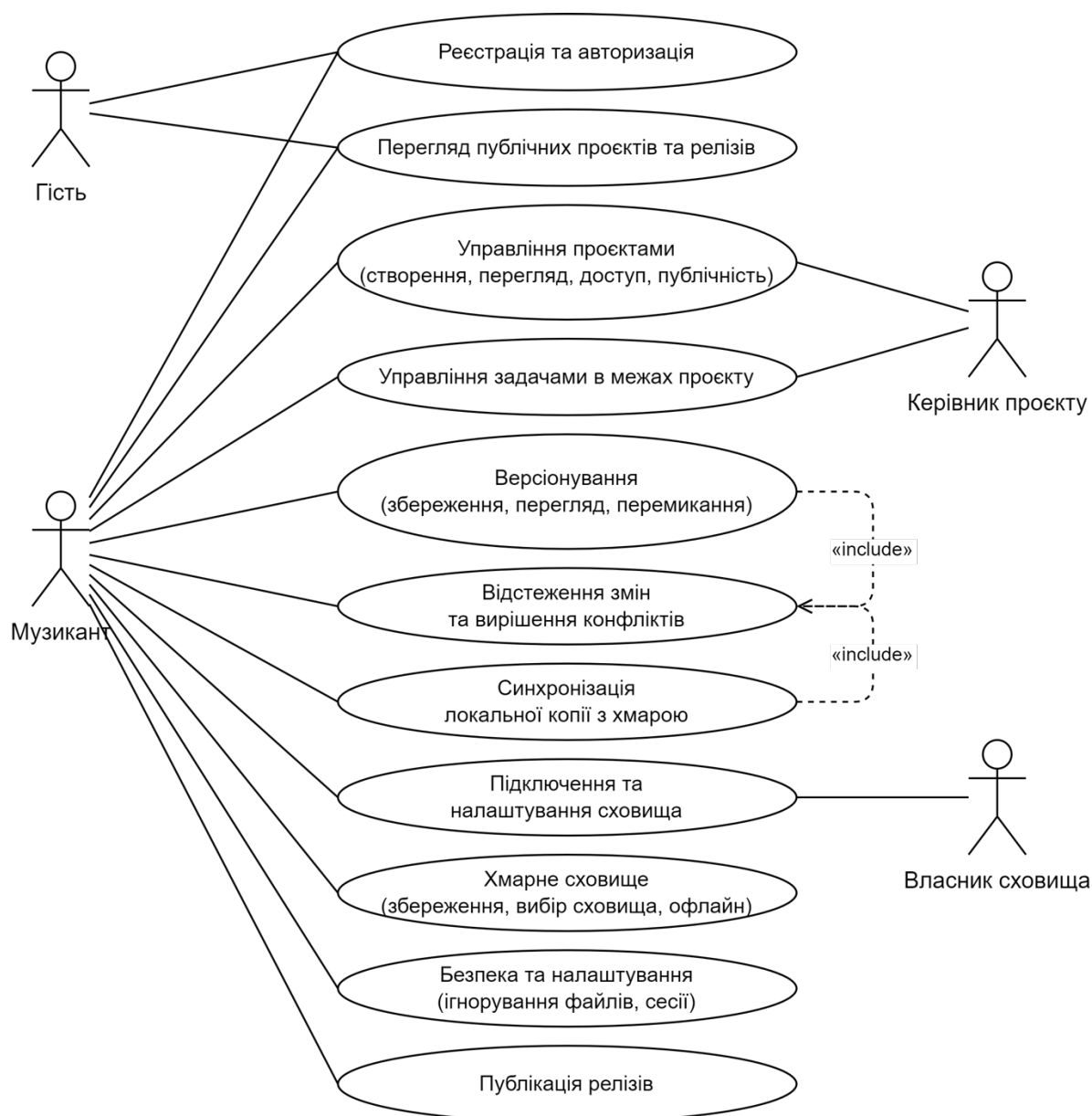


Рис. 2.1. Діаграма варіантів використання програмного комплексу

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				25
Змн.	Арк.	№ докум.	Підпис	Дата		

При цьому вирішення конфліктів ускладнене природою аудіо файлів: на відміну від, наприклад, файлів вихідного коду, де для вирішення конфліктів можна застосувати порядкове порівняння, вміст аудіо файлів це бінарні дані закодовані певним кодеком. Визначити, які саме зміни відбулися в аудіо вмісті файлу на основі цих даних практично неможливо.

Втім, часто це не є необхідним, оскільки семантична структура проєкту здебільшого представлена у файлах проєктів цифрових робочих станцій (DAW), таких форматів як “.als” для Ableton Live або “.rpp” для Reaper. Більшість з цих форматів являють собою структуру проєкту описану в XML та стиснену алгоритмами стискування на кшталт gzip. Це дозволяє зчитати вміст цих файлів і проводити семантичне та структурне порівняння та вирішення конфліктів. Хоча розроблювана система підтримує інтеграцію подібного функціоналу і це є планом майбутніх досліджень, в межах цієї кваліфікаційної роботи вирішення конфліктів реалізовано іншим чином.

Додаток, виявивши розбіжності, пропонує користувачеві дві опції: перезаписати локальну історію версій на хмарну, або зберегти обидва ланцюга версій в історії, з’єднавши їх автоматично генерованою «rollback»-версією, яка відтворює стан проєкту перед початком локальних змін. Такий підхід можна вважати допустимим враховуючи, що найчастіше конфлікт декількох версій у музичних проєктах вирішується вибором однієї з них як основної і відкиданням іншої. Водночас такий підхід дозволяє зберегти всі версії в історії, що робить можливим повернення до них у будь-який момент за потреби.

Альтернативним шляхом вирішення конфліктів та керування гілками є запити на злиття, які здійснюються на вебплатформі. Користувач обирає цільову гілку, гілку-джерело та робить запит на злиття, після чого серверний додаток генерує merge-версію, яка замінює стан цільової гілки на стан гілки-джерела, і додає цю версію на цільову гілку.

Оскільки розроблювана система контролю версій є гібридною, до сценаріїв використання також належать підключення та налаштування

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

сховища, управління проєктами на вебплатформі та керування їх публічністю і доступом користувачів до них, а також безпосередньо синхронізація локальної копії проєкту з хмарною. Додавати та налаштовувати сховища можуть користувачі, що мають відповідні дані (з точки зору системи, вони розглядаються як власники сховища). Втім, під'єднати конкретне сховище до свого проєкту може будь-який користувач, якому надано доступ до цього сховища.

Як елементи колаборації та організації роботи, в сценарії використання входить управління задачами в межах проєкту, попередньо описані запити на злиття, а також перегляд і публікація релізів.

Більшість сценаріїв використання пов'язаних з вебплатформою передбачають авторизацію, тож вона також належить до сценаріїв використання, як і реєстрація. При цьому перегляд публічних проєктів та релізів лишається доступним незареєстрованим користувачам.

Весь функціонал версіонування реалізовано в спільній бібліотеці, яка надаватиме основні сутності системи контролю версій та реалізацію всіх основних операцій над ними та взаємодії з проєктом. Розуміння цих сутностей і структури бібліотеки необхідне для подальшого проєктування додатків з її застосуванням.

Найменшою атомарною сутністю системи контролю версій є екземпляр вмісту будь-якого файлу в проєкті, або ж блоб. Фактично, ця сутність представлена у вигляді пари елементів: файлу з безпосереднім вмістом блобу і метаданих блобу. В ролі назви для файлу вмісту блобів використовується хеш цього вмісту, тим самим реалізовано механізм content-addressable сховища, а отже усунена дуплікація даних та збережено інші переваги цього механізму. Метадані блобу використовуються для уникнення постійної взаємодії з файлом вмісту для отримання таких його характеристик як, наприклад, розмір.

Наступна сутність – версія файлу (FileSnapshot) – представляє певну версію файлу проєкту і містить його шлях відносно кореневої директорії

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				27
Змн.	Арк.	№ докум.	Підпис	Дата		

проєкту, ідентифікатор блобу, що відповідає вмісту цієї версії файлу, а також дата внесення останніх змін в файл на момент збереження цієї версії. Саме ця сутність використовується для представлення переліку файлів у певній версії проєкту.

Проте, зберігати кожен версію проєкту як повний список версій файлів в ній є неефективно. Замість цього, використовуватиметься сутність зміни файлу (FileChange), яка містить дві версії файлу – «до» і «після», кожна з яких може бути або порожня, або конкретною версією файлу (екземпляром FileSnapshot). На основі порожності версій «до» і «після» можна одразу визначити тип зміни: новим файлам відповідає існування лише версії «після», видаленню файлів відповідає існування лише версії «до», а існування обох версій вказує на зміну попередньо існуючого файлу. Перейменування – окремий вид зміни файлу – можна також визначити, порівнявши ідентифікатори блобів і шляхи в обох версіях.

Також є потреба зберігати авторство версій проєктів. Це включає таку інформацію як ім'я користувача, його електронну пошту та ідентифікатор користувача на вебплатформі для інтерактивного відображення. При цьому система має підтримувати анонімних (незарєєстрованих) авторів, оскільки десктоп-додаток має функціонувати і в офлайн режимі. Тому для представлення авторів вводиться сутність UserIdentity, яка містить ім'я, та опціонально електронну адресу та ідентифікатор. Хоча ця сутність не обов'язково зберігатиметься як самостійна, вона є явно відокремленою з точки зору об'єктно-орієнтованої моделі системи.

Сама сутність версії проєкту (Commit) міститиме повідомлення (або ж назву версії), список змін у вигляді екземплярів FileChanges, інформацію про автора у вигляді екземпляра UserIdentity, дату та час збереження версії, ідентифікатор версії – хеш всіх вище перерахованих властивостей сутності, та опціонально ідентифікатор батьківської версії. За порівняння версій файлів, версій проєктів, ланцюгів комітів тощо відповідає сервіс порівняння (DiffService).

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

Іншою важливою сутністю для роботи системи контролю версій є посилання (Ref). За допомогою посилань система відслідковує, якій версії наразі відповідає стан поточної робочої директорії, певна гілка, останнє отримане від сервера оновлення тощо. Посилання зберігатимуться у вигляді послідовних записів (RefLogEntry), кожен з яких містить ключ (або назву) посилання, минуле значення, нове значення, час і повідомлення.

Наступним важливим елементом об'єктно-орієнтованої моделі основної бібліотеки є робоча директорія (WorkingDirectory). Фактично, це інтерфейс для взаємодії системи з файлами у директорії проєкту, що дозволяє абстрагувати фактичну реалізацію. Всі маніпуляції з файлами проєкту здійснюватимуться саме через робочу директорію, і за замовчуванням бібліотека надаватиме реалізацію робочої директорії, що взаємодіє безпосередньо із файловою системою користувача – LocalWorkingDirectory.

Практично для кожної з цих сутностей (і відповідного домену) бібліотека передбачає сховища (Stores), які абстрагують реалізацію збереження даних, сервіси, які реалізують операції над сутностями, та класи або структури для представлення самих даних. Для вмісту блобів є абстракція бекендів зберігання вмісту блобів (IBlobStorageBackend), що дозволяє змінювати реалізацію різних сховищ без втручання в бібліотеку. Всі елементи в бібліотеці базуються на інтерфейсах та патерні «Ін'єкція залежностей», що дає можливість в різних додатках використовувати різні реалізації певних компонентів. Для версій проєкту і робочих директорій використано патерн «будівельник», що у випадку версій проєкту розбиває створення версії на окремі кроки, а для робочої директорії дозволяє поєднати ін'єкцію залежностей з іншими параметрами (на кшталт шляху до кореневої директорії проєкту) при створенні екземпляру класу.

Оркестратором всіх сутностей і операцій над проєктом як з єдиним цілим є репозиторій. Для двосторонньої взаємодії з репозиторієм застосовано два патерни – «команда» і «спостерігач». Патерн «команда» дозволяє винести реалізацію окремих операцій з репозиторієм у відповідні класи та полегшує

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				29
Змн.	Арк.	№ докум.	Підпис	Дата		

використання бібліотеки в віконного додатка, для якого цей патерн є природним і добре підтримуваним. Патерн «спостерігач» полегшує інтеграцію бібліотеки з іншими елементами системи, зокрема графічним інтерфейсом віконного додатка.

Таким чином, бібліотеку поділено на компоненти, кожен з яких відповідає окремому домену або сутності, і які разом складають ієрархію залежностей (рис. 2.2). Базуючись на інтерфейсах, а не конкретних реалізаціях, компоненти підтримують різні реалізації без втручання в бібліотеку.

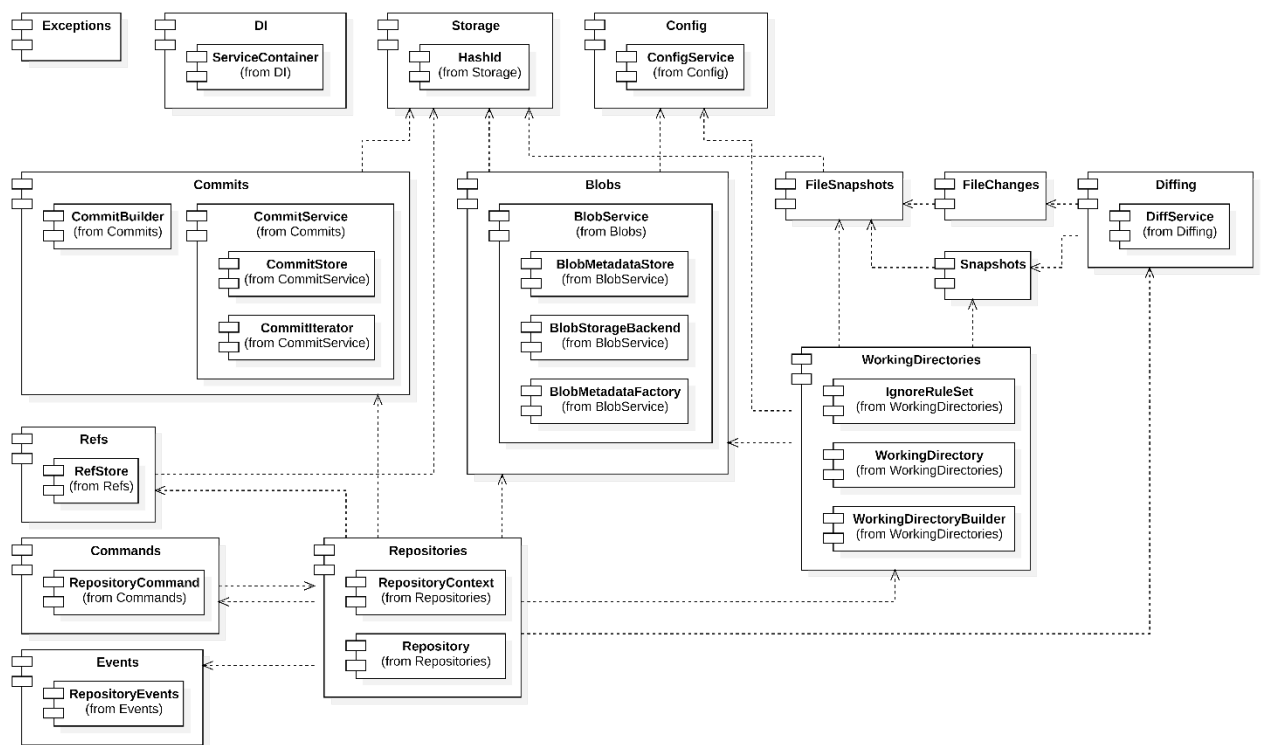


Рис. 2.2. Діаграма компонентів бібліотеки системи контролю версій

Основна частина бізнес-логіки в серверному і віконному додатках базуватиметься саме на цій об'єктно-орієнтованій моделі. При цьому для збереження розглянутих сутностей кожен з додатків використовуватиме власні класи-сутності і здійснюватиме конвертацію об'єктів між класами бібліотеки і класами-сутностями Entity Framework Core [27].

Водночас, віконний розширює цю модель власними класами. Зокрема, саме на стороні клієнта відбудуватиметься аналіз метаданих файлів проєкту, з

метою можливості введення End-to-End шифрування та зменшення навантаження на серверну частину. Аналізу підлягатимуть здебільшого файли проєктів DAW для отримання такої інформації як перелік використаних плагінів, тип наявного вмісту (аудіо, MIDI), бітрейт, темп та інша інформація про проєкт. Аналіз відбуватиметься при збереженні нової версії, а результати зберігатимуться в локальній базі даних проєкту у вигляді пари ключ-значення, де значення це масив рядків. Кожен збережений набір метаданих матиме прив'язку до відповідної версії, а також час створення та час надсилання на сервер для уникнення повторного надсилання. Ці дані згруповано у сутності SnapshotMetadataEntity.

Реалізацію аналізу метаданих покладено на класи-аналізатори, для яких створено спільний інтерфейс ISnapshotAnalyzer. В майбутньому по мірі розвитку проєкту додаватимуться нові класи-аналізатори для підтримки нових DAW та змін у форматі їх проєктів. Всі ці класи-аналізатори реєструються в головному контейнері ін'єкції залежності додатка, і як наслідок зібрані у вигляді колекції серед інших залежностей в клас-оркестратор SnapshotMetadataService. Завданням цього класу є запуск всіх аналізаторів при створенні нової версії за допомогою попередньо описаного механізму подій, злиття і збереження результатів в локальну базу даних проєкту, а також надсилання цих результатів серверній частині під час виконання синхронізації (push-операція).

До інших класів-сутностей віконного додатка можна віднести декілька класів ViewModel, зокрема, клас ActiveRepoViewModel для утримування стану відкритого на даний момент репозиторію та клас FileTransferViewModel для утримання стану і прогресу передачі файлу між віконним і серверним додатками.

Серверний додаток розширює об'єктно-орієнтовану модель бібліотеки ще більше, оскільки містить багато бізнес-логіки, яку реалізовано виключно на стороні серверу. Практично в кожному модулі серверного додатка

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

застосовано патерн «Репозиторій» для відокремлення бізнес-логіки від деталей зберігання даних.

Модуль користувачів містить класи User та UserAvatar для представлення інформації про користувача. Модуль автентифікації містить класи UserCredentials, Session, RefreshToken та DeviceInfo для представлення даних користувача для входу, активних сесій, прив'язаних до сесій refresh-токенів та інформацію про пристрій, якому відповідає певна сесія.

Модуль Storages містить класи StorageType для представлення підтримуваних видів сховищ та схеми їх конфігурації за допомогою класів-нащадків від SchemaField, клас Storage для представлення конкретного сховища та клас StorageAccess для представлення прав доступу користувача до сховища. Модуль Transfers в свою чергу не містить сутностей, а використовуючи сутності з модуля Storages реалізовує всю логіку передачі вмісту файлів на основі патерну «Адаптер» [14]. Зокрема, для всіх підтримуваних видів сховищ створюються класи-адаптери, що реалізують інтерфейс IStorageAdapter з методами для отримання вмісту файлу або створення TusStore для використання в автогенерованих ендпоінтах бібліотекою tusdotnet. Для створення екземплярів класів-адаптерів використано патерн «Фабрика» і створено клас StorageAdapterFactory, який містить в собі реалізацію створення усіх підтримуваних адаптерів.

Модуль Projects містить клас Project для представлення інформації про проєкт в межах веб-платформи та клас ProjectAccess для представлення прав доступу користувача до проєкту. Модуль Vcs крім всіх класів для взаємодії з основною бібліотекою СКВ містить клас MergeRequest для представлення запиту на злиття гілок та клас SnapshotMetadata для збереження метаданих проєкту для певної його версії. Модуль Tasks відповідає за задачі та містить класи ProjectTask для представлення задачі до певного проєкту, а також клас TaskAssignment для представлення призначення користувачів на виконання задачі.

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		32

2.2 Розробка бази даних системи

З точки зору серверного додатка, дані зберігатимуться в двох базах даних, одна з яких призначена для зберігання сутностей на рівні веб-платформи (користувачі, сесії, інформація про проекти, сховища, права доступу тощо), а друга – для зберігання сутностей СКВ (коміти, інформація про блоби, гілки тощо). Таке розділення дозволяє масштабувати кожен з баз даних незалежно у відповідь на навантаження запитом щодо конкретної групи даних, а також працювати з двома окремими спрощеними схемами замість однієї надмірно ускладненої [1].

При цьому фактично екземпляр СУБД, на роль якої було обрано PostgreSQL, може бути один. Проте серверний додаток взаємодіє з двома окремими контекстами Entity Framework Core, кожен з яких під'єднано до окремої бази даних. Всі таблиці бази даних рівня веб-платформи (AppDb) описано нижче в таблицях 2.1 – 2.14.

Таблиця 2.1

Таблиця «users»

Поле	Тип	Призначення
Id	uuid (PK)	Первинний ключ
Username	varchar(64)	Унікальний логін користувача
DisplayName	varchar(128)	Відображуване ім'я
Email	varchar(256)	Електронна адреса (унікальна, в нижньому регістрі)
IsEmailVerified	bool	Чи підтверджено email
CreatedAt	timestampz	Дата створення облікового запису
UpdatedAt	timestampz	Дата останнього оновлення
DeletedAt	timestampz	Дата «м'якого» видалення

Сутність користувача зберігається в таблиці «users» (таблиця 2.1) і зберігає дані про користувача платформи. До цих даних належить унікальний ідентифікатор та логін користувача, його електронна адреса та відображуване

		Регенель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

ім'я. За допомогою поля IsEmailVerified відслідковується, чи підтверджено електронну адресу. Часові мітки CreatedAt, UpdatedAt і DeletedAt використовуються для фіксації моменту створення, останньої зміни, а також «м'якого» видалення користувача.

Таблиця 2.2

Таблиця «user_avatars»

Поле	Тип	Призначення
Id	uuid (PK)	Первинний ключ
UserId	uuid (FK)	Власник аватара
StorageKey	varchar(512)	Ключ файлу у сховищі
Url	varchar(512)	Публічна URL-адреса зображення
SizeBytes	bigint	Розмір файлу в байтах
MimeType	varchar(128)	MIME-тип зображення
CreatedAt	timestampz	Дата завантаження аватара

Аватари користувачів зберігаються в таблиці «user_avatars» (таблиця 2.2). Кожен запис прив'язано до власника через поле UserId та описує одне завантажене зображення. Поле StorageKey містить ключ файлу у сховищі, а Url зберігає публічну адресу зображення. Додатково фіксується розмір файлу в байтах через SizeBytes, MIME-тип через MimeType та момент завантаження через CreatedAt.

Таблиця 2.3

Таблиця «user_credentials»

Поле	Тип	Призначення
Id	uuid (PK)	Первинний ключ
UserId	uuid (FK)	Власник даних для входу
PasswordHash	varchar(512)	Хеш паролю
CreatedAt	timestampz	Дата встановлення паролю
UpdatedAt	timestampz	Дата останньої зміни паролю

Дані для входу користувача за паролем зберігаються в таблиці «user_credentials» (таблиця 2.3). Запис прив'язано до користувача через поле UserId, а хеш паролю зберігається в полі PasswordHash. Часові мітки CreatedAt і UpdatedAt фіксують момент встановлення паролю та його останньої зміни.

Таблиця 2.4

Таблиця «sessions»

Поле	Тип	Призначення
Id	uuid (PK)	Первинний ключ
UserId	uuid (FK)	Власник сесії
IpAddress	varchar(64)	IP-адреса, з якої створено сесію
device_user_agent	varchar(512)	User-Agent клієнта
device_name	varchar(128)	Назва пристрою
device_os	varchar(128)	Операційна система пристрою
device_browser	varchar(128)	Браузер пристрою
CreatedAt	timestampz	Дата створення сесії
LastActiveAt	timestampz	Дата останньої активності
RevokedAt	timestampz	Дата відкликання сесії

Інформація про активні сесії користувачів зберігаються в таблиці «sessions» (таблиця 2.4). Кожен запис прив'язано до власника через UserId та представляє один сеанс автентифікації. Зберігаються IP-адреса, User-Agent клієнта, назва пристрою, операційна система та браузер. Часові мітки CreatedAt і LastActiveAt фіксують момент створення сесії та останньої активності, а поле RevokedAt зберігає момент відкликання сесії.

Таблиця 2.5

Таблиця «refresh_tokens»

Поле	Тип	Призначення
Id	uuid (PK)	Первинний ключ

SessionId	uuid (FK)	Прив'язана сесія
TokenHash	varchar(64)	Хеш токена
CreatedAt	timestampz	Дата видачі токена
ExpiresAt	timestampz	Дата закінчення терміну дії

Токени оновлення сесії зберігаються в таблиці «refresh_tokens» (таблиця 2.5). Кожен запис прив'язано до сесії через поле SessionId та містить хеш токена в полі TokenHash, щоб уникнути викриття токена. Часові мітки CreatedAt і ExpiresAt фіксують момент видачі токена та закінчення терміну його дії.

Таблиця 2.6

Таблиця «storage_types»

Поле	Тип	Призначення
Id	uuid (PK)	Первинний ключ
Key	varchar(64)	Унікальний програмний ключ типу
Label	varchar(64)	Відображувана назва типу
Description	varchar(256)	Опис типу сховища
ConfigSchema	jsonb	JSON-схема полів конфігурації

Типи сховищ зберігаються в таблиці «storage_types» (таблиця 2.6). Кожен запис представляє окремий підтримуваний тип сховища та містить унікальний програмний ключ у полі Key, відображувану назву в полі Label і текстовий опис у полі Description. Поле ConfigSchema зберігає JSON-схему полів конфігурації, яку має задовольняти сховище відповідного типу.

Таблиця 2.7

Таблиця «storages»

Поле	Тип	Призначення
Id	uuid (PK)	Первинний ключ

Name	varchar(128)	Назва сховища
StorageTypeId	uuid (FK)	Тип сховища
Config	jsonb	Конфігурація для підключення
IsPublic	bool	Чи є сховище публічним
IsDefault	bool	Чи є сховище наданим за замовчуванням
CreatedAt	timestampz	Дата створення
UpdatedAt	timestampz	Дата останнього оновлення

Сховища зберігаються в таблиці «storages» (таблиця 2.7). Кожен запис представляє окреме налаштоване сховище та містить його назву в полі Name і тип через посилання StorageTypeId. Параметри підключення зберігаються у форматі JSON у полі Config. Прапорці IsPublic та IsDefault визначають, чи є сховище публічним та чи надається воно за замовчуванням, а часові мітки CreatedAt і UpdatedAt фіксують момент створення та останнього оновлення.

Таблиця 2.8

Таблиця «storage_access»

Поле	Тип	Призначення
StorageId	uuid (PK, FK)	Сховище, до якого надано доступ
UserId	uuid (PK, FK)	Користувач, якому надано доступ
AccessType	int (enum)	Рівень доступу
CreatedAt	timestampz	Дата надання доступу
UpdatedAt	timestampz	Дата останнього оновлення

Доступ користувачів до сховищ зберігається в таблиці «storage_access» (таблиця 2.8). Кожен запис має ключ StorageId і UserId, а поле AccessType визначає рівень доступу. Часові мітки CreatedAt і UpdatedAt фіксують момент надання доступу та його останнього оновлення.

Таблиця «projects»

Поле	Тип	Призначення
Id	uuid (PK)	Первинний ключ
AuthorId	uuid (FK)	Автор проєкту
StorageId	uuid (FK)	Сховище для файлів проєкту
Title	varchar(128)	Назва проєкту
Description	varchar(1024)	Опис проєкту
IsPublic	Bool	Чи є проєкт публічним
IsInitialized	Bool	Чи завантажено принаймні одну версію
DefaultRefName	varchar(256)?	Назва основної гілки
CreatedAt	timestampz	Дата створення
UpdatedAt	timestampz	Дата останнього оновлення
DeletedAt	timestampz	Дата «м'якого» видалення

Проекти зберігаються в таблиці «projects» (таблиця 2.9). Запис містить автора проєкту та сховище для файлів через StorageId і AuthorId, а також назву та опис. Поле IsPublic визначає, чи є проєкт публічним, а IsInitialized показує, чи завантажено принаймні одну версію. Поле DefaultRefName зберігає назву основної гілки. Часові мітки CreatedAt, UpdatedAt і DeletedAt фіксують момент створення, останньої зміни та «м'якого» видалення.

Таблиця 2.10

Таблиця «project_access»

Поле	Тип	Призначення
ProjectId	uuid (PK, FK)	Проект, до якого надано доступ
UserId	uuid (PK, FK)	Користувач, якому надано доступ
AccessType	int (enum)	Рівень доступу
CreatedAt	timestampz	Дата надання доступу
UpdatedAt	timestampz	Дата останнього оновлення

Доступ користувачів до проєктів зберігається в таблиці «project_access» (таблиця 2.10). Кожен запис має складений ключ ProjectId і UserId, а поле AccessType визначає рівень наданого доступу. Часові мітки CreatedAt і UpdatedAt фіксують момент надання доступу та його останнього оновлення.

Таблиця 2.11

Таблиця «project_tasks»

Поле	Тип	Призначення
Id	uuid (PK)	Первинний ключ
ProjectId	uuid (FK)	Проект, до якого належить задача
AuthorId	uuid? (FK)	Автор задачі
Title	varchar(256)	Назва задачі
Description	varchar(4096)?	Опис задачі
Status	int (enum)	Статус
CommitId	varchar(32)?	Прив'язаний коміт
Deadline	timestampz?	Дедлайн (необов'язковий)
CreatedAt	timestampz	Дата створення
UpdatedAt	timestampz	Дата останнього оновлення

Задачі проєкту зберігаються в таблиці «project_tasks» (таблиця 2.11). Кожен запис прив'язано до проєкту через ProjectId та до автора через AuthorId, а також містить назву й опис задачі. Поле Status зберігає поточний статус задачі, поле CommitId дозволяє прив'язати задачу до конкретного коміту, а поле Deadline задає термін виконання. Часові мітки CreatedAt і UpdatedAt фіксують момент створення та останньої зміни задачі.

Таблиця 2.12

Таблиця «task_assignments»

Поле	Тип	Призначення
TaskId	uuid (PK, FK)	Задача, на яку призначено користувача

продовж. табл. 2.12

UserId	uuid (PK, FK)	Призначений користувач
CreatedAt	timestampz	Дата створення

Призначення користувачів на виконання задач зберігаються в таблиці «task_assignments» (таблиця 2.12). Кожен запис зв'язує задачу та користувача через складений ключ TaskId і UserId. Часова мітка CreatedAt фіксує момент призначення виконавця.

Таблиця 2.13

Таблиця «releases»

Поле	Тип	Призначення
Id	uuid (PK)	Первинний ключ
ProjectId	uuid (FK)	Проект, до якого належить реліз
AuthorId	uuid? (FK)	Автор релізу
Title	varchar(256)	Назва релізу
CreatedAt	timestampz	Дата створення

Релізи зберігаються в таблиці «releases» (таблиця 2.13). Кожен запис має ідентифікатор в полі Id, проєкт та автора через поля ProjectId та AuthorId, а також назву в полі Title та часову мітку створення в полі CreatedAt.

Таблиця 2.14

Таблиця «release_files»

Поле	Тип	Призначення
Id	uuid (PK)	Первинний ключ
ReleaseId	uuid (FK)	Реліз, до якого належить файл
FileName	varchar(512)	Назва файлу
BlobId	varchar(32)	Ідентифікатор відповідного блобу
CreatedAt	timestampz	Дата створення

Інформація про файли, які належать до певного релізу зберігаються в таблиці «release_files» (таблиця 2.14). Кожен запис прив'язано до реліза за допомогою ReleaseId. Поле FileName зберігає назву файлу, BlobId зберігає ідентифікатор блобу з відповідним вмістом. Часова мітка CreatedAt містить час створення запису.

Сутності системи контролю версій, а також сутність з результатами аналізу метаданих та запити на злиття гілок зберігаються в окремій базі даних рівня СКВ (VcsDb). Таблиці цієї бази даних описано в таблицях 2.15 – 2.19.

Таблиця 2.15

Таблиця «blob_metadata»

Поле	Тип	Призначення
Id	varchar(32) (PK)	Хеш-ідентифікатор блобу
ProjectId	uuid (PK)	Проект, якому належить блоб
Length	bigint	Розмір блобу в байтах

Метадані блобів системи контролю версій зберігаються в таблиці «blob_metadata» (таблиця 2.15). Кожен запис ідентифікується складеним ключем з хеш-ідентифікатора блобу в полі Id та проекту в полі ProjectId. Поле Length зберігає розмір блобу в байтах.

Таблиця 2.16

Таблиця «commits»

Поле	Тип	Призначення
Id	varchar(32) (PK)	Хеш-ідентифікатор коміту
ProjectId	uuid (PK)	Проект, якому належить коміт
ParentId	varchar(32)?	Хеш батьківського коміту
SecondParentId	varchar(32)?	Хеш другого батьківського коміту
Kind	int (enum)	Тип коміту (Default, Revert, Merge)
Message	varchar(4096)	Повідомлення коміту
CreatedAt	timestampz	Дата створення коміту

AuthorId	uuid?	UUID автора
AuthorName	varchar(256)?	Відображуване ім'я автора
AuthorEmail	varchar(512)?	Email автора
Changes	jsonb	Масив змін файлів у коміті

Коміти зберігаються в таблиці «commits» (таблиця 2.16). Кожен запис ідентифікується складеним ключем з хеш-ідентифікатора коміту та проєкту, а поля ParentId і SecondParentId зберігають хеші батьківських комітів. Поле Kind визначає тип коміту, поле Message містить його повідомлення. Дані автора зберігаються в полях AuthorId, AuthorName і AuthorEmail. Поле Changes зберігає масив змін файлів у форматі JSON. Часова мітка CreatedAt зберігає момент створення коміту.

Таблиця 2.17

Таблиця «refs»

Поле	Тип	Призначення
ProjectId	uuid (PK)	Проект, якому належить гілка
Name	varchar(256) (PK)	Назва гілки
CommitId	varchar(32)?	Хеш коміту, на який вказує гілка

Гілки та інші вказівники системи контролю версій зберігаються в таблиці «refs» (таблиця 2.17). Кожен запис ідентифікується складеним ключем з проєкту в полі ProjectId та назви гілки в полі Name. Поле CommitId зберігає хеш коміту, на який вказує гілка.

Таблиця 2.18

Таблиця «merge_requests»

Поле	Тип	Призначення
Id	uuid (PK, FK)	Первинний ключ
AuthorId	uuid	Автор запиту на злиття

ProjectId	uuid	Проект, до якого відноситься запит
Title	varchar(4096)	Заголовок запиту на злиття
TargetRefName	varchar(256)	Назва цільової гілки
SourceRefName	varchar(256)	Назва гілки-джерела
MergeCommitId	varchar(32)	Хеш merge-коміту
CreatedAt	timestampz	Дата створення запиту

Запити на злиття гілок зберігаються в таблиці «merge_requests» (таблиця 2.18). Кожен запис містить автора через AuthorId та проект через ProjectId. Поле Title містить заголовок, поля TargetRefName і SourceRefName зберігають назви цільової гілки та гілки-джерела, а поле MergeCommitId містить хеш merge-коміту. Поле CreatedAt містить момент створення запиту.

Таблиця 2.19

Таблиця «snapshot_metadata»

Поле	Тип	Призначення
CommitId	varchar(32) (PK)	Хеш коміту, до якого прив'язані метадані
ProjectId	uuid (PK)	Проект, якому належать метадані
Data	jsonb	Метадані у вигляді пар ключ-значення
SubmittedAt	timestampz	Дата подання метаданих клієнтом

Метадані до вмісту проекту зберігаються в таблиці «snapshot_metadata» (таблиця 2.19). Кожен запис ідентифікується складеним ключем з хеш-ідентифікатора коміту в полі CommitId та проекту в полі ProjectId. Поле Data зберігає метадані у вигляді пар ключ-значення з індексом GIN, а поле SubmittedAt фіксує момент подання метаданих клієнтом.

Водночас віконний додаток також використовує реляційну базу даних для збереження даних конкретного проекту. Альтернативою є збереження даних у файлах з розробленим власним форматом, проте в межах цієї кваліфікаційної роботи обрано реляційну базу даних (зокрема, SQLite) як

рішення, що надає фіксовану схему та повну підтримку Entity Framework Core при допустимо нижчій швидкодії порівняно з власним форматом.

Окрім сутностей системи контролю версій, в цій базі даних зберігається також частина конфігурації проєкту та результати аналізу метаданих. Опис всіх таблиць наведено в таблицях 2.20 – 2.26.

Таблиця 2.20

Таблиця «BlobMetadata»

Поле	Тип	Призначення
Id	BLOB(16) (PK)	Хеш-ідентифікатор блобу
Length	INTEGER	Розмір блобу в байтах

Метадані блобів локальної бази даних зберігаються в таблиці «BlobMetadata» (таблиця 2.20). Кожен запис ідентифікується хеш-ідентифікатором блобу в полі Id. Поле Length зберігає розмір блобу в байтах.

Таблиця 2.21

Таблиця «Commits»

Поле	Тип	Призначення
Id	BLOB(16) (PK)	Хеш-ідентифікатор коміту
ParentId	BLOB(16)?	Хеш батьківського коміту
SecondParentId	BLOB(16)?	Хеш другого батьківського коміту
Kind	INTEGER (enum)	Тип коміту (Default, Revert, Merge)
Message	TEXT(4096)	Повідомлення коміту
CreatedAt	INTEGER	Дата створення (Unix timestamp)
Author_UserId	TEXT (uuid)?	ID зареєстрованого користувача
Author_Name	varchar(256)?	Відображуване ім'я автора
Author_Email	varchar(512)?	Email автора

Коміти локальної бази даних зберігаються в таблиці «Commits» (таблиця 2.21). Кожен запис ідентифікується хеш-ідентифікатором коміту в полі Id. Поля ParentId і SecondParentId зберігають хеші батьківських комітів

за головною і побічною гілкою відповідно. Поле Kind визначає тип коміту, поле Message містить його повідомлення. Часова мітка CreatedAt фіксує момент створення у форматі Unix timestamp. Відомості про автора коміту зберігаються в полях Author_UserId, Author_Name і Author_Email.

Таблиця 2.22

Таблиця «FileSnapshots»

Поле	Тип	Призначення
Id	TEXT (uuid, PK)	Унікальний ідентифікатор версії файлу
BlobId	BLOB(16) (FK)	Хеш вмісту файлу
FilePath	TEXT(2048)	Шлях до файлу у робочій директорії
LastModified	INTEGER	Дата останньої зміни файлу (Unix timestamp)

Версії файлів зберігаються в таблиці «FileSnapshots» (таблиця 2.22). Кожен запис має унікальний ідентифікатор у полі Id та посиляється на блок з вмістом цієї версії файлу через BlobId. Поле FilePath зберігає відносний шлях до файлу у робочій директорії. Часова мітка LastModified фіксує момент останньої зміни файлу у форматі Unix timestamp, який зберігається з метою кешування стану проєкту.

Таблиця 2.23

Таблиця «FileChanges»

Поле	Тип	Призначення
Id	TEXT (uuid, PK)	Унікальний ідентифікатор запису про зміну
CommitId	BLOB(16) (FK)	Коміт, до якого належить ця зміна
BeforeSnapshotId	TEXT? (FK)	Версія файлу до зміни
AfterSnapshotId	TEXT? (FK)	Версія файлу після зміни

Зміни файлів зберігаються в таблиці «FileChanges» (таблиця 2.23). Кожен запис має унікальний ідентифікатор у полі Id та прив'язується до

коміту через CommitId. Поля BeforeSnapshotId і AfterSnapshotId посилаються на версії файлу до та після зміни, дозволяючи відстежувати характер зміни.

Таблиця 2.24

Таблиця «RefLogEntries»

Поле	Тип	Призначення
Id	INTEGER (PK)	Унікальний ідентифікатор запису
RefKey	TEXT(256)	Назва гілки або вказівника
OldHash	BLOB(16)?	Попереднє значення вказівника
NewHash	BLOB(16)?	Нове значення вказівника
Message	TEXT(1024)	Опис операції
Timestamp	INTEGER	Час здійснення операції

Історія змін вказівників зберігається в таблиці «RefLogEntries» (таблиця 2.24). Кожен запис має унікальний ідентифікатор у полі Id та містить назву гілки або вказівника в полі RefKey. Поля OldHash і NewHash зберігають попереднє та нове значення вказівника, поле Message містить опис операції. Timestamp фіксує момент операції у форматі Unix timestamp.

Таблиця 2.25

Таблиця «SnapshotMetadata»

Поле	Тип	Призначення
CommitId	BLOB(16) (PK)	Хеш коміту, до якого прив'язані метадані
Data	TEXT	Метадані у форматі JSON
CreatedAt	INTEGER	Час створення (Unix timestamp)
SyncedAt	INTEGER?	Час синхронізації з сервером

Метадані вмісту проєкту локально зберігаються в таблиці «SnapshotMetadata» (таблиця 2.25). Кожен запис має хеш-ідентифікатор коміту в полі CommitId. Поле Data зберігає метадані у форматі JSON, поле CreatedAt та SyncedAt фіксують час створення та здійснення аналізу.

Таблиця «Config»

Поле	Тип	Призначення
Key	TEXT(256) (PK)	Ключ налаштування
Value	TEXT?	Значення налаштування

Налаштування проекту зберігаються в таблиці «Config» (таблиця 2.26). Кожен запис ідентифікується ключем налаштування в полі Key. Поле Value зберігає значення відповідного налаштування.

Для керування схемами баз даних та міграціями використано засоби Entity Framework Core. Міграція локальної бази даних проекту здійснюється віконним додатком автоматично. Для серверного додатка є потреба в автоматизації та передбачуваності процесу міграції та розгортання БД. Тому засоби Entity Framework Core, використовуються спочатку для генерації C# міграції, а потім – ідемпотентних SQL-скриптів на їх основі. Для керування схемою баз даних на основі цих скриптів застосовано інструмент Flyway [12].

2.3 Проктування та реалізація алгоритмів роботи системи

Один із найважливіших алгоритмів роботи серверного додатка – передача даних між клієнтом і сховищем проекту. Завантаження вмісту на клієнт і на сховище використовують різні протоколи, проте мають деякі спільні компоненти.

Завантаження вмісту блобів на сховище реалізовано на основі протоколу TUS – відкритого стандарту для відновлюваного завантаження файлів через HTTP. Протокол розбиває передачу даних на окремі частини, що дозволяє перервати і продовжити завантаження без втрати вже переданих даних, що є критично важливим з огляду на великі розміри файлів музичних проєктів. Діаграму послідовності процесу завантаження наведено на рис. 2.3.

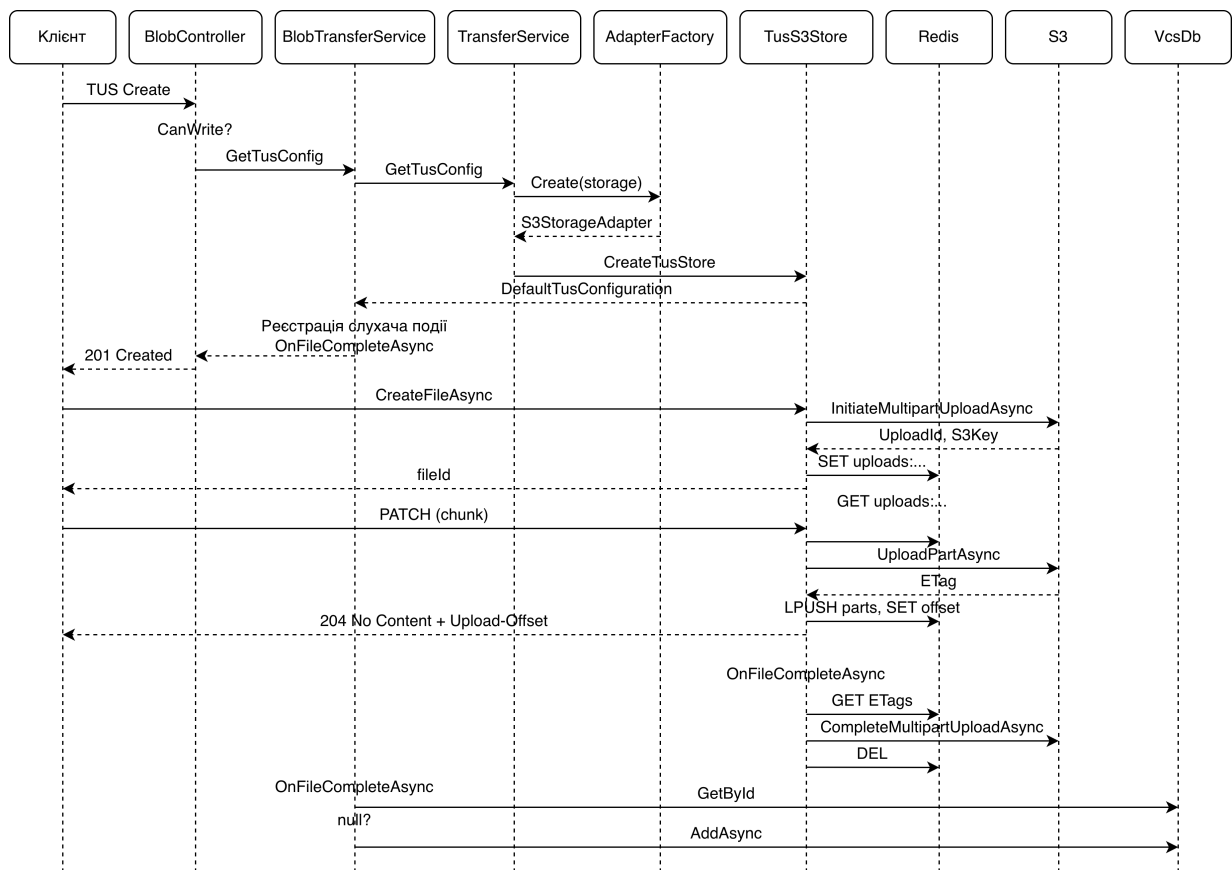


Рис. 2.3. Діаграма послідовності процесу завантаження файлів на сховище

Процес завантаження починається з того, що віконний клієнт надсилає запит на ендпоінт `api/projects/{projectId}/blobs`, який обробляється класом `BlobController`. Контролер перевіряє, чи має поточний автентифікований користувач право на запис до вказаного проєкту, після чого делегує формування TUS-конфігурації сервісу `BlobTransferService`. Той, в свою чергу, звертається до `TransferService`, який завантажує з бази даних відповідне сховище та через `StorageAdapterFactory` створює конкретний адаптер для роботи з ним. Фабрика визначає тип сховища за ключем (наприклад, `aws-s3`) та десеріалізує збережену конфігурацію підключення у відповідний клас (наприклад, `S3StorageConfig`). Таким чином, логіка взаємодії з конкретним видом сховища повністю інкапсульована в адаптері, а решта системи працює через уніфікований інтерфейс `IStorageAdapter`.

Для сховищ типу S3 адаптер повертає екземпляр `TusS3Store` – власної реалізації TUS-сховища, що відповідає інтерфейсам бібліотеки `tusdotnet`. Під

час самого завантаження TusS3Store використовує механізм Multipart Upload API від Amazon S3: на початку завантаження ініціюється завантаження частинами через `InitiateMultipartUploadAsync`, яке повертає унікальний `UploadId`. Метадані поточного завантаження (ключ об'єкта в S3, ідентифікатор завантаження, зміщення, оригінальні TUS-метадані) зберігаються в Redis із обмеженим часом життя (30 хвилин) під ключем `uploads:{userId}:{fileId}`. Це забезпечує відстеження стану між окремими HTTP-запитами без збереження стану в пам'яті сервера, а також дозволяє обробляти завантаження будь-яким доступним в даний момент екземпляром серверного додатка за умови, що всі вони працюють з єдиним екземпляром Redis.

Кожен наступний фрагмент даних від клієнта обробляється методом `AppendDataAsync`: дані зчитуються в буфер у пам'яті та надсилаються на сховище S3 як окрема частина через `UploadPartAsync`. Після кожної успішно завантаженої частини її номер та ETag зберігаються в Redis у вигляді списку під ключем `uploads:{userId}:{fileId}:parts`, а зміщення в записі завантаження оновлюється.

Коли клієнт завершив надсилання даних, бібліотека `tusdotnet` викликає подію `OnFileCompleteAsync`, яка обробляється на двох рівнях. Спочатку `TusS3Store.CompleteUploadAsync` завершує багаточастинне завантаження в S3, надсилаючи `CompleteMultipartUploadRequest` з переліком всіх збережених ETags і видаляючи відповідні записи з Redis. Після цього спрацьовує обробник в `BlobTransferService.OnBlobUploadCompleteAsync`, який отримує метадані завантаження, витягує з них хеш-ідентифікатор блобу та розмір файлу, і зберігає запис в таблицю `blob_metadata` у базі даних `VcsDb`. Перед збереженням виконується перевірка: якщо запис з таким ідентифікатором вже існує, повторне збереження не відбувається, що захищає від дублювання при повторних спробах завантаження.

Процес завантаження вмісту файлів зі сховища є значно простішим порівняно із завантаженням і реалізований як єдиний синхронний потік

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				49
Змн.	Арк.	№ докум.	Підпис	Дата		

даних через серверний додаток до клієнта. Діаграму послідовності для процесу завантаження зображено на рис. 2.4.

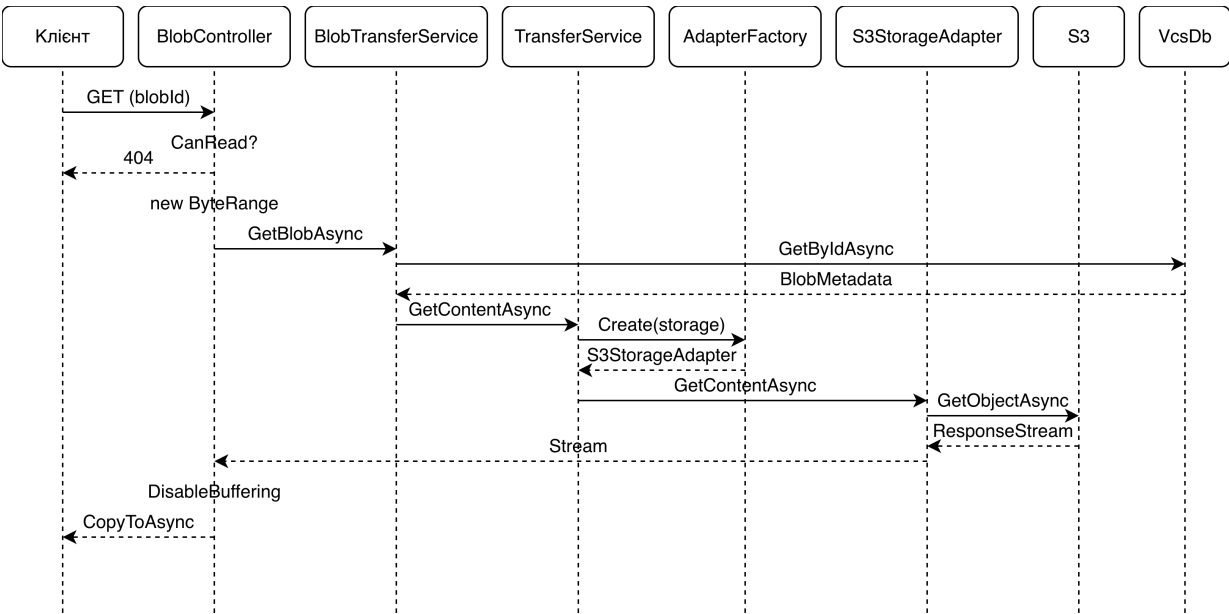


Рис. 2.4. Діаграма послідовності процесу завантаження файлів зі сховища

Запит ініціюється клієнтом через HTTP GET на ендпоінт `api/projects/{projectId}/blobs/{blobId}`, який обробляється тим самим `BlobController`. Контролер виконує перевірку прав доступу – чи має поточний користувач право на читання відповідного проєкту. Якщо проєкт є публічним або користувач має до нього доступ, запит обробляється далі; в іншому випадку повертається відповідь з кодом 404 для уникнення розкриття факту існування ресурсу.

Для підтримки часткового завантаження контролер перевіряє наявність заголовка `Range` у вхідному запиті. Якщо він присутній, значення конвертується у структуру `ByteRange` з полями `Start` та `End`, яка передається далі для обробки. Це дозволяє клієнту завантажувати великі файли частинами або відновлювати перерване завантаження з певної позиції – що є критично важливим з огляду на великі розміри аудіофайлів. Практично всі сучасні браузери підтримують і використовують цей механізм для відновлюваного завантаження файлів.

Контролер делегує отримання вмісту `servis BlobTransferService`, передаючи об'єкт проєкту, хеш-ідентифікатор блобу та структуру `ByteRange`. Сервіс спочатку перевіряє наявність метаданих блобу в базі даних `VcsDb` через репозиторій, і якщо запис відсутній, генерується помилка `NotFoundException`. Ці метадані необхідні для визначення загального розміру файлу, який використовується при формуванні заголовка `Content-Range` у відповіді. Наступним кроком формується шлях до файлу на сховищі у вигляді `vcs/{projectId}/blobs/{blobIdHex}` і в `TransferService` викликається метод `GetContentAsync`, який через `StorageAdapterFactory` отримує відповідний адаптер та звертається до `S3` із запитом `GetObjectAsync`, передаючи діапазон байтів якщо він був вказаний.

Отриманий потік даних повертається назад до контролера, який формує `HTTP`-відповідь безпосередньо перед початком передачі тіла. Для запитів без `Range` встановлюється статус `200`, а для часткових – `206 Partial Content`. Контролер також додає заголовки `Accept-Ranges`, `Content-Type`, `Content-Length` та, для часткових запитів, `Content-Range` з точними межами переданого діапазону. Якщо клієнт передав параметр `fileName`, контролер додає заголовок `Content-Disposition` з екранованою назвою файлу згідно стандарту `RFC5987`. Щоб уникнути буферизації відповіді в пам'яті сервера, буферизація вимикається явно через виклик `IHttpResponseBodyFeature DisableBuffering()`, після чого вміст потоку безпосередньо копіюється в тіло відповіді через `stream.CopyToAsync(Response.Body)`. Аналогічний підхід використовується для завантаження вмісту файлів з релізів.

Ці алгоритми завантаження вмісту файлів введено в систему, в першу чергу, як частину більших операцій – отримання оновлень від серверу (`Pull`) і завантаження оновлень на сервер (`Push`). Ці дві операції є фундаментальними для системи і охоплюють обидва додатки. Діаграму послідовності `Push`-операції наведено нижче на рис. 2.5.

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

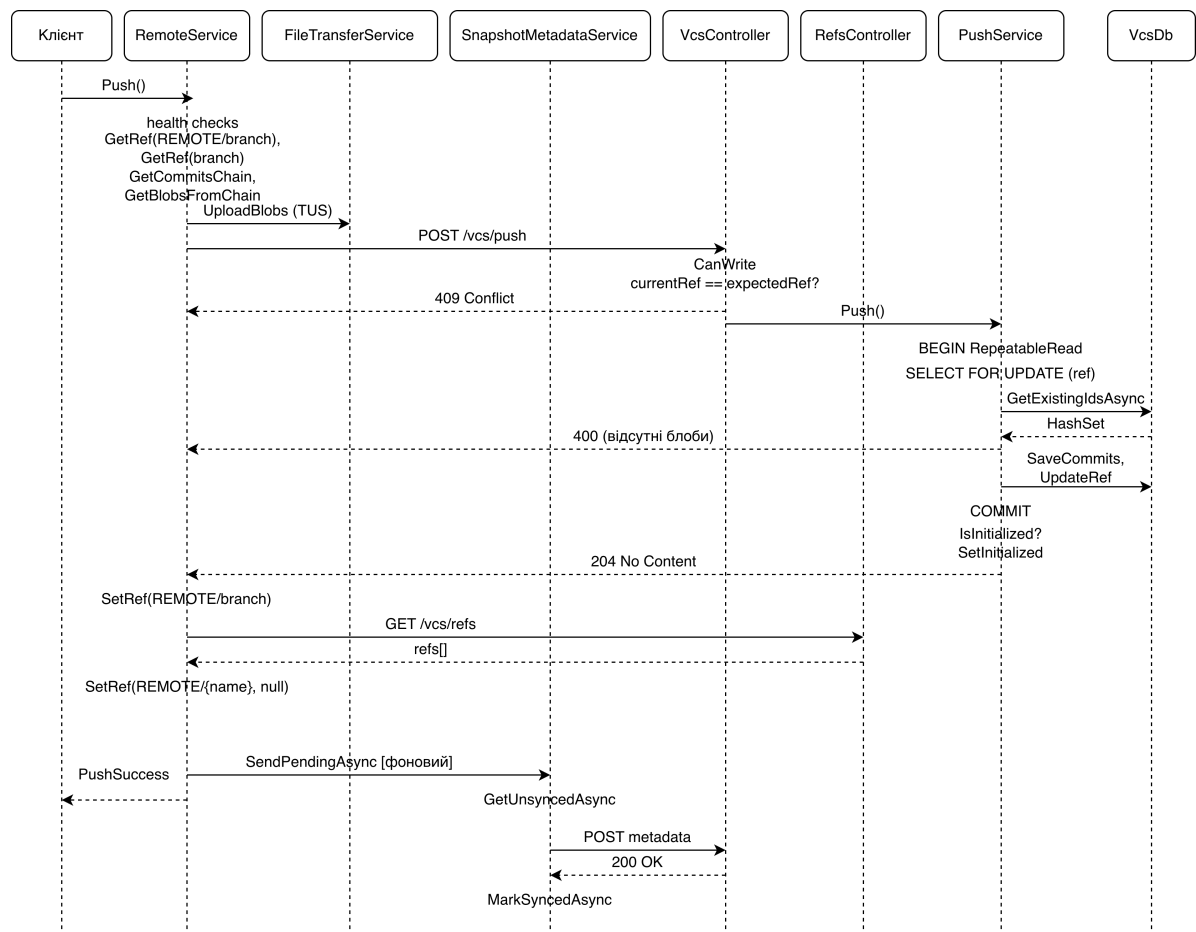


Рис. 2.5. Діаграма послідовності операції Push

Завантаження оновлень на сервер починається на стороні віконного додатка. Щоб оптимізувати запити, віконний і серверний додатки обмінюються лише оновленнями, а не повною історією версій з певної гілки. Для цього віконний додаток відслідковує не лише локальні значення вказівників гілок, а й останнє синхронізоване з сервером значення для кожного з них, додаючи префікс REMOTE/ до назви гілки в локальній базі даних.

При ініціюванні операції RemoteService у віконному додатку спочатку перевіряє з'єднання з сервером, а також доступність сховища проєкту за допомогою health-ендпоінтів. Якщо сервер і сховище доступні – формується ланцюг версій від останнього синхронізованого з сервером коміту до останнього локального. З цього ланцюга виділяються ідентифікатори всіх залучених блобів, після чого для кожного з них через FileTransferService

ініціюється завантаження за попередньо описаним алгоритмом. Завдяки відновлюваній природі протоколу TUS, перерване завантаження може бути продовжено з місця зупинки без повторної передачі вже завантажених даних. Після успішного завантаження всіх блобів за допомогою ApiClient на сервер надсилається запит, який містить ланцюг нових комітів, а також очікуване значення вказівника гілки на момент відправки.

Далі обробка операції продовжується на сервері. Спочатку VcsController перевіряє існування проєкту, дані користувача та його право на запис. Якщо ці перевірки пройдено – порівнюється поточне значення вказівника гілки в базі даних із тим, яке надав віконний додаток як очікуване. Якщо ці значення не збігаються, це означає, що віконний додаток оперує застарілими даними, запит відхиляється з відповідним кодом помилки, і віконний додаток має спершу отримати останні оновлення від сервера операцією Pull.

Якщо вказівники гілок збігаються – починається обробка комітів. Якщо надіслано не порожній ланцюг, PushService починає транзакцію з рівнем ізоляції RepeatableRead і блокує запис вказівника гілки запитом SELECT FOR UPDATE. Завдяки цьому всі інші одночасні операції запису очікують завершення поточної, що унеможлиблює пошкодження даних за умов паралельних Push-запитів від різних клієнтів.

Наступним кроком PushService перевіряє існування всіх блобів, залучених у нових комітах, викликом GetExistingIdsAsync – метод повертає множину ідентифікаторів блобів, наявних у базі даних. Якщо хоча б один із необхідних блобів відсутній, запит відхиляється, оскільки відсутність запису в blob_metadata розцінюється як відсутність вмісту на сховищі. Якщо ж існування всіх блобів підтверджено, нові коміти зберігаються в базу даних і значення вказівника гілки оновлюється, після чого транзакція завершується.

Після завершення транзакції перевіряється, чи є проєкт ініційованим. Якщо ні – поточна операція є першою для цього проєкту, і поле IsInitialized

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				53
Змн.	Арк.	№ докум.	Підпис	Дата		

оновлюється до значення true. Після цього клієнту повертається відповідь зі статусом 204 No Content.

Отримавши підтвердження успішної операції, віконний додаток виконує додаткові оновлення. За допомогою запиту на ендпоінт `/projects/{projectId}/vcs/refs` отримується перелік всіх гілок, відомих серверу. Гілки, які вже існують локально, ігноруються, а нові зберігаються в локальній базі даних зі значенням вказівника null, що відображує стан, за якого між клієнтом і сервером ще не відбувався обмін історією для цієї гілки.

Останнім кроком у фоновому режимі здійснюється надсилання сформованих наборів метаданих для версій. `SnapshotMetadataService` перевіряє всі локальні записи, прив'язані до комітів із щойно надісланого ланцюга. Записи, в яких поле `SyncedAt` має значення null, надсилаються на сервер, після чого `SyncedAt` оновлюється до поточного моменту синхронізації.

Операція отримання оновлень (Pull) також починається на стороні віконного додатка. Як і перед Push, `RemoteServer` спершу перевіряє доступність серверу і сховища проєкту. Далі виконується запит до `VcsController` з наданням значення вказівника гілки, на якому востаннє було здійснено синхронізацію.

`VcsController` виконує перевірку існування проєкту та прав доступу користувача до нього. Якщо ці перевірки пройдено – отримується поточне значення вказівника гілки на сервері. Якщо така гілка існує і значення отримано, формується ланцюг комітів від вказівника, отриманого від віконного додатка, до поточного вказівника гілки на сервері. З отриманого ланцюга формується перелік залучених ідентифікаторів блобів, для яких отримуються метадані. Ці метадані блобів і ланцюг нових комітів надсилається у відповіді віконному додатку.

Отримавши відповідь з успішним статус-кодом, що вказує на те, що всі перевірки на існування і права доступу пройдено, `RemoteService` оновлює перелік гілок, аналогічно до того як це здійснюється наприкінці Push-

		Регенель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				54
Змн.	Арк.	№ докум.	Підпис	Дата		

опреации. Після цього отриманий перелік блобів передається методу DownloadBlobs, який виявляє ідентифікатори блобів, які ще не наявні локально, завантажує їх за допомогою FileTransferService і зберігає їх метадані в локальній базі даних.

З отриманого ланцюга нових комітів виділяється ідентифікатор останнього, і керування передається на виконання команди ApplyPullCommand. Ця команда перевіряє, чи є конфлікт між локальною історією комітів і отриманою від серверу. Якщо конфліктів немає – виконання команди продовжується, і нові коміти додаються до локальної бази даних в порядку починаючи від найстарішого.

Коли коміти додано, оновлюються значення обох вказівників гілки – звичайного, який представляє локальний стан гілки, і з префіксом REMOTE/. Далі керування передається на виконання CheckoutCommand, щоб перемикнути стан робочої директорії на останні отримані від серверу оновлення.

Якщо ж на початку роботи ApplyPullCommand виявлено конфлікт локальної історії і отриманих від серверу оновлень, виконання передчасно переривається, а RemoteService передає клієнту відповідний об'єкт результату, який включає в себе ланцюг нових комітів та нове значення вказівника гілки. На цьому робота метода Pull в RemoteService завершується, і таким чином у випадку виявлення конфліктів зміни в проєкт не вносяться автоматично.

Клієнт, отримавши об'єкт результату конфлікту, відображує користувачеві повідомлення про виявлений конфлікт, і пропонує скасувати отримання оновлень і не вносити змін в локальну історію, перезаписати локальну історію на отримані від серверу оновлення, або здійснити перебазування локальних змін (rebase). Якщо обрано скасування, операція просто завершується, оскільки ніяких змін на цей момент ще не було внесено через виявлений конфлікт.

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				55
Змн.	Арк.	№ докум.	Підпис	Дата		

Якщо обрано перезапис локальної історії, виконується команда `OverwriteHistoryCommand`. Спочатку перевіряється, чи не містить робоча директорія незбережених змін, щоб передчасно зупинити операцію, якщо вони наявні. Далі знаходиться спільний батьківський коміт для останніх комітів з локальної історії та отриманих оновлень. Якщо спільної точки в історії не знайдено – перевіряється, чи містить новий ланцюг комітів від серверу кореневий коміт. Якщо ні – операція зупиняється з помилкою `InvalidOperationException`.

Після цього команда зберігає нові коміти в локальній базі даних та оновлює вказівник локальної гілки, а керування передається на виконання `CheckoutCommand`, аналогічно до `PullCommand`. Після цього вказівник гілки з префіксом `REMOTE/` оновлює оркестратор, що викликав команду.

Якщо ж обрано перебазування локальних змін (`rebase`), виконується команда `RebaseCommand`. Як і у випадку `OverwriteHistoryCommand`, спочатку виконується перевірка на незбережені зміни. Далі знаходиться спільний батьківський коміт для обох ланцюгів. Якщо такий коміт не знайдено – додатково виконується перевірка на наявність нового кореневого коміта в отриманих оновленнях. В обох випадках, отримуємо визначені ланцюги історії, наявної виключно локально, і отриманої від серверу.

Щоб з'єднати ці два ланцюги в одну повноцінну історію версій, потрібно згенерувати спеціальний коміт-відкат (`revert`), який скасує зміни, внесені в отриманих оновленнях, щоб після нього можна було доєднати локальні зміни, не спотворивши кінцевий стан гілки. Для цього формується стан проєкту з накладенням змін з отриманих оновлень, а також стан спільної для обох ланцюгів точки в історії, якою є або стан на момент спільного батьківського коміту, або порожній стан у випадку повного конфлікту історій гілок.

За допомогою `DiffService` обраховується різниця цих двох станів. При цьому стан сформований з отриманих оновлень використовується як стан «до» (`beforeSnapshot`), а стан зі спільної точки в історії як стан «після»

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				56
Змн.	Арк.	№ докум.	Підпис	Дата		

(afterSnapshot). В результаті отримуємо набір змін, які необхідно застосувати після історії, яка відповідає серверу, щоб повернути стан проєкту до останньої спільної точки, на основі якої будувалися всі суто локальні коміти. Цей перелік змін зберігається в новому автоматично згенерованому коміті, який має тип (kind) Revert, і батьківським комітом для нього зазначається останній коміт з отриманих оновлень. Автором цього коміту зазначається поточний користувач.

Ідентифікатори комітів є хешами їх даних, включаючи ідентифікатор батьківського коміту, а отже просто змінити батьківський коміт першого суто-локального коміту не можна, оскільки це порушить цілісність даних. Тому для ланцюга суто-локальних комітів створюється повна копія, в коміті якої дублюється все з оригінальних комітів, крім ідентифікаторів батьківських комітів, оскільки їх змінено.

З'єднавши спільну для сервера і віконного додатка історію, отримані від сервера оновлення, згенерований revert-коміт і копію ланцюга суто-локальних комітів, формується повна історія гілки. Такий підхід зберігає всі версії в історії, що дозволяє перемикнуватися на них і використовувати за потреби, але водночас зберігає зміни, які були внесені локально, і дозволяє продовжити роботу над ними. Фактично, перебазування гілки відображує явний намір користувача обрати локальний стан проєкту замість збереженого на сервері. Отримані оновлення, revert-коміт і копії суто-локальних комітів зберігаються в локальній базі даних, після чого вказівник гілки оновлюється на останній коміт. Як і у випадку OverwriteHistoryCommand, керування передається на виконання CheckoutCommand, після чого оркестратор оновлює вказівник гілки з префіксом REMOTE/.

2.4 Реалізація компонентів системи

Серверний додаток розділено на проєкти Api та Lib. Проєкт Api є точкою входу – він містить лише файл Program.cs, в якому здійснюється реєстрація залежностей та конфігурація додатка [10]. Всю бізнес-логіку

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		57

модулів розміщено в Lib, що дозволяє в майбутньому підключити інші точки входу без дублювання коду (наприклад, при розділенні API на публічне і приватне). Схематично структуру серверного додатка зображено нижче на рис. 2.6.

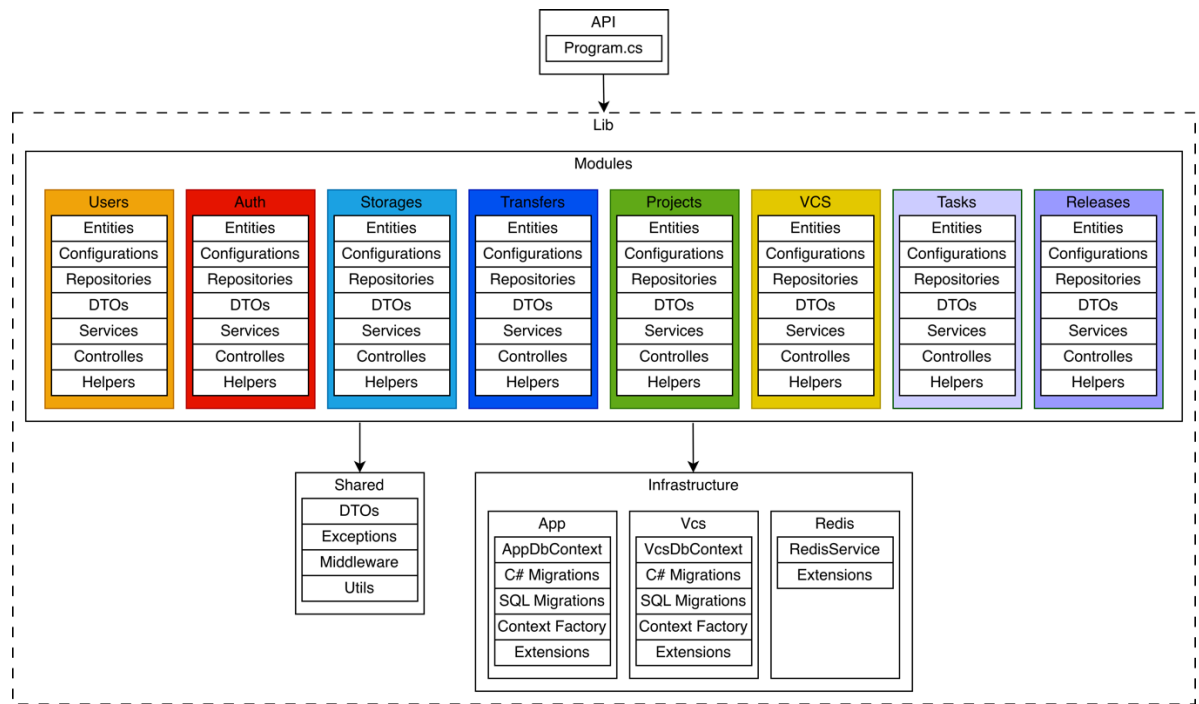


Рис. 2.6. Структура серверного додатка

Бібліотека Lib організована за допомогою трьох директорій, з яких Infrastructure відповідає за компоненти необхідні для роботи з базами даних, Modules містить всі модулі, виділяючи кожному окрему піддиректорію, а Shared містить спільні для всіх модулів компоненти.

Зокрема, директорія Infrastructure містить технічну інфраструктуру, незалежну від бізнес-логіки. Це включає два контексти баз даних (AppDbContext і VcsDbContext) з міграціями та SQL-скриптами для Flyway, а також сервіс для роботи з Redis (RedisService). Два окремі контексти відображають логічне розділення даних, описане в розділі 2.2. Винесення інфраструктури в окрему директорію ізолює технічні деталі роботи з базами даних від бізнес-логіки, що зосереджена в модулях. Завдяки цьому зміна способу доступу до даних не потребує втручання в доменну логіку модулів.

Директорія Modules є основною і розбита на вісім модулів, а саме Users, Auth, Storages, Transfers, Projects, Tasks, Releases та Vcs. Всі модулі внутрішньо притримуються єдиних правил організації по шарах, таких як Controllers, Services, Repositories, Entities, DTOs та Configurations. Такий підхід реалізує вертикальне розбиття за доменом, а в ньому горизонтальне розбиття за шаром. Тому всі файли, що стосуються певного функціоналу зосереджені в одному місці і організовано за роллю файлу в модулі.

Всередині кожного модуля дотримано чіткий поділ на шари і їх відповідальності. Таким чином, контролери приймають HTTP-запити та делегують роботу сервісам, сервіси містять бізнес-логіку та взаємодіють з репозиторіями, репозиторії інкапсулюють запити до бази даних через EF Core. Взаємодія між шарами здійснюється виключно через інтерфейси, що дозволяє замінювати реалізації незалежно.

Реєстрацію залежностей та ініціювання кожного модуля винесено в окремий клас з суфіксом Module (наприклад, VcsModule, AuthModule), який підключається в Program.cs. Цей клас містить методи-розширення для реєстрації залежностей модуля, їх конфігурації, а також для додавання конфігурацій сутностей в контекст бази даних та налаштування на кшталт конвертерів для власних типів даних.

Директорія Shared містить спільні для всіх модулів компоненти, такі як спільні DTO для пагінації, класи помилок (NotFoundException, ForbiddenException тощо), а також GlobalExceptionHandlerMiddleware що перехоплює виключення і перетворює їх на відповідні HTTP-відповіді.

Роботу серверного додатка зі сховищами можна розглядати як окрему підсистему. Підсистема сховищ складається з двох модулів, з яких модуль Storages відповідає за управління сховищами як сутностями вебплатформи, а модуль Transfers відповідає за безпосередню передачу даних через адаптери з використанням цих сховищ. Узагальнену структуру підсистеми роботи зі сховищами наведено нижче на рис. 2.7.

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				59
Змн.	Арк.	№ докум.	Підпис	Дата		

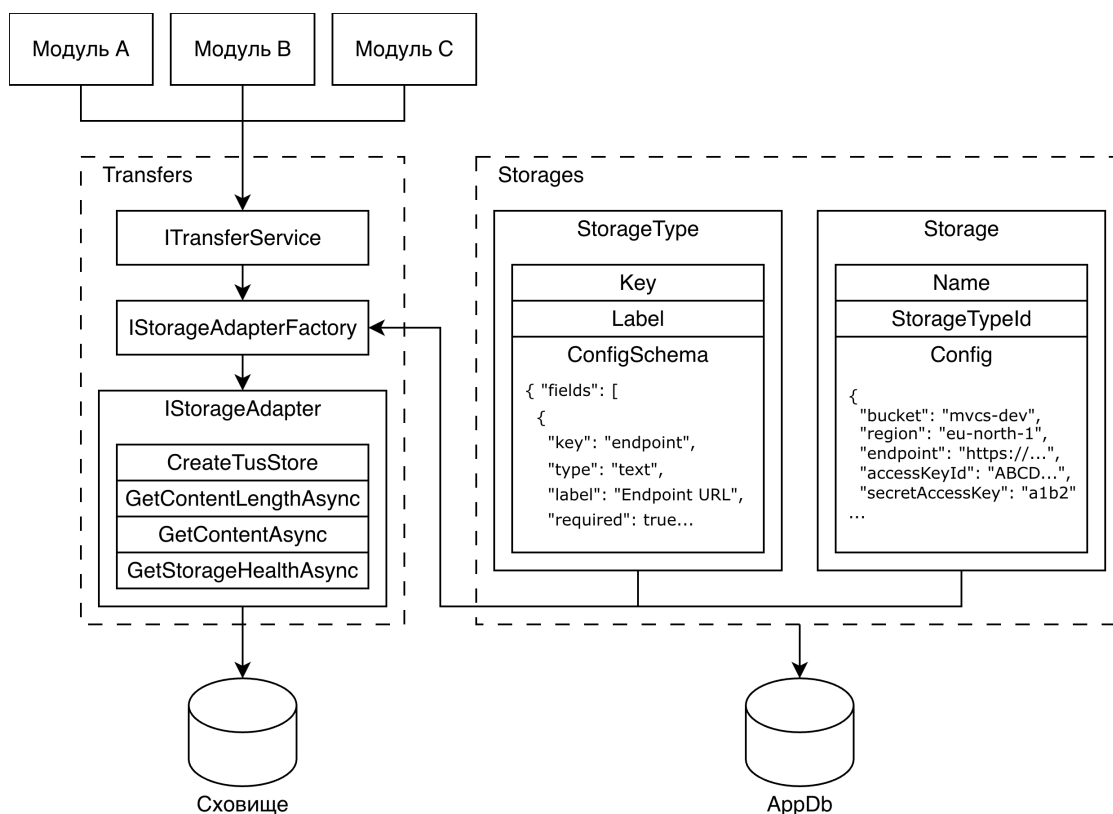


Рис. 2.7. Структура підсистеми роботи зі сховищами

Модуль Storages зберігає дані про сховища в AppDbContext. Сутність StorageType представляє підтримуваний тип сховища (наприклад, aws-s3) і містить програмний ключ Key, відображувану назву Label та ConfigSchema – JSON-схему, яка описує які поля потрібно заповнити для підключення сховища цього типу. Схема складається з об'єктів SchemaField різних підтипів (TextSchemaField, PasswordSchemaField, BooleanSchemaField, NumberSchemaField, SelectSchemaField), кожен з яких містить правила валідації та інші налаштування, які використовуються для динамічної побудови форми на стороні веб-клієнта, і при додаванні нового підтримуваного типу сховища, зміни в код веб-клієнта вносити не потрібно. Також ця схема визначає поведінку серверного додатка, наприклад, стосовно валідації отриманих від клієнта конфігурацій, або приховування певних полів при формуванні відповідей.

Сутність Storage представляє конкретне сховище і містить назву, посилання на тип, а також Config – JSON-рядок з фактичними параметрами

		Регенель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				60
Змн.	Арк.	№ докум.	Підпис	Дата		

для підключення до цього конкретного сховища (endpoint, ключі, назва бакету тощо). Це дозволяє кожному типу сховища мати власну структуру. Доступ до сховища визначається за допомогою сутності StorageAccess з рівнями ReadWrite та Owner. Сама сутність Storage інкапсулює логіку перевірки прав через методи CanRead, CanWrite та IsOwnedBy для зручності використання.

В модулі Transfers є реалізація патерну «Адаптер» для абстрагування роботи з конкретними видами сховищ. Інтерфейс IStorageAdapter визначає операції, які потрібні бути реалізовані для підтримуваного сховища. Зокрема, CreateTusStore для ініціації завантаження, GetContentAsync для отримання вмісту з підтримкою ByteRange, GetContentLengthAsync для отримання розміру файлу та GetStorageHealthAsync для перевірки доступності сховища. Наразі реалізовано один адаптер – S3StorageAdapter, який при створенні отримує десеріалізований об'єкт S3StorageConfig та створює AmazonS3Client з відповідними обліковими даними.

StorageAdapterFactory відповідає за створення адаптера на основі сутності Storage. Цей клас зчитує ключ типу сховища з навантаженої навігаційної властивості StorageType.Key, десеріалізує поле Config у відповідну модель конфігурації та повертає готовий адаптер. Додавання підтримки нового типу сховища зводиться до реалізації IStorageAdapter, додавання відповідного case у фабрику та реєстрації типу в базі даних.

Така організація відповідає принципу відкритості/закритості, оскільки розширення системи новим типом сховища не потребує зміни наявного коду фабрики чи адаптерів, а лише додавання нових. Це знижує ризик внесення помилок у вже працюючий функціонал при додаванні підтримки нових сховищ. Сам перелік підтримуваних типів при цьому залишається керованим через базу даних, а не описаним виключно в коді.

TransferService виконує роль фасаду між рештою системи та адаптерами. Він завантажує сутність Storage з бази через IStorageService, делегує створення адаптера фабриці та здійснює виклики методів у

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

відповідного адаптера. Завдяки цьому жоден з модулів, що використовують передачу файлів, не залежить безпосередньо ні від StorageAdapterFactory, ні від конкретних адаптерів, і працює виключно через ITransferService.

Таким чином дотримано високу ізоляцію модулів один від одного, за якої модулі вільно використовують свої внутрішні компоненти, але з інших модулів можуть використовувати тільки сервіси. В майбутньому при потребі масштабувати систему і винести окремий модуль в окремий контейнер або фізичний сервер, програмні зміни зводяться до реєстрації нової реалізації інтерфейсів сервісів вилученого модуля, наприклад, на основі gRPC або іншого способу комунікації серверів між собою.

Розглянуті вище компоненти стосуються серверного додатка, проте значна частина роботи з передачею файлів та синхронізацією виконується на стороні віконного додатка. Далі описано ключові компоненти на стороні клієнта, побудовані на єдиному механізмі взаємодії з сервером.

Для взаємодії з серверним додатком у клієнтському додатку реалізовано клас ApiClient. Його роль полягає в оновленні access-токена при його інвалідації [18]. При отриманні відповіді від сервера спеціальна функція-предикат, надана при конфігурації, визначає чи слід оновити токен і повторити запит. Якщо так – всі нові запити утримуються в черзі за допомогою TaskCompletionSource, виконується запит на оновлення токенів, і залежно від результату всі запити в черзі разом з початковим або надсилаються повторно з оновленим токеном, або відхиляються з відповідною помилкою. Завдяки цьому жоден з сервісів, що використовує ApiClient, не потребує власної логіки оновлення токенів при їх інвалідації.

На основі ApiClient побудовано TusClient, який містить реалізацію протоколу TUS на стороні клієнта. Клас реалізує всі основні операції протоколу, такі як Create для ініціації завантаження, Patch для надсилання фрагментів даних, Head для отримання поточного зміщення при відновленні перерваного завантаження. Завдяки цьому завантаження файлів на сховище

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

є відновлюваним – при перериванні з’єднання клієнт може продовжити передачу з точки зупинки без повторної передачі вже завантажених даних.

Керування всіма операціями передачі файлів в межах додатка централізовано в FileTransferService. Сервіс взаємодіє з сервером через ApiClient та TusClient, надає спрощені методи для завантаження файлів на сховище та зі сховища, а також керує чергою паралельних передач за допомогою SemaphoreSlim з налаштованим лімітом одночасних завантажень для контролю використання мережевого трафіку. Окрім цього, сервіс надає перелік завантажень, їх прогрес та стан для відображення у графічному інтерфейсі на панелі завантажень.

За синхронізацію локальних даних проєкту з сервером відповідає RemoteService. Він реалізує операцію Push для завантаження локальних оновлень на сервер, операцію Pull для отримання оновлень з сервера, та Clone для копіювання проєкту разом з переліком гілок з сервера у новий локальний проєкт. Кожна з цих операцій включає не лише обмін метаданими комітів і вказівниками гілок, а й завантаження або надсилання вмісту файлів, пов’язаних з відповідним ланцюгом версій, з використанням FileTransferService та BlobService. Також саме цей сервіс відповідає за відслідковування стану сервера і сховища, виконуючи необхідні запити на health-ендпоінти і змінюючи відповідні значення в RemoteViewModel.

Конфігурація додатка організована на двох рівнях з різними стратегіями зберігання. Глобальна конфігурація, яка стосується всього додатка, зберігається в JSON-файлі локально і керується через LocalConfigService. При ініціалізації сервіс завантажує файл у пам’ять і всі подальші звернення обслуговуються з кешу без звернення до диска. При кожному записі сервіс серіалізує лише ті ключі, чиї простори імен (або скоупи) позначені як ті, що потрібно зберігати, що дозволяє зберігати тимчасові runtime-значення в пам’яті без їх запису на диск. Конфігурація конкретного репозиторію зберігається безпосередньо в SQLite-базі проєкту через SqliteConfigService, який не кешує значення і кожного разу звертається

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		63

до бази. Усі ключі конфігурації зосереджені в статичних класах ConfigKeys та RepoConfigKeys з ієрархічною структурою скоупів задля уникнення проблеми «магічних рядків» при роботі з конфігурацією.

Управління активним проєктом централізовано в RepositoryService. Сервіс зберігає посилання на поточний відкритий екземпляр класу CoalescingRepository, керує його відкриттям та закриттям і дублює події репозиторію через власні події OnRepositoryChanged, OnCommitAsync та OnCheckoutAsync, на які підписуються ViewModel-класи додатка. RepositoryFactory відповідає за фактичне створення екземплярів класу: збирає всі залежності, створює робочу директорію з вказаного шляху та зчитує файл правил ігнорування якщо він налаштований. Перед поверненням проєкт огортається в CoalescingRepository – обгортку, яка об'єднує одночасне виконання однакових команд. Це використовується для довготривалих процесів, таких як отримання статусу робочої директорії, які можуть бути викликані з декількох ViewModel-класів одночасно. Якщо команда певного типу вже виконується і надходить ще один запит того самого типу, другий запит приєднується до вже існуючого Task через SemaphoreSlim і отримує той самий результат без повторного виконання.

ActiveRepoViewModel є центральним джерелом стану відкритого проєкту для всього додатка. Цей клас підписується на події RepositoryService і підтримує актуальний список гілок та назву поточної гілки, оновлюючи їх при кожній операції коміту, перемикання або явного запиту оновлення. ActiveRepoBarViewModel є складнішим класом для панелі керування відкритим проєктом, який доповнює стан з ActiveRepoViewModel та RemoteViewModel і керує взаємодією з гілками та операціями синхронізації. При вибраній гілці з префіксом REMOTE/ та порожнім вказівником клас автоматично ініціює Pull для отримання її вмісту з сервера. RemoteViewModel окремо відстежує три незалежні умови доступності серверної частини, де IsRemoteSet вказує на наявність прив'язки до проєкту на сервері, IsServerOnline вказує на доступність сервера, а IsStorageOnline

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				64
Змн.	Арк.	№ докум.	Підпис	Дата		

вказує на доступність сховища. RemoteStatusBar відображає повідомлення у верхній частині інтерфейсу якщо хоча б одна з умов не виконана.

Аналіз метаданих вмісту проєкту побудований на розширюваній системі аналізаторів. Інтерфейс ISnapshotAnalyzer визначає єдиний метод AnalyzeAsync, який приймає стан робочої директорії проєкту і сервіс доступу до блобів та повертає метадані у вигляді пар ключ-значення. Конкретні реалізації винесено в проєкт MetadataAnalyzers, як, наприклад, ReaperAnalyzer, який обробляє файли проєктів Reaper з розширенням .grp. Він зчитує вміст файлу через BlobTextReader, який отримує потік з IBlobService і декодує його як текст з автоматичним визначенням кодування. Після зчитування аналізатор застосовує набір регулярних виразів для виявлення версії Reaper, кількості треків, типів контенту (аудіо, MIDI), назв VST, CLAP і AU плагінів, темпу, розміру та частоти дискретизації. Допоміжний клас SnapshotFiles надає методи фільтрації файлів за розширенням, що використовується аналізаторами для відбору лише релевантних файлів.

Висновки до другого розділу

У другому розділі спроектовано та реалізовано ключові компоненти системи контролю версій. Визначено варіанти використання та об'єктно-орієнтовану модель як спільної бібліотеки, так і серверного і віконного додатків, описано підходи до вирішення конфліктів версій та збору метаданих проєкту. Розроблено схему двох баз даних, а саме AppDb для сутностей вебплатформи та VcsDb для даних системи контролю версій, з обґрунтуванням їх розділення. Описано алгоритми основних операцій системи, до яких належать завантаження і отримання вмісту файлів, Push, Pull та Rebase, а також реалізацію ключових компонентів серверного і віконного додатків, зокрема підсистему роботи зі сховищами, систему управління конфігурацією та аналізаторів метаданих.

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		65

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБСИСТЕМОЮ ТА ВІКОННИМ ДОДАТКОМ

3.1. Порядок встановлення та налаштування системи

Для розгортання серверного додатка необхідно підготувати сервер з операційною системою Linux. На сервері має бути встановлено Docker Engine версії 24 або вище та Docker Compose v2. В більш складних варіантах розгортання передбачається застосування Docker Swarm. Після клонування репозиторію в кореневій директорії проєкту створюється файл .env з переліком змінних середовища, наведеним у таблиці 3.1.

Таблиця 3.1

Перелік змінних середовища серверного додатка

Змінна	Призначення
INTERNAL_NUGET_HOST	Адреса приватного NuGet-сервера для завантаження пакету спільної бібліотеки під час збирання Docker-образу
POSTGRES_USER	Логін адміністратора PostgreSQL
POSTGRES_PASS	Пароль адміністратора PostgreSQL
POSTGRES_DB	Назва службової бази даних PostgreSQL
POSTGRES_APP_USER	Логін користувача для підключення додатка
POSTGRES_APP_PASS	Пароль користувача для підключення додатка
POSTGRES_DB_APP	Назва бази даних рівня вебплатформи (AppDb)
POSTGRES_DB_VCS	Назва бази даних системи контролю версій (VcsDb)
REDIS_PASS	Пароль для підключення до Redis
JWT_ACCESS_SECRET	Ключ для підпису access-токенів
JWT_REFRESH_SECRET	Ключ для підпису refresh-токенів

Після заповнення `.env` розгортання здійснюється однією командою “`docker compose -f docker-compose.yml up -d`”. Docker Compose послідовно запускає контейнери відповідно до визначених умов залежностей. Спочатку ініціалізується PostgreSQL, після чого виконуються міграції баз даних через Flyway – окремо для AppDb і VcsDb. Лише після успішного завершення міграцій запускається серверний додаток. Nginx запускається останнім, після того як серверний додаток успішно пройде healthcheck. Перевірити стан усіх контейнерів можна командою “`docker compose ps`”.

Для збереження та розповсюдження пакету спільної бібліотеки в межах інфраструктури використовується BaGet як легкий NuGet-сервер з відкритим кодом. Перед першим розгортанням серверного додатка необхідно розгорнути BaGet та опублікувати в ньому пакет бібліотеки. Після розгортання BaGet адреса сервера вказується у змінній `INTERNAL_NUGET_HOST` у файлі `.env`, і під час збирання Docker-образу серверного додатка пакет завантажується з цієї адреси.

Nginx налаштовано як зворотний проксі-сервер, який приймає вхідні з’єднання на портах 80 і 443 та перенаправляє запити до серверного додатка. Конфігурація розділена на окремі файли в директорії `conf.d/`, в яких запити на шлях `/api` перенаправляються до серверного додатка, а запити на `/` – до вебклієнта якщо його додано при розгортанні. HTTPS-з’єднання підтримуються на рівні Nginx, де зберігаються SSL-сертифікати, розміщені в директорії `./nginx/certs`, яка монтується в контейнер лише для читання.

Ініціалізація бази даних відбувається в два етапи. На першому етапі PostgreSQL при першому запуску автоматично виконує скрипти з директорії `./postgres/init`. Скрипт `init.sh` створює бази даних AppDb і VcsDb, якщо вони ще не існують, після чого створює окремого користувача додатка з відповідним паролем і надає йому повні привілеї на обидві бази даних та їх публічні схеми. Скрипт виконується лише при першій ініціалізації тому файли бази даних ще відсутні на диску. При повторному запуску контейнера з вже існуючим томом `postgres_data` цей скрипт не виконується.

		Регенель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				67
Змн.	Арк.	№ докум.	Підпис	Дата		

На другому етапі запускаються два контейнери Flyway, по одному для кожної бази даних. Flyway підключається до відповідної бази даних і застосовує SQL-міграції з директорій `be/Lib/Infrastructure/App/Migrations/Sql` та `be/Lib/Infrastructure/Vcs/Migrations/Sql`. Назви файлів міграцій відповідають конвенціям Flyway, наприклад `V001__init.sql` – де префікс `V` і порядковий номер визначають послідовність застосування. Кожна застосована міграція реєструється в таблиці `flyway_schema_history`, яку веде сам Flyway і яка дозволяє йому при повторному запуску пропускати вже виконані міграції, а також в таблиці `__EFMigrationsHistory`, запис до якої вбудовано в самі SQL-скрипти згенеровані засобами Entity Framework Core. Серверний додаток запускається лише після того, як обидва контейнери Flyway завершили роботу зі статусом успіху.

Для налаштування CI/CD автоматизації використовується GitHub Actions [16]. У репозиторії створюється файл `.github/workflows/deploy.yml`, який визначає автоматичне розгортання оновлень при злитті змін у гілку `main`. Workflow підключається до сервера по SSH, завантажує актуальний стан репозиторію командою `git pull` та перезапускає стек командою `“docker compose -f docker-compose.yml up -d –build”`, яка перезбирає змінені образи і замінює контейнери без зупинки незмінених сервісів. Адреса сервера, ім'я користувача та SSH-ключ зберігаються як зашифровані GitHub Secrets і не потрапляють в код репозиторію. Повний вміст workflow-файлу наведено в Додатку А.

Для розгортання вебклієнта передбачено окремий файл `docker-compose.frontend.yml`, який підключається додатково до основного стеку командою `“docker compose -f docker-compose.yml -f docker-compose.frontend.yml up -d”`. Файл додає контейнер «frontend» з образом вебклієнта. Образ збирається окремо з використанням багатоетапного Dockerfile, в якому на першому етапі виконується збірка Next.js додатка в production-режимі зі standalone-варіантом результату, а на другому формується мінімальний образ запуску, який містить лише зібраний додаток

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				68
Змн.	Арк.	№ докум.	Підпис	Дата		

без вихідного коду та залежностей для збірки, що суттєво зменшує розмір фінального образу. При підключенні файлу вебклієнта Docker Compose також перевизначає конфігурацію Nginx, монтуючи замість базових конфігураційних файлів розширені варіанти, які додають перенаправлення запитів на кореневий шлях / до контейнера вебклієнта на порт 3000, тоді як маршрутизація запитів на /api до серверного додатка залишається незмінною.

3.2. Структура інтерфейсів клієнтських додатків

Віконний додаток здатен працювати без з'єднання з сервером, надаючи весь функціонал контролю версій локально. При запуску додатка, користувач за замовчуванням потрапляє на головну сторінку, на якій доступні опції створення нового проєкту, відкриття вже існуючого локального проєкту або клонування проєкту з серверу (рис. 3.1).

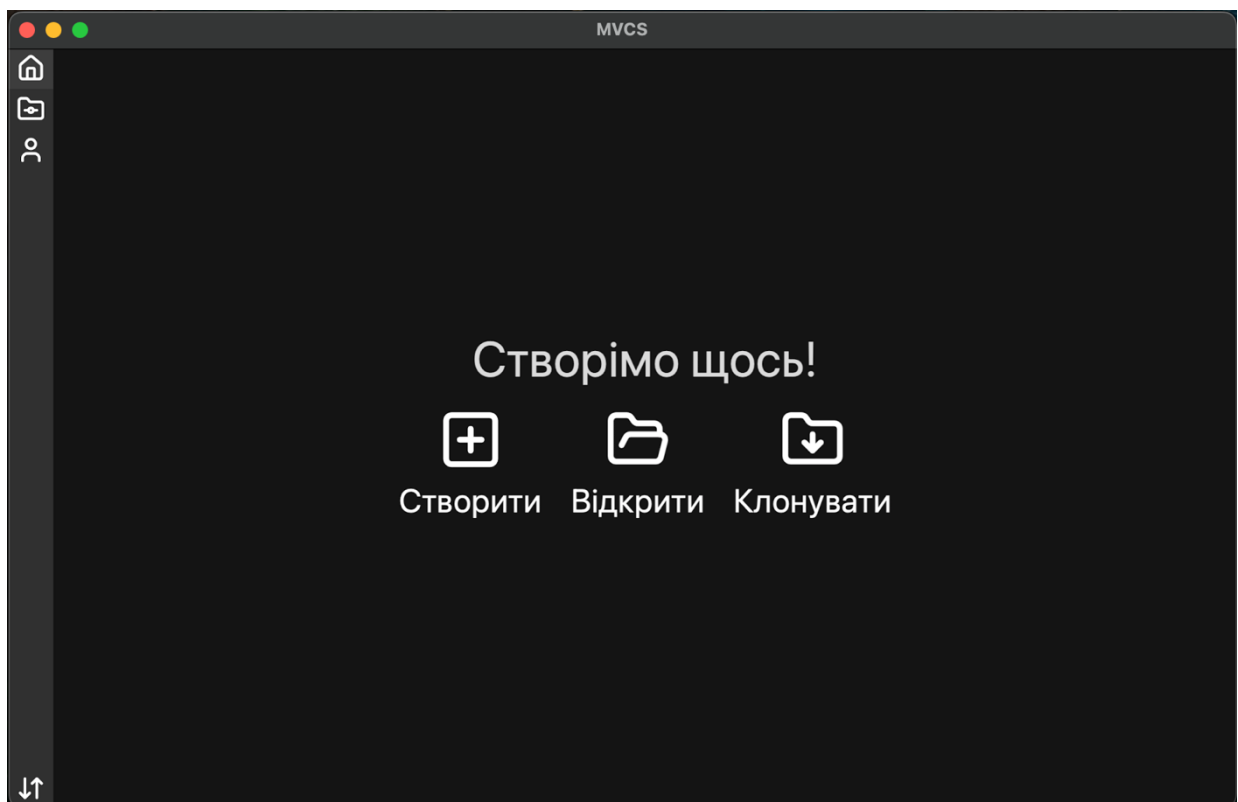


Рис. 3.1. Головна сторінка віконного додатка

		Регенель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				69
Змн.	Арк.	№ докум.	Підпис	Дата		

При створенні або відкритті проєкту викликатиметься меню вибору директорії надане операційною системою. При клонуванні відкривається окреме меню, в якому користувач вводить ідентифікатор проєкту для клонування, та обирає директорію, в якій буде клоновано проєкт (рис. 3.2).

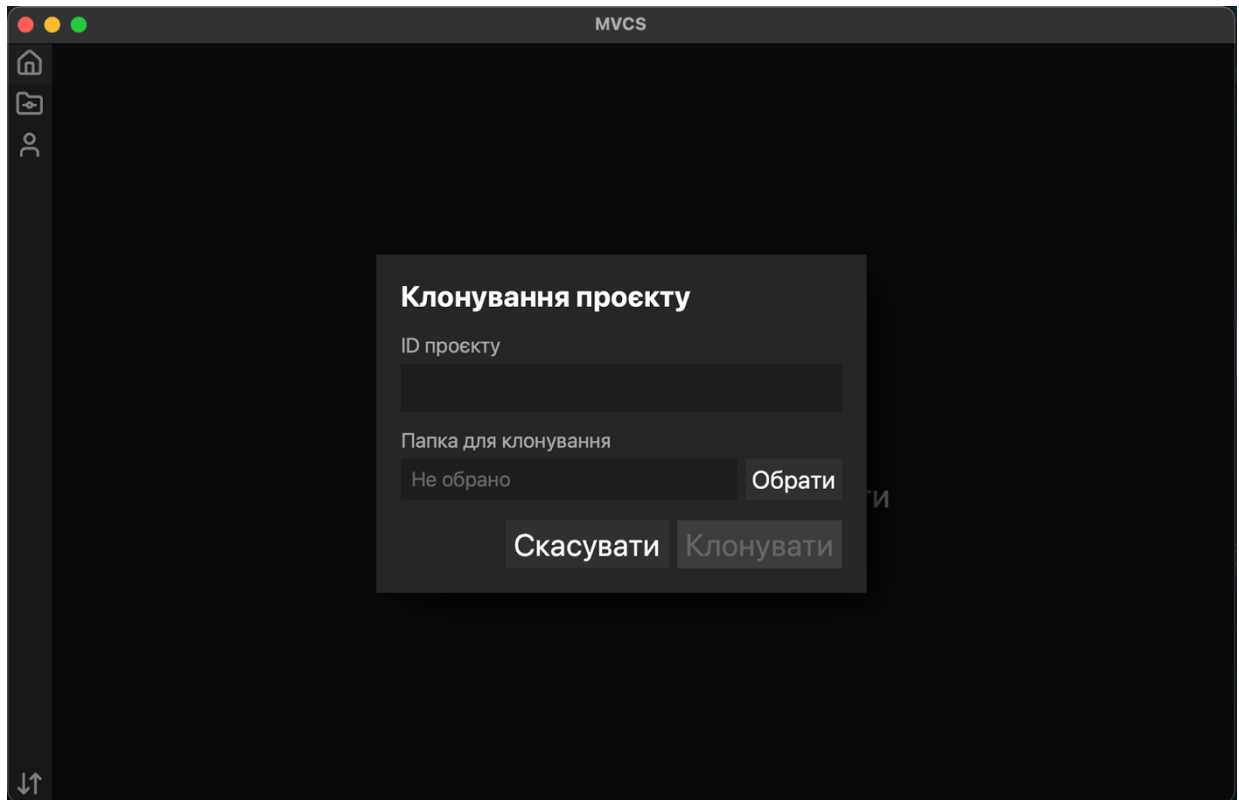


Рис. 3.2. Меню клонування проєкту

При успішному виконанні будь-якої з трьох запропонованих дій, користувача перенаправляє на вкладку активного проєкту (рис. 3.3). У верхній частині цієї вкладки розташована панель керування активним проєктом. Ця панель містить кнопку для примусового оновлення стану, шлях до відкритого проєкту, назву поточної гілки з можливістю перемикавання на іншу або створення нової, кнопки для операцій Push, Pull, а також відкриття налаштувань проєкту.

Горизонтально вкладку поділено на дві панелі, з яких ліва є панеллю історії, а права – панеллю збереження нової версії. На панелі історії відображується історія версій для поточної гілки, зеленою іконкою

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

позначається поточна версія, а інші версії мають кнопку для перемикання на них. Для кожної версії зазначається її автор, дата створення та повідомлення.

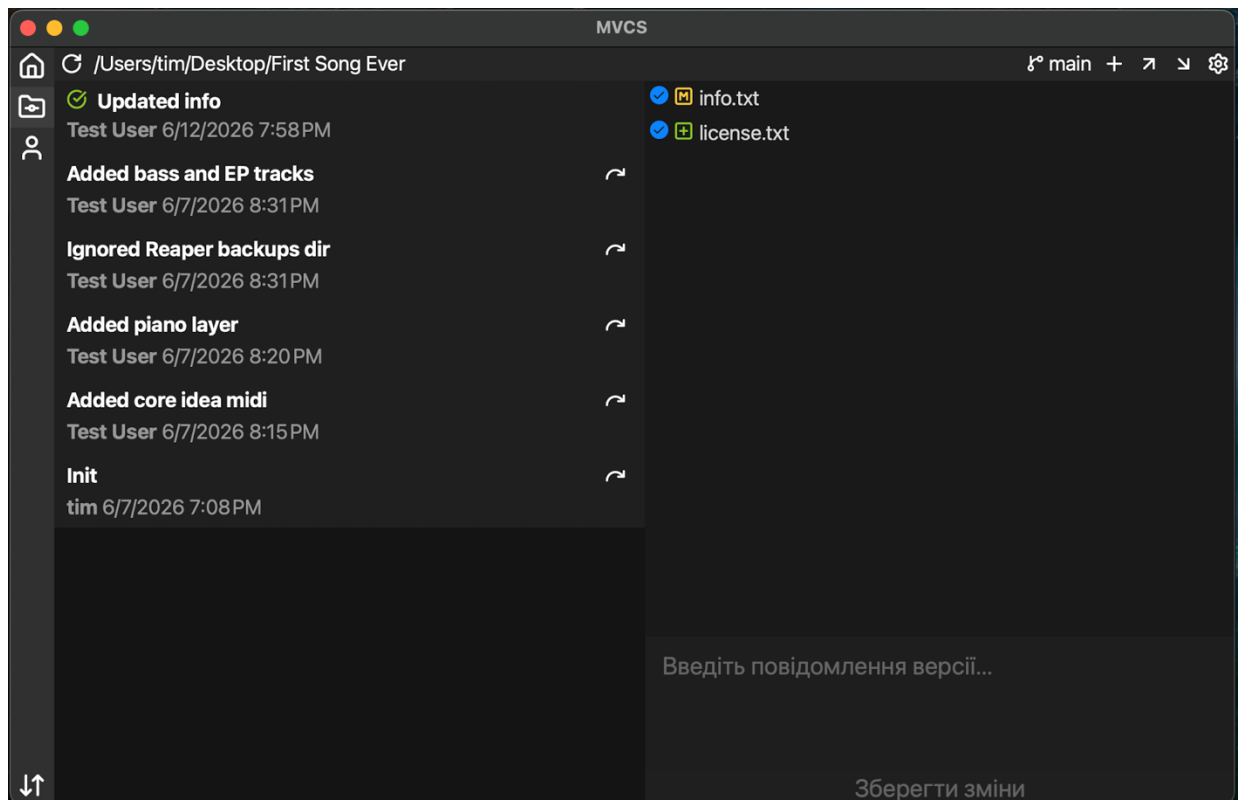


Рис. 3.3. Вкладка активного проєкту

На панелі збереження нової версії відображується перелік виявлених змін. Біля кожного файлу іконкою позначено тип зміни, щоб явно розрізняти створені, видалені і змінені файли. Кожен файл можна включити або виключити з майбутньої версії, щоб керувати, які саме зміни буде збережено в історію. Нижче розміщено поле для введення повідомлення версії. При невалідному значенні під цим полем відображується повідомлення про помилку. В нижній частині панелі розміщено поле для вибіру задачі, до якої буде прив'язано нову версію (необов'язково), а також кнопка для збереження нової версії.

Меню налаштувань проєкту наразі містить одне поле – ідентифікатор для прив'язки локального проєкту до проєкту на сервері (рис. 3.4). Без цього ідентифікатора операції Push та Pull недоступні.

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				71
Змн.	Арк.	№ докум.	Підпис	Дата		

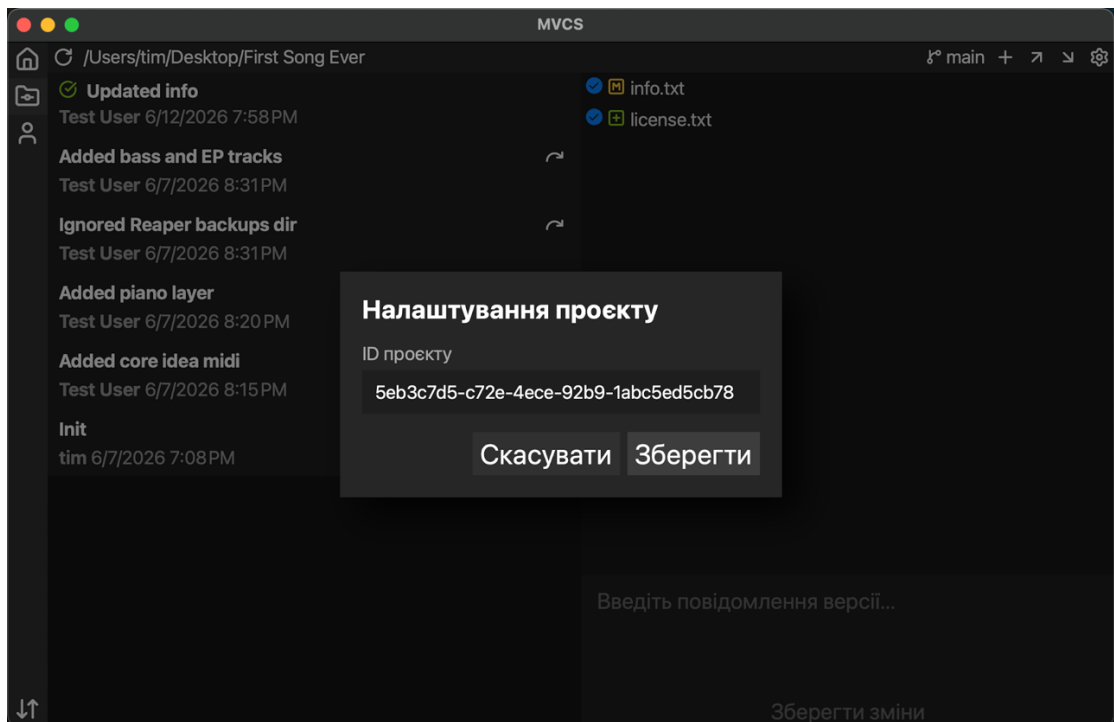


Рис. 3.4. Меню налаштувань проєкту

При створенні нової гілки користувач має ввести її назву (рис. 3.5). Ця назва перевіряється на предмет конфлікту з системними вказівниками (HEAD) або вже існуючими гілками.

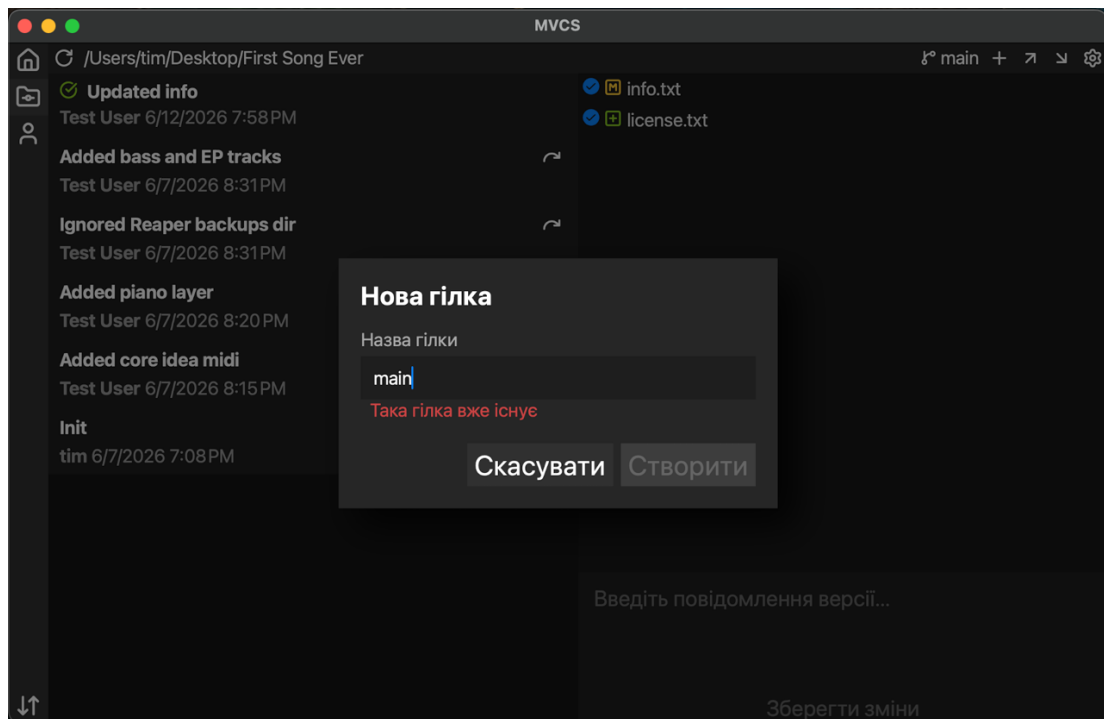


Рис. 3.5. Меню створення нової гілки

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				72
Змн.	Арк.	№ докум.	Підпис	Дата		

Якщо під час отримання оновлень з серверу виявлено конфлікт, відображується меню вирішення конфлікту(рис. 3.6). При виборі опції перезапису або перебазування користувачеві відображується повідомлення про наслідки цих операцій для отримання підтвердження.

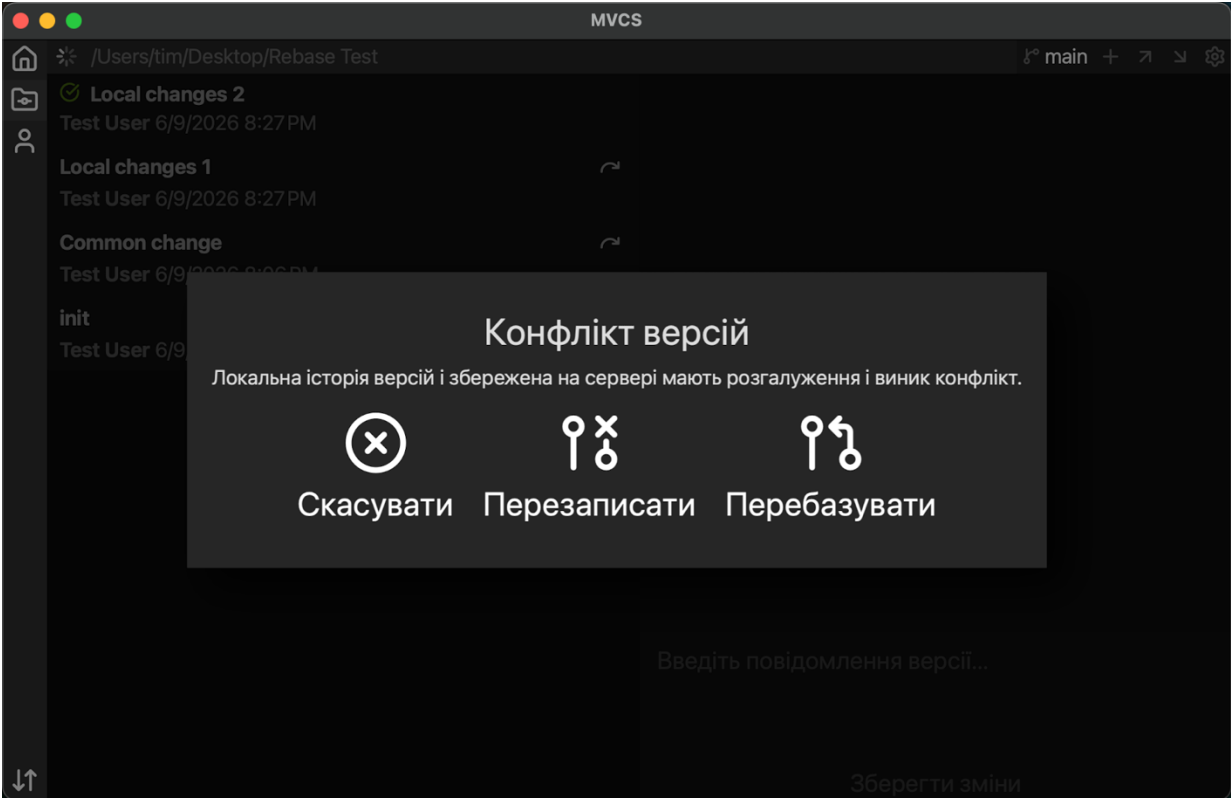


Рис. 3.6. Меню вирішення конфліктів

В лівій нижній частині інтерфейсу розміщено вкладку передачі файлів. В ній в реальному часі відображується перелік активних завантажень зі сховища або на сховище (рис. 3.7).

Для кожного завантаження зазначається його тип за допомогою іконки (стрілка вниз вказує на завантаження на пристрій користувача, а стрілка вгору – на завантаження на сховище), назва файлу який завантажується, прогрес завантаження у відсотках. Якщо завантаження успішно завершене – воно помічається зеленим кольором. Якщо ж виникає помилка, застосовується червоний колір, а під смугою прогресу відображується повідомлення про помилку.

		Регенель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				73
Змн.	Арк.	№ докум.	Підпис	Дата		

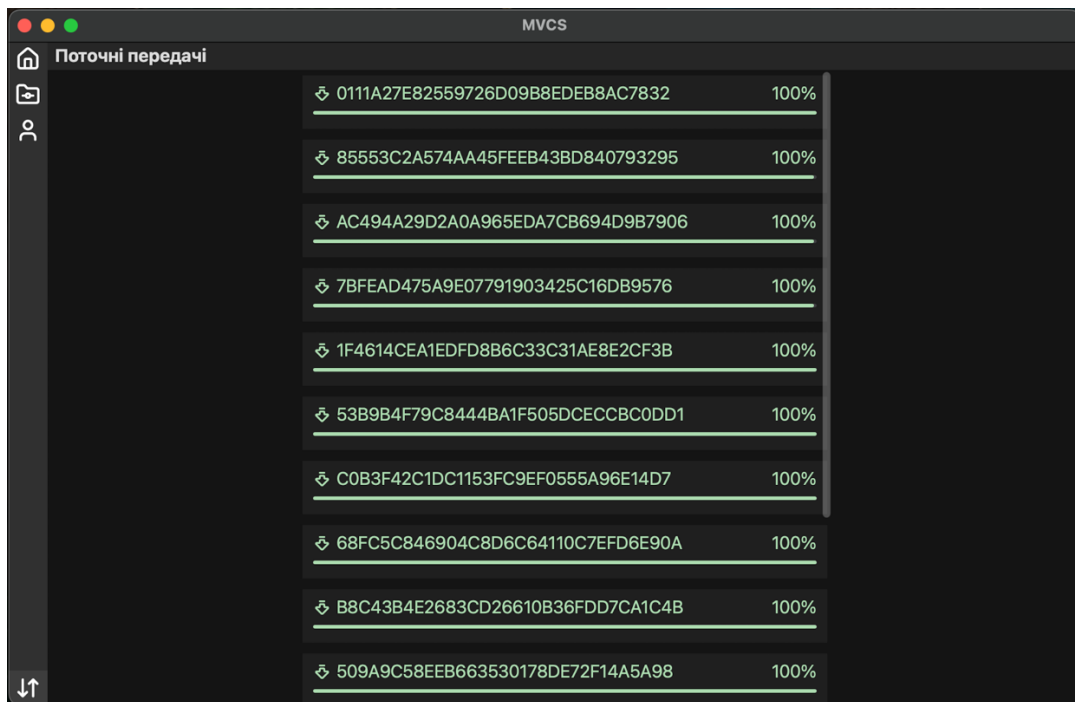


Рис. 3.7. Вкладка передачі файлів

Щоб операції пов'язані з сервером були доступні, або щоб в авторстві версій використовувалися дані з облікового запису користувача, необхідно здійснити вхід в обліковий запис. Це можна зробити з вкладки користувача, де вміст визначається автоматично по наявності і дійсності токенів (рис. 3.8).

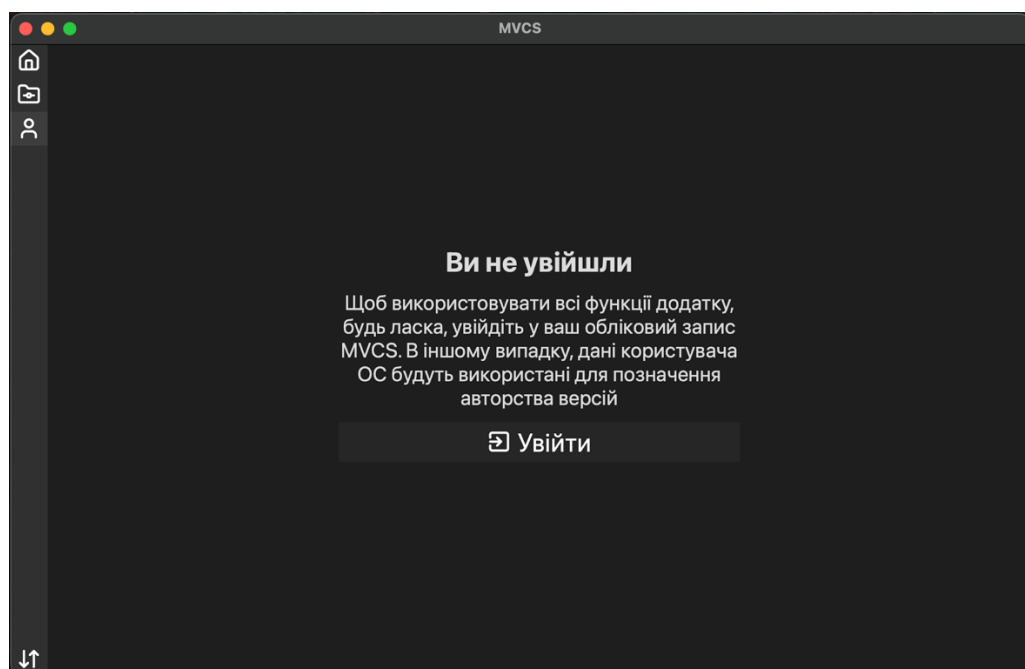


Рис. 3.8. Вкладка користувача

Якщо користувач ініціює вхід, генерується код для входу згідно стандарту Device Auth Flow. Відображуються інструкції, а також кнопка для переходу за посиланням в браузері з автоматичним введенням коду (рис. 3.9).

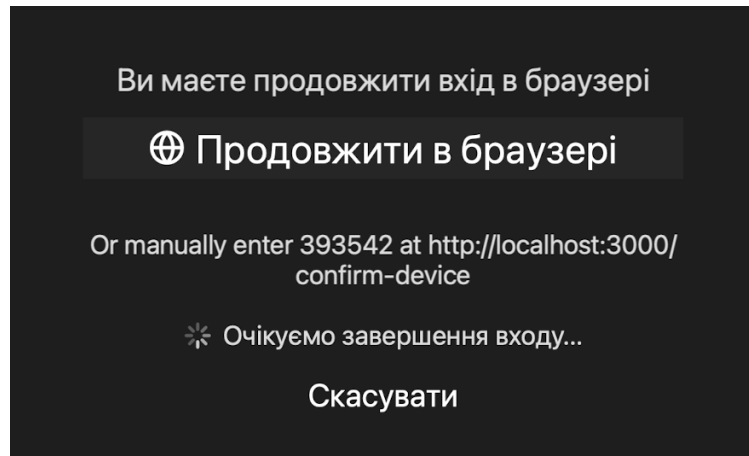


Рис. 3.9. Інструкції та дані для входу

Далі вхід продовжується на стороні вебклієнта. Користувач спершу має увійти в свій обліковий запис на вебклієнті за допомогою форми входу (рис. 3.10). Якщо користувач перейшов за посиланням підтвердження входу з віконного додатка, і при цьому не був автентифікований у вебклієнті, на форму входу його буде перенаправлено автоматично.

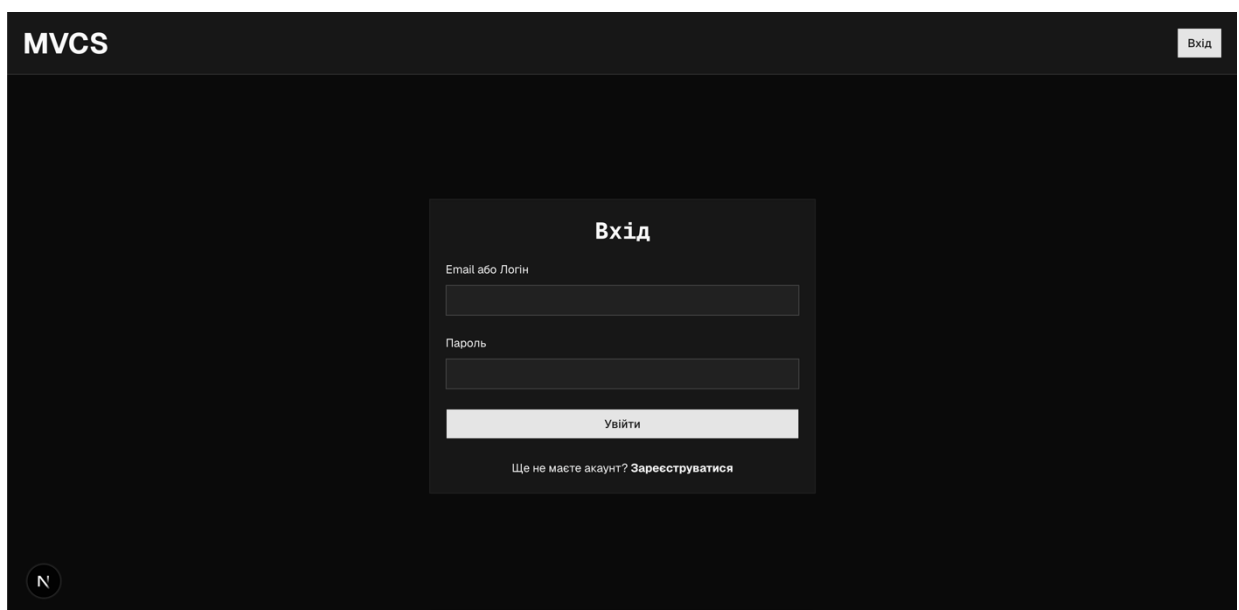


Рис. 3.10. Форма для входу на вебклієнті

		Регенель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				75
Змн.	Арк.	№ докум.	Підпис	Дата		

При цьому початкове посилання і код який воно містить буде збережено, а після успішного входу – використано для автоматичного перенаправлення назад. На сторінці підтвердження входу розміщено поле для введення коду з можливістю автозаповнення. Якщо код дійсний, після його введення буде відображено інформацію про пристрій, з якого ініційовано вхід (рис. 3.11).

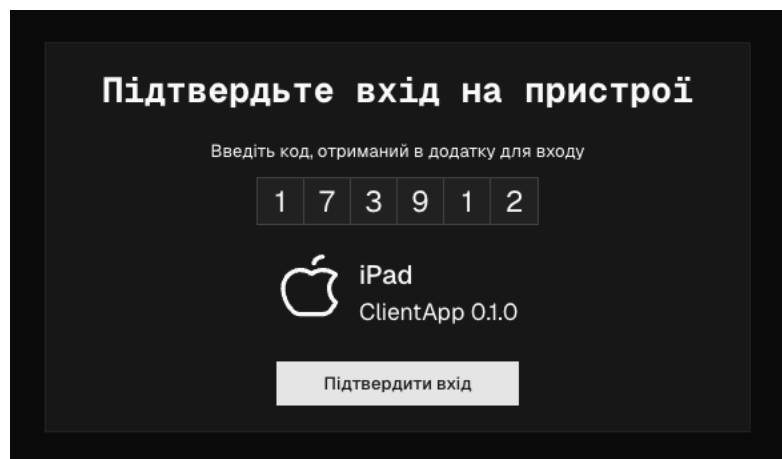


Рис. 3.11. Форма підтвердження входу на пристрої

Отримавши підтвердження входу, віконний додаток оновлює стан поточного користувача, і відтепер на вкладці користувача відображуються дані з його облікового запису та кнопка для виходу (рис. 3.12)

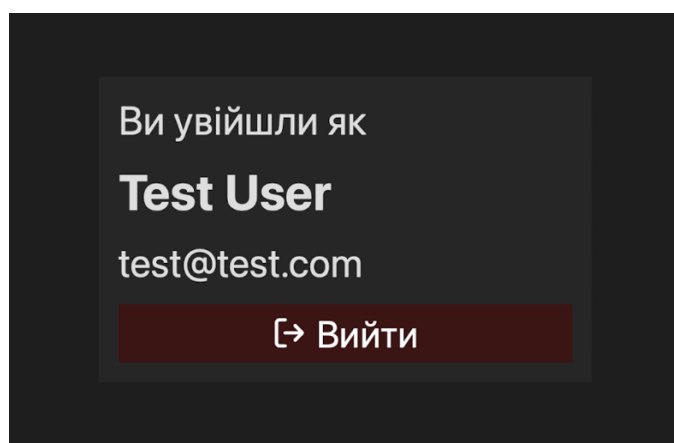


Рис. 3.12. Дані поточного користувача після успішного входу

		Регенель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				76
Змн.	Арк.	№ докум.	Підпис	Дата		

Решта функціоналу системи доступна через вебклієнт. В першу чергу користувачеві доступне керування сховищами. На сторінці сховищ відображується перелік сховищ, які надані платформою, які створив користувач, та до яких йому надано доступ (рис. 3.13).

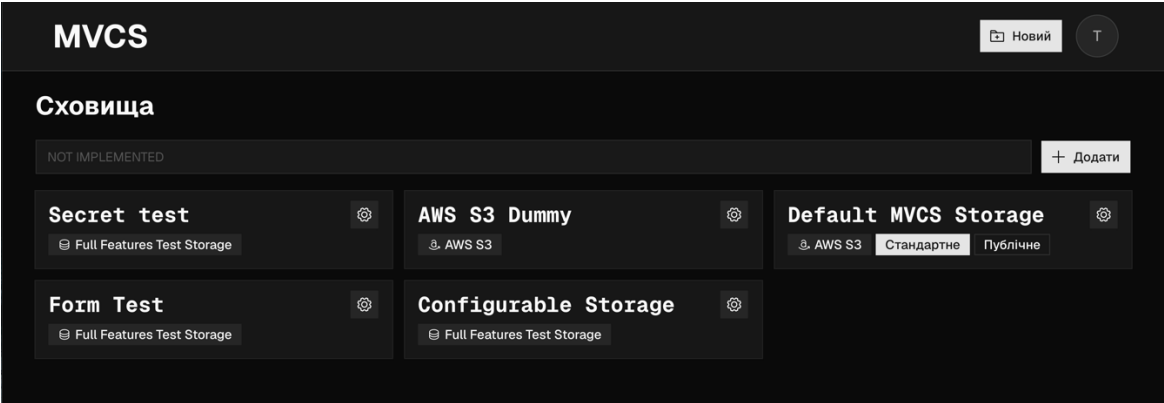


Рис. 3.13. Сторінка сховищ

З цієї сторінки користувач може створити нове сховище. У формі створення сховища спершу обирається його тип, після чого на основі схеми конфігурації цього типу сховища побудується решта форми (рис. 3.14).

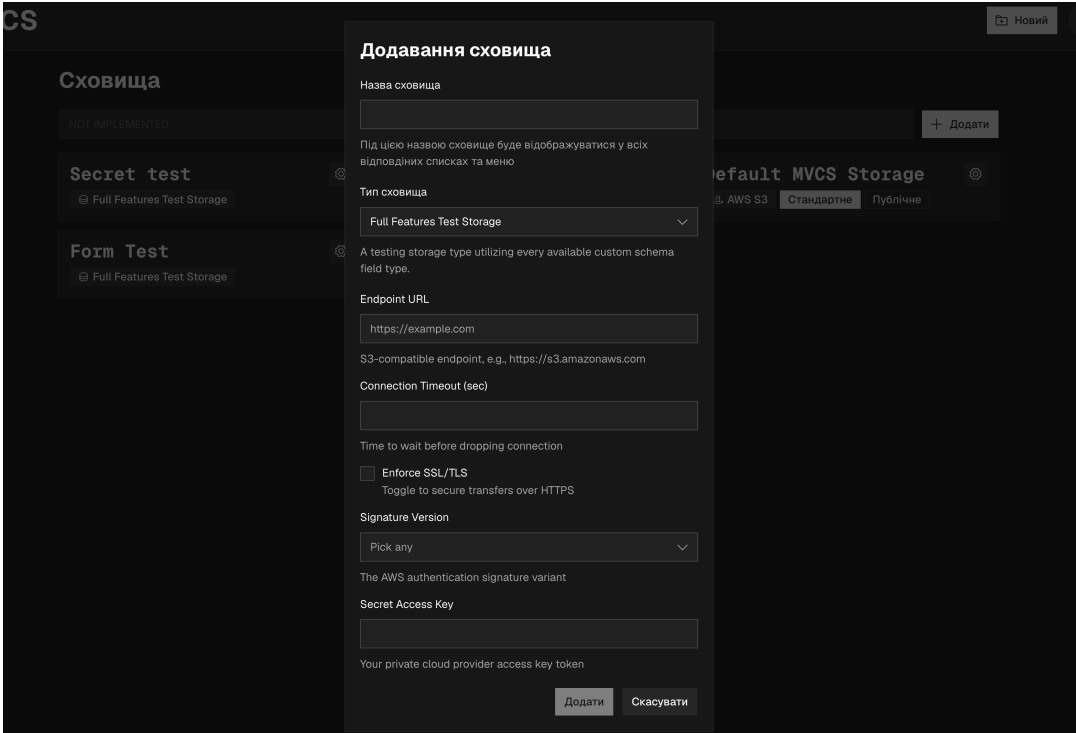


Рис. 3.14. Створення нового сховища

На сторінці конкретного сховища відображується його доступність на даний момент, а також, якщо користувач має відповідні права, налаштування сховища і керування правами доступу користувачів до нього (рис. 3.15)

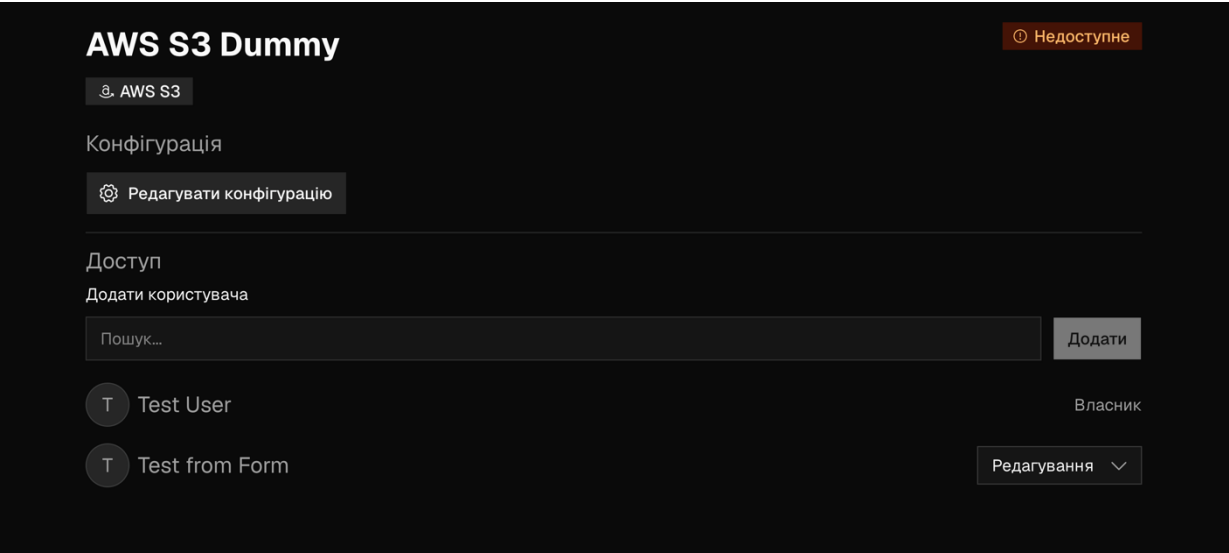


Рис. 3.15. Сторінка конкретного сховища

Користувач може переглядати перелік доступних йому проєктів на сторінці профілю (рис. 3.16). З цієї ж сторінки йому доступне керування власним обліковим записом.

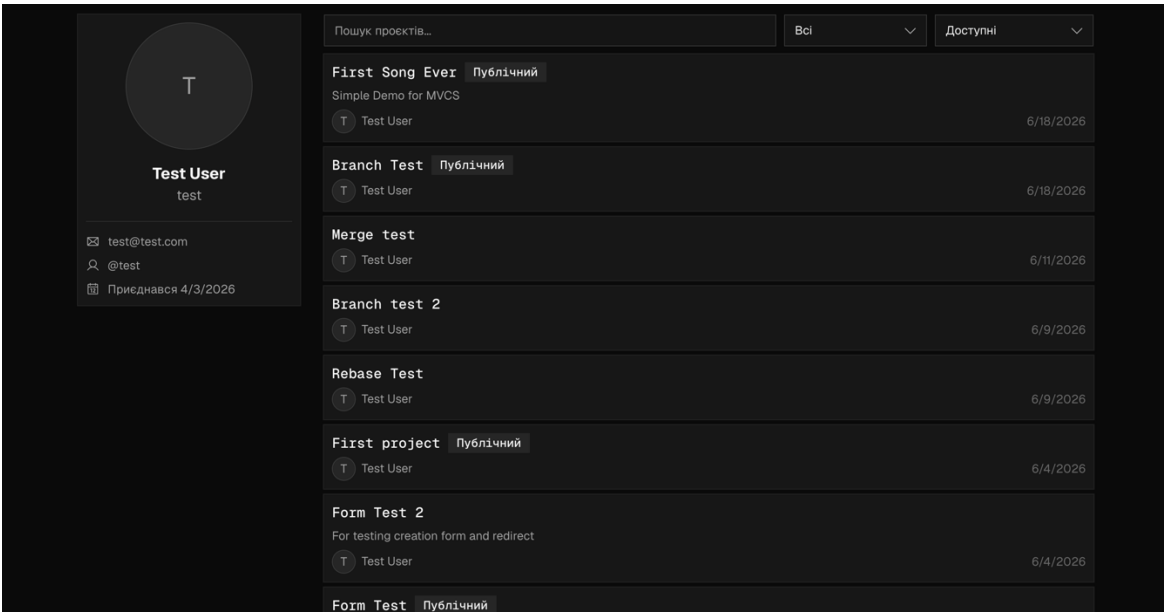


Рис. 3.16. Сторінка профілю користувача

З будь-якої сторінки у верхній частині інтерфейсу доступне створення нового проєкту. При натисканні на кнопку «Новий» користувачеві відображується форма, в якій необхідно ввести інформацію про проєкт, вказати його публічність та обрати сховище з переліку доступних (рис. 3.17).

Рис. 3.17. Створення нового сховища

На сторінці конкретного проєкту зверху відображуються його назва, опис та автор. Також відображується актуальна доступність сховища і статус публічності проєкту. Ідентифікатор для клонування або прив'язки локального проєкту копіюється при натисканні.

Нижче розміщено перемикач вкладок, серед яких доступні Файли, Історія, Злиття, Задачі та Налаштування. Деякі з вкладок можуть бути заблоковані за недоступності функції через стан проєкту або права користувача.

На вкладці з файлами вміст залежить від того, чи ініційовано проєкт. Якщо проєкт не ініційовано, відображується інструкція для користувача і дублюється ідентифікатор проєкту для копіювання (рис. 3.18).

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				79
Змн.	Арк.	№ докум.	Підпис	Дата		

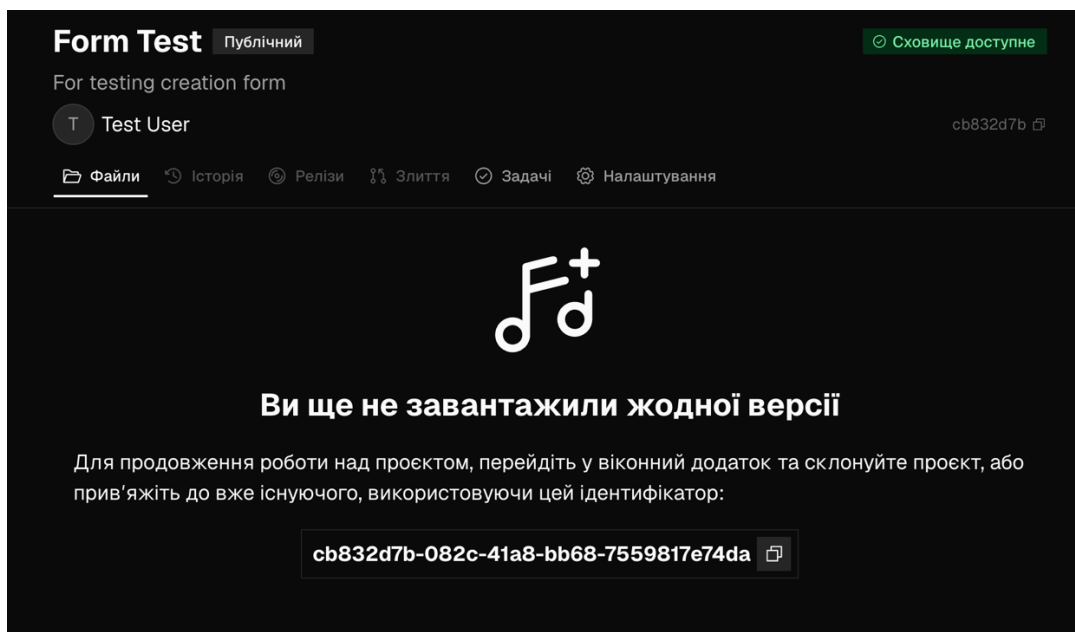


Рис. 3.18. Сторінка неініційованого проєкту

Якщо ж проєкт ініційовано, відображується вибір гілки, інформація про останню версію на ній, а також дерево файлів та директорій на момент цієї версії (рис. 3.19). При натисканні на директорію вона розкривається, а при натисканні на файл його буде завантажено через браузер. На цій вкладці також відображуються результати аналізу метаданих для цієї версії.

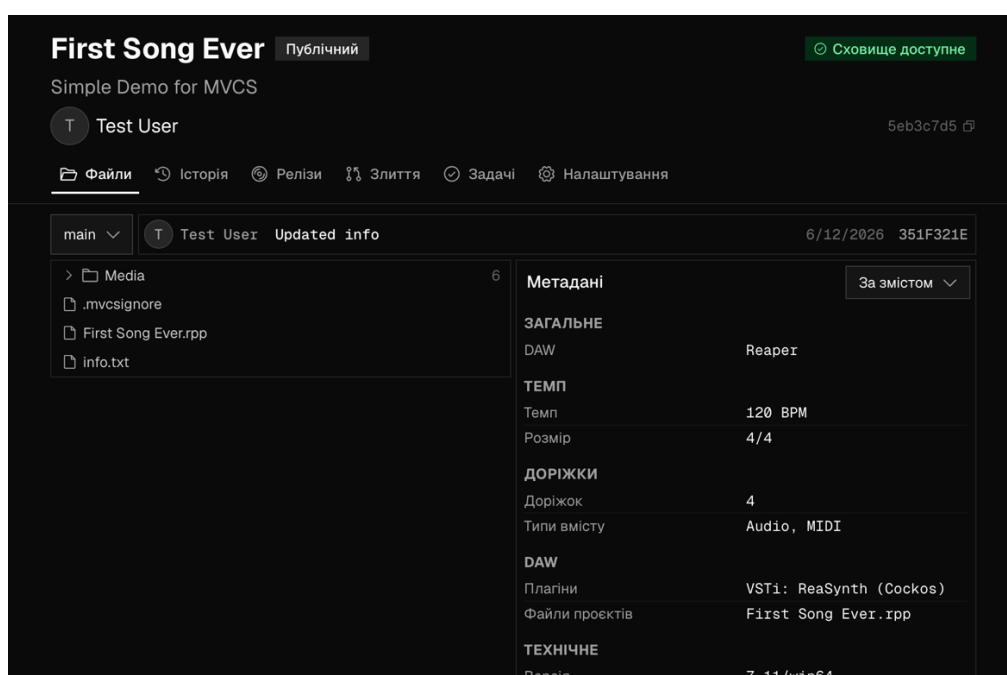


Рис. 3.19. Вкладка з вмістом проєкту

		Регенель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				80
Змн.	Арк.	№ докум.	Підпис	Дата		

На вкладці історії версій також доступний перемикач гілки, а нижче відображується перелік версій цієї гілки з групуванням по даті (рис. 3.20). Для кожної версії зазначається

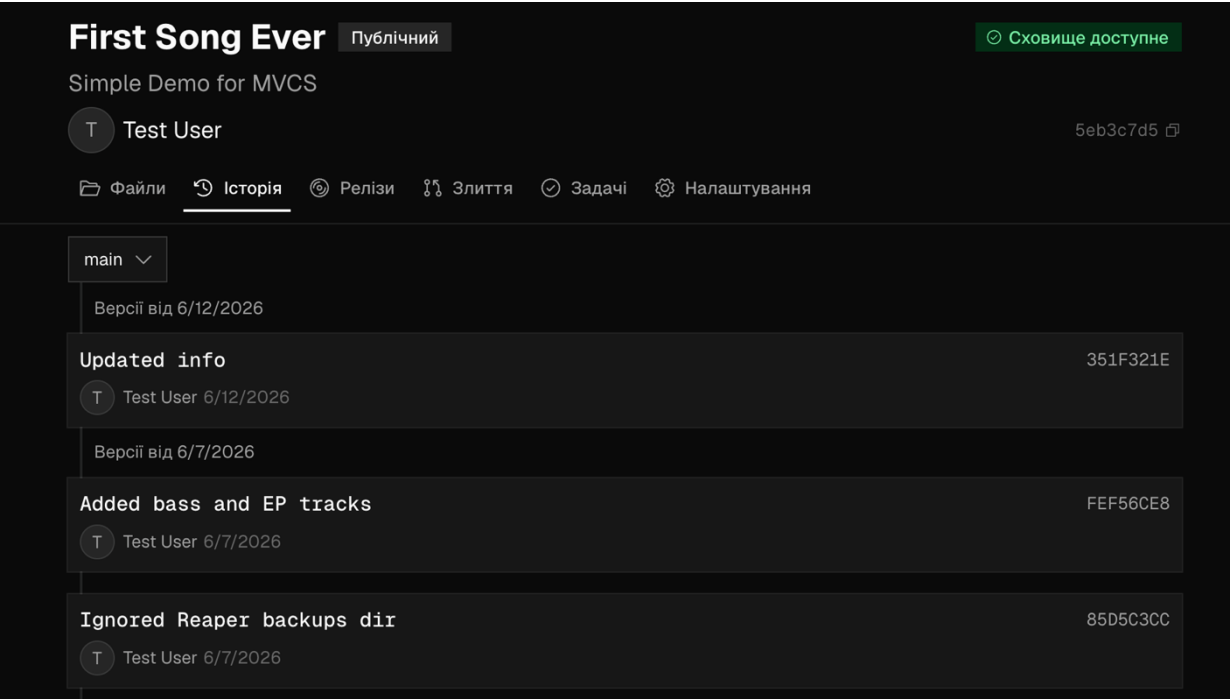


Рис. 3.20. Вкладка з історією версій

На вкладці злиття здійснюються відображуються запити на злиття. Для здійснення запити обираються дві гілки та вводиться повідомлення. Нижче в переліку відображуються попередні запити та інформація про них (рис. 3.21).

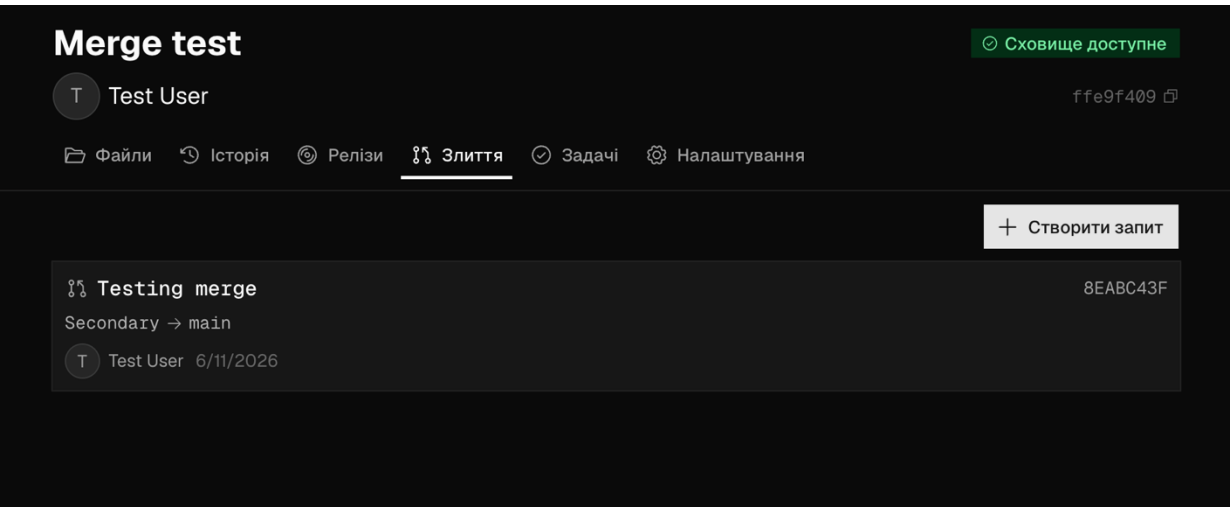


Рис. 3.21. Вкладка запитів на злиття

На вкладці релізів відображується історія релізів для проєкту, від нових до старих (рис. 3.22). Для кожного релізу відображується його назва, автор, дата та перелік файлів з можливістю прослуховування для аудіо.

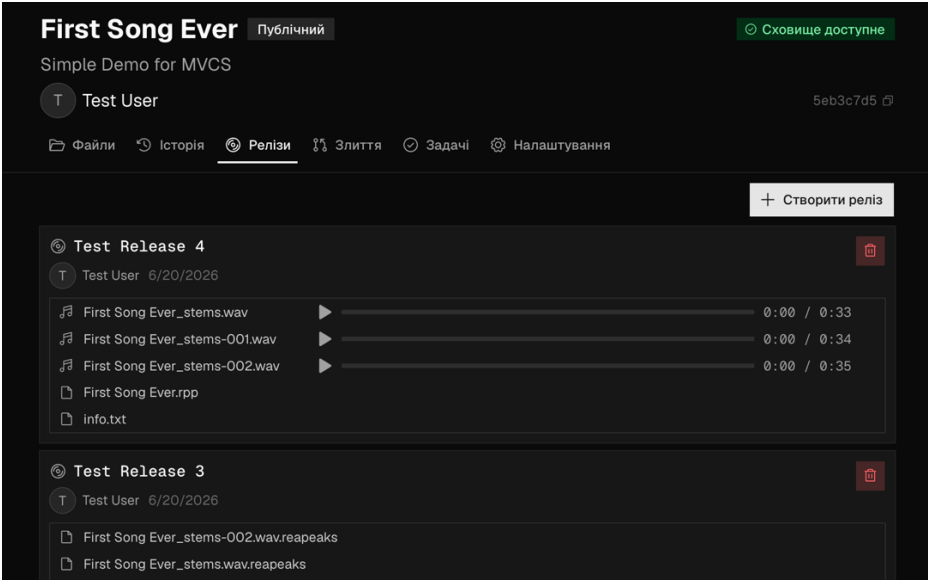


Рис. 3.22. Вкладка релізів

При створенні нового релізу необхідно вказати його назву, обрати гілку та вибрати файли, які потраплять у реліз (рис. 3.23). Створення релізів доступне користувачам з правом запису та власнику проєкту.

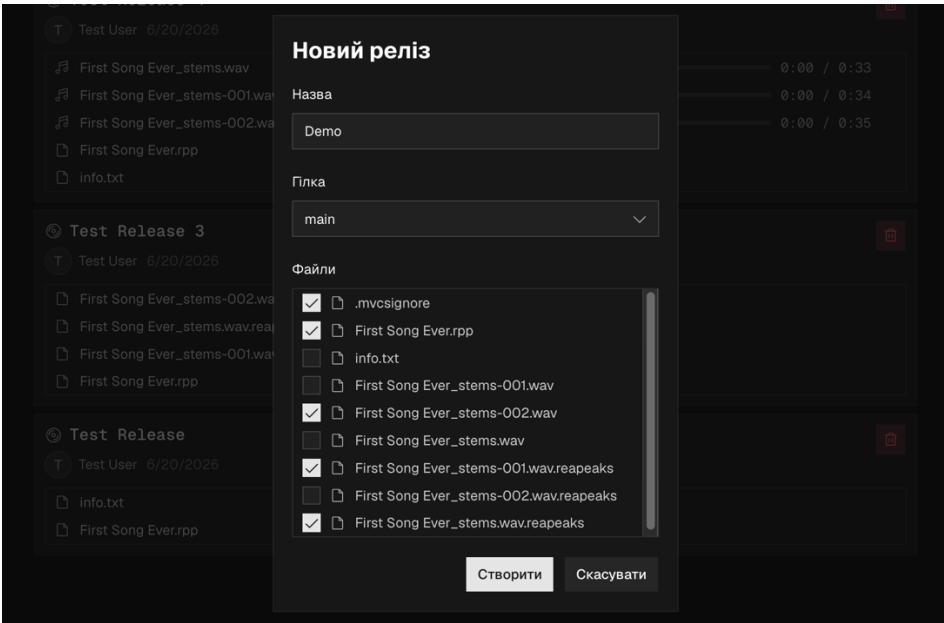


Рис. 3.23. Створення нового релізу

На вкладці задач відображується перелік задач та їх стани (рис. 3.24).
Всі задачі відсортовано в порядку останніх змін, від нещодавніх до старих. З переліку можна перейти до детальних відомостей про задачу.

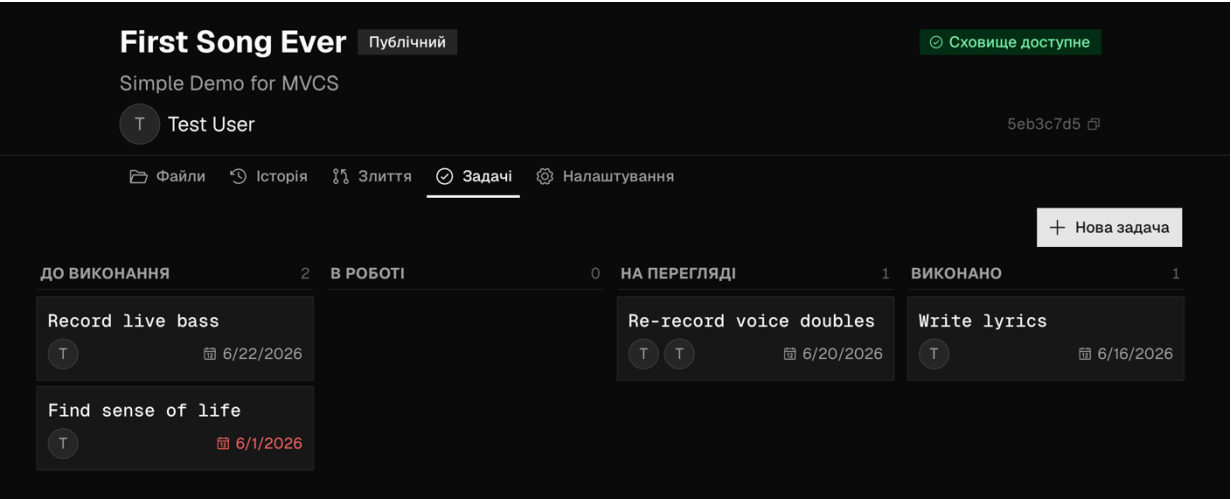


Рис. 3.24. Вкладка задач

В модальному вікні конкретної задачі відображуються її назва, опис, дедлайн, прив’язана версія та призначені користувачі (рис. 3.25). За наявності прав, користувачеві доступне редагування цих відомостей.

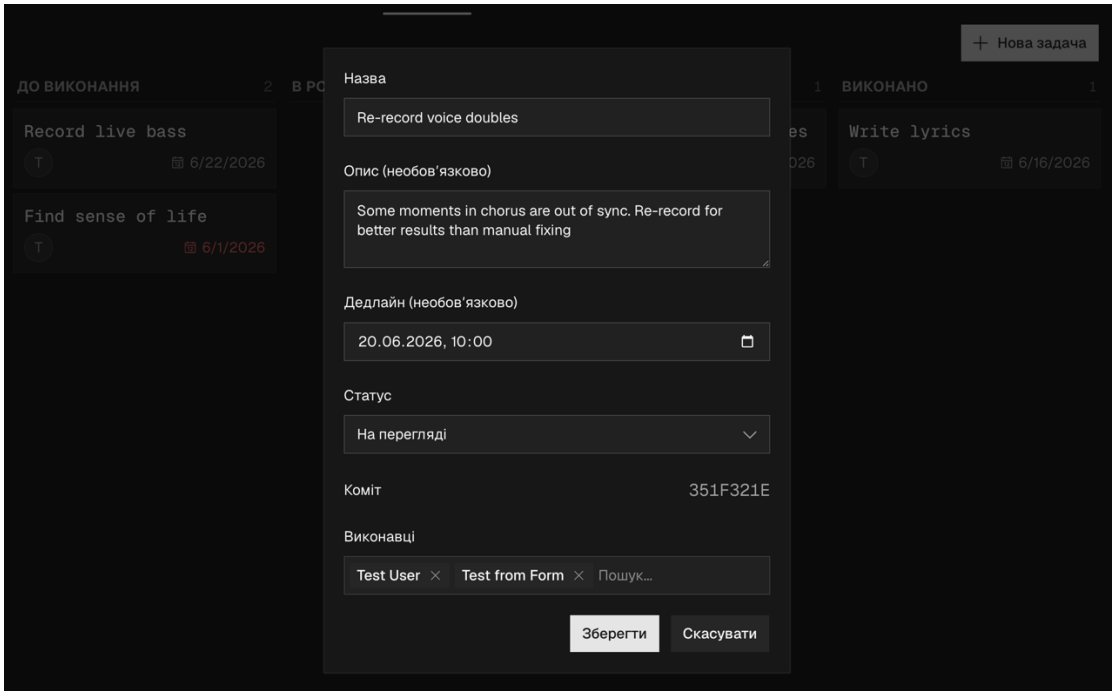


Рис. 3.25. Модальне вікно конкретної задачі

На вкладці налаштувань користувачеві, який має достатній рівень прав, доступне керування відомостями про проєкт, його публічністю, а також правами доступу інших користувачів до проєкту (рис. 3.26).

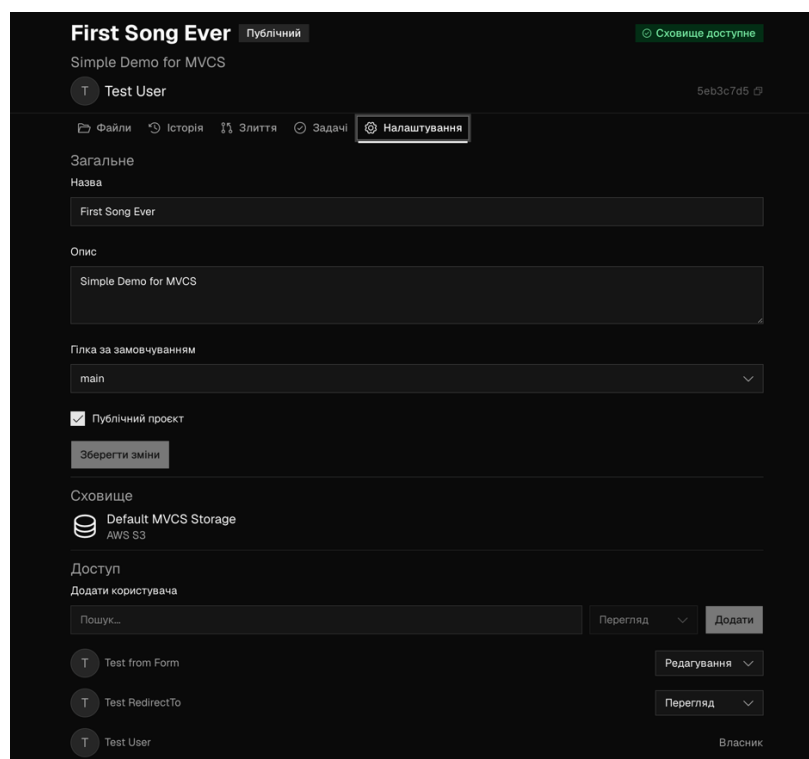


Рис. 3.26. Вкладка налаштувань проєкту

Пошук по публічних проєктах можна здійснити з головної сторінки (рис. 3.27). Пошук здійснюється по назві, опису проєкту, а список має «нескінченну прокрутку», тобто результати підвантажуються автоматично.

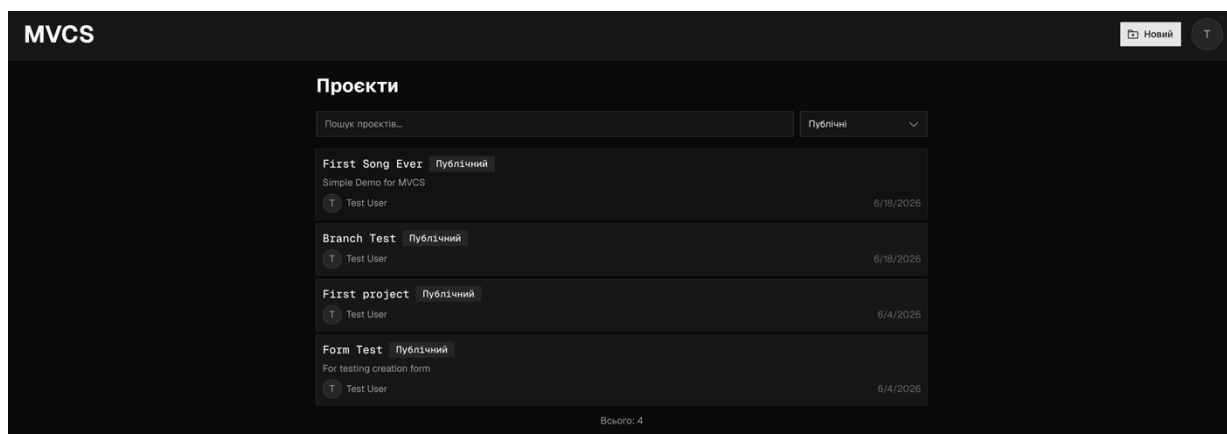


Рис. 3.27. Головна сторінка з пошуком публічних проєктів

		Регенель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				84
Змн.	Арк.	№ докум.	Підпис	Дата		

3.3. Тестування роботи програмного комплексу

Тестування програмного комплексу проводилось шляхом ручної перевірки основних сценаріїв роботи системи контролю версій та інтеграції додатків між собою. Для кожного сценарію визначались початкові умови, послідовність дій та очікуваний результат, після чого фіксувався фактичний результат виконання. Сценарії та результати наведено в таблиці 3.2.

Таблиця 3.2

Тестові сценарії функціоналу системи

Сценарій	Умови	Очікуваний результат	Фактичний результат
Push нових версій	Локальна історія випереджає серверну на одну або більше версій	Нові версії збережено на сервері, вказівники гілок оновлено, метадані синхронізовано	Відповідає очікуваному
Push із застарілими даними	Локальна історія базується на застарілому стані серверної	Сервер повертає помилку конфлікту, відображає повідомлення про потребу здійснити Pull	Відповідає очікуваному
Pull оновлень	Серверна історія випереджає локальну, конфліктів немає	Нові версії збережено локально, вказівники гілок оновлено, відсутні блоби завантажено	Відповідає очікуваному
Pull з конфліктом	Локальна і серверна історії мають розбіжність	Клієнт відображує меню з опціями вирішення конфлікту	Відповідає очікуваному
Вирішення конфлікту перезаписом	Локальна і серверна історії мають розбіжність	Локальна історія замінена серверною, локальні версії відкинуто	Відповідає очікуваному
Вирішення конфлікту перебазуванням	Локальна і серверна історії мають розбіжність	Локальні версії перебудовано поверх серверних після згенерованої версії-відкату.	Відповідає очікуваному

Clone проєкту	Проєкт існує на сервері, користувач має право на читання	Локальна копія проєкту створена, всі гілки і блоби завантажено, отримано історію версій для гілки за замовчуванням	Відповідає очікуваному
Відновлення перерваного завантаження блобів	Передача файлу перервана	Після відновлення з'єднання передача продовжується з місця зупинки без повторної передачі вже завантажених даних	Відповідає очікуваному
Синхронізація метаданих	Push виконано успішно, метадані сформовано локально	Метадані надіслані на сервер у фоновому режимі, поле SyncedAt оновлено	Відповідає очікуваному
Push без попереднього завантаження блобів	Локальна гілка містить нові версії, частина блобів не завантажена на сервер	Сервер відхиляє запит з помилкою про відсутність блобів	Відповідає очікуваному

За результатами тестування всі перевірені сценарії виконались відповідно до очікувань. Алгоритми синхронізації даних між клієнтом і сервером коректно обробляють як стандартні сценарії, так і помилкові випадки – конфлікти версій, відсутні блоби та переривання передачі файлів. Механізм відновлюваного завантаження на основі протоколу TUS підтвердив свою ефективність при перериванні з'єднання. Система контролю версій в цілому функціонує відповідно до визначених вимог.

Детальніше розглянемо сценарій вирішення конфлікту перебазуванням, який є одним з найскладніших алгоритмів системи. Після виявлення розбіжності між локальною і серверною історією гілки користувачу пропонується обрати спосіб вирішення конфлікту. При виборі

		Регенель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				86
Змн.	Арк.	№ докум.	Підпис	Дата		

перебазування RebaseCommand послідовно перебудовує кожну локальну версію поверх отриманих від сервера, копіюючи зміни файлів і повідомлення, але змінюючи ідентифікатори комітів. Результат виконання сценарію наведено на рисунку 3.28, де в панелі історії відображується перебудований ланцюг версій зі змінами з серверу та версією-відкатом.

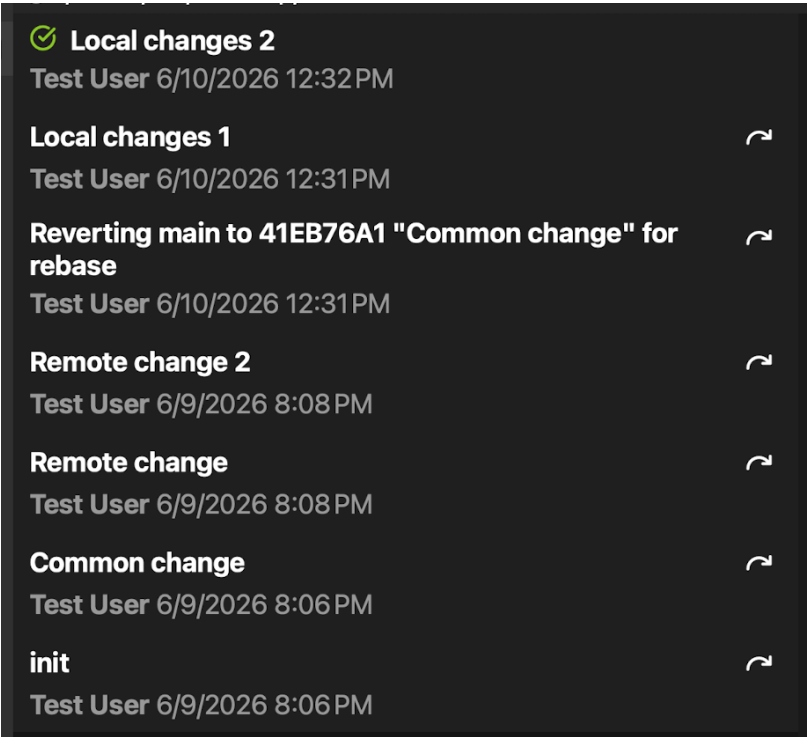


Рис. 3.28. Результати виконання перебазування

Окремо перевірено поведінку системи при недоступності сервера або сховища проєкту. RemoteService перевіряє доступність серверного додатка і сховища через відповідні health-ендпоінти. У разі негативного результату перевірки операції з сервером блокуються, а у верхній частині інтерфейсу відображується повідомлення про проблему (рис. 3.29). Це дозволяє користувачеві одразу зрозуміти причину неможливості синхронізації.

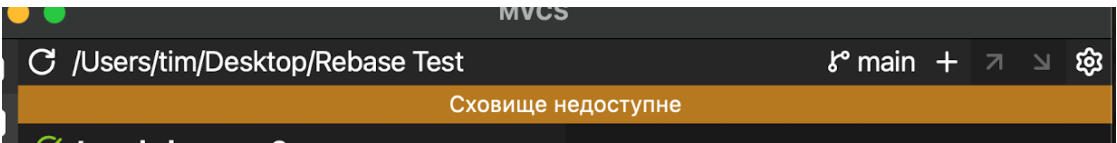


Рис. 3.29. Повідомлення про недоступність сховища

Висновки до третього розділу

У третьому розділі описано порядок розгортання та налаштування системи, інтерфейс віконного додатка і вебклієнта, а також результати тестування основних сценаріїв роботи.

Розгортання серверної частини здійснюється за допомогою Docker Compose і включає сім контейнерів, запуск яких автоматично координується через умови залежностей та перевірки здоров'я. Ініціалізація бази даних відбувається за допомогою скрипта `init.sh`, який створює бази даних і користувача додатку, після чого Flyway застосовує SQL-міграції окремо для AppDb і VcsDb [25]. Для автоматизації розгортання налаштовано CI/CD pipeline на основі GitHub Actions, який при злитті змін в гілку `main` автоматично розгортає оновлення на сервері [17]. Вебклієнт можна розгорнути разом із серверним додатком через застосування `docker-compose.frontend.yml` з мінімальним Docker-образом зібраним через багатетапний `Dockerfile`.

Інтерфейс віконного додатку організовано навколо вкладки активного проєкту з панеллю історії версій та панеллю збереження нової версії. Автентифікація реалізована через вебклієнт з підтвердженням входу на пристрої, що дозволяє уникнути введення облікових даних безпосередньо у віконному додатку.

Тестування охопило десять основних сценаріїв роботи системи контролю версій, серед яких Push, Pull, Clone, Rebase та вирішення конфліктів, а також чотири сценарії валідації та обробки помилок. Всі перевірені сценарії виконались відповідно до очікувань. Система коректно обробляє одночасні запити завдяки блокуванню на рівні транзакцій, відновлює перервані завантаження файлів без повторної передачі вже завантажених даних та своєчасно інформує користувача про проблеми з підключенням через повідомлення в інтерфейсі.

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				88
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи розроблено програмний комплекс системи контролю версій для музичних проєктів, що складається з серверного додатка на базі ASP.NET Core, віконного додатка на базі .NET MAUI та вебклієнта на базі Next.js.

У першому розділі проведено аналіз предметної області та обґрунтовано доцільність розробки. Встановлено що наявні рішення загального призначення погано пристосовані до роботи з музичними проєктами, а єдиний прямий аналог в цій сфері перебуває на стадії розробки. Визначено архітектуру системи на основі спільної бібліотеки з функціоналом контролю версій, яка використовується як серверним, так і віконним додатком. Обґрунтовано вибір технологічного стеку та протоколів передачі даних з урахуванням вимог до роботи з файлами великого розміру.

У другому розділі спроектовано та реалізовано компоненти системи. Розроблено схему двох баз даних з обґрунтуванням їх розділення на AppDb для сутностей вебплатформи та VcsDb для даних системи контролю версій. Детально описано алгоритми основних операцій, зокрема завантаження і отримання вмісту файлів через протокол TUS та Range-запити, Push і Pull як фундаментальні операції синхронізації між клієнтом і сервером, та Rebase для вирішення конфліктів версій зі збереженням локальних змін. Описано реалізацію підсистеми роботи зі сховищами на основі патерну «Адаптер», яка дозволяє підключати різні типи сховищ без змін у решті системи, а також систему аналізаторів метаданих вмісту проєкту з підтримкою DAW Reaper.

У третьому розділі описано порядок розгортання системи засобами Docker Compose з автоматизованою ініціалізацією баз даних та CI/CD pipeline на основі GitHub Actions. Описано інтерфейс віконного додатку та вебклієнта. За результатами ручного тестування чотирнадцяти сценаріїв підтверджено коректність роботи основних алгоритмів системи, включаючи обробку одночасних запитів, відновлення перерваних завантажень та інформування користувача про проблеми з доступністю сховища.

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				89
Змн.	Арк.	№ докум.	Підпис	Дата		

Розроблений програмний комплекс реалізує повний набір функцій системи контролю версій адаптованих до специфіки роботи з музичними проєктами. Підтримка користувацьких сховищ через підсистему адаптерів дозволяє зберігати вміст проєктів на власній інфраструктурі користувача без передачі облікових даних за межі серверного додатку. Спільна робота багатьох користувачів забезпечується через механізм блокування на рівні транзакцій бази даних, що унеможлиблює пошкодження даних при одночасних операціях запису. Аналіз та синхронізація метаданих вмісту проєктів реалізовані через розширювану систему аналізаторів, яка наразі підтримує формат Reaper і може бути доповнена підтримкою інших DAW без змін у решті системи. Ефективна передача файлів великого розміру досягається завдяки протоколу TUS з підтримкою відновлення перерваних завантажень та збереженням стану передачі в Redis.

Подальший розвиток системи може передбачати розширення переліку підтримуваних типів сховищ шляхом реалізації нових адаптерів – зокрема Google Drive, Dropbox та власний додаток для перетворення довільного пристрою на сховище. Перспективним напрямком також є додавання підтримки інших DAW у підсистемі аналізаторів метаданих, що розширить аудиторію потенційних користувачів системи. Також цінними нововведеннями буде End-to-end шифрування вмісту файлів, резервне копіювання проєкту на другорядне сховище та розширена система прав доступу до проєкту. Окремим напрямком розвитку є інтеграція з існуючим ПЗ, зокрема з системами управління на кшталт Trello або Monday та інтеграція безпосередньо з інтерфейсом DAW для покращеного досвіду користувача. Додатково розглядається виокремлення публічного API для того, щоб інші розробники могли інтегрувати свої продукти з розробленою системою, розробляти власні клієнтські додатки тощо.

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				90
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Берко А. Ю., Верес О. М., Пасічник В. В. Системи баз даних та знань. Кн. 2: Системи управління базами даних та знань. Львів: Магнолія-2006, 2022. 584 с.
2. Бублик В. В. Об'єктно-орієнтоване програмування. 606 с.
3. Пірог О. А. Безпека вебдодатків. 203 с.
4. Amazon S3 API Reference. AWS Documentation. 2026. URL: <https://docs.aws.amazon.com/AmazonS3/latest/API/Welcome.html> (дата звернення: 10.02.2026).
5. ASP.NET Core Documentation. Microsoft. 2026. URL: <https://learn.microsoft.com/en-us/aspnet/core> (дата звернення: 01.03.2026).
6. Carlson J. L. Redis in Action. Manning Publications, 2013. 320 p.
7. Chacon S., Straub B. Pro Git. 2nd ed. Apress, 2014. 456 p.
8. Cwalina K., Abrams B. Framework Design Guidelines. Conventions, Idioms, and Patterns for Reusable .NET Libraries. Addison-Wesley, 2008. 578 p.
9. DAWLab – Version Control for Music Production. URL: <https://dawlab.online/> (дата звернення: 30.01.2026).
10. De Sanctis V. Building Web APIs with ASP.NET Core. Manning Publications, 2023. 392 p.
11. Durner D., Leis V., Neumann T. Exploiting Cloud Object Storage for High-Performance Analytics. Proceedings of the VLDB Endowment. 2023. Vol. 16, no. 11. P. 2769–2782. URL: <https://www.vldb.org/pvldb/vol16/p2769-durner.pdf> (дата звернення: 15.03.2026).
12. Flyway Documentation. Red Gate. 2026. URL: <https://documentation.red-gate.com/fd> (дата звернення: 05.03.2026).
13. Fuksa M., Speth S., Becker S. MVVM Revisited: Exploring Design Variants of the Model-View-ViewModel Pattern. 28th International Conference on Enterprise Design, Operations, and Computing. 2024.
14. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994. 395 p.

		Регенель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				91
Змн.	Арк.	№ докум.	Підпис	Дата		

15. Git Large File Storage. URL: <https://git-lfs.com/> (дата звернення: 27.01.2026).
16. GitHub Actions Documentation. GitHub Docs. 2026. URL: <https://docs.github.com/en/actions> (дата звернення: 20.03.2026).
17. Humble J., Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley, 2010. 512 p.
18. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT): RFC 7519. Internet Engineering Task Force, 2015. URL: <https://www.rfc-editor.org/rfc/rfc7519> (дата звернення: 15.02.2026).
19. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p.
20. Momjian B. PostgreSQL: Introduction and Concepts. URL: https://momjian.us/main/writings/pgsql/aw_pgsql_book/ (дата звернення: 01.03.2026).
21. Nath P., Urgaonkar B., Sivasubramaniam A. Evaluating the usefulness of content addressable storage for high-performance data intensive applications. Proceedings of the 17th International Symposium on High Performance Distributed Computing. ACM, 2008. P. 35–44.
22. Next.js Documentation. Vercel. 2026. URL: <https://nextjs.org/docs> (дата звернення: 15.03.2026).
23. Nginx Documentation. 2026. URL: <https://nginx.org/en/docs/> (дата звернення: 20.03.2026).
24. .NET MAUI Documentation. Microsoft. 2026. URL: <https://learn.microsoft.com/en-us/dotnet/maui> (дата звернення: 01.03.2026).
25. Obe R., Hsu L. PostgreSQL: Up and Running. 4th ed. O'Reilly Media, 2026. 400 p.
26. Richardson L., Amundsen M., Ruby S. RESTful Web APIs: Services for a Changing World. O'Reilly Media, 2013. 406 p.

		Регель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				92
Змн.	Арк.	№ докум.	Підпис	Дата		

27. Smith J. P. Entity Framework Core in Action. 2nd ed. Manning Publications, 2021. 600 p.

28. Su R., Li X. Modular Monolith: Is This the Trend in Software Architecture? SATrends '24: Proceedings of the 1st International Workshop on New Trends in Software Architecture. 2024. P. 10–13.

29. Turnbull J. The Docker Book: Containerization is the New Virtualization. James Turnbull, 2018. 349 p.

30. tus Resumable Upload Protocol 1.0. URL: <https://tus.io/protocols/resumable-upload> (дата звернення: 10.02.2026).

		Регенель Т.Ю.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Фант М.О.				93
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

Конфігурація GitHub Actions для розгортання серверного додатку

name: Deploy

on:

push:

branches: [main]

jobs:

deploy:

runs-on: ubuntu-latest

steps:

- name: Checkout repository
uses: actions/checkout@v4

- name: Deploy to server
uses: appleboy/ssh-action@v1
with:
 host: \${{ secrets.SERVER_HOST }}
 username: \${{ secrets.SERVER_USER }}
 key: \${{ secrets.SERVER_SSH_KEY }}
 script: |
 set -e
 cd /opt/mvcs
 PREVIOUS_COMMIT=\$(git rev-parse HEAD)
 echo \$PREVIOUS_COMMIT > /tmp/mvcs_previous_commit
 git pull origin main
 docker compose -f docker-compose.yml up -d --build

- name: Verify deployment
uses: appleboy/ssh-action@v1
with:
 host: \${{ secrets.SERVER_HOST }}
 username: \${{ secrets.SERVER_USER }}
 key: \${{ secrets.SERVER_SSH_KEY }}
 script: |
 set -e
 cd /opt/mvcs
 echo "Waiting for containers to start..."

Продовження додатка А

```
sleep 30
CONTAINERS=("nginx" "api" "postgres" "redis")
FAILED=0

for CONTAINER in "${CONTAINERS[@]"; do
    STATUS=$(docker inspect --format='{{.State.Status}}' $CONTAINER
2>/dev/null || echo "missing")
    HEALTH=$(docker inspect --format='{{if
.State.Health}}{{.State.Health.Status}}{{else}}none{{end}}' $CONTAINER
2>/dev/null || echo "missing")

    if [ "$STATUS" != "running" ]; then
        echo "FAIL: $CONTAINER is $STATUS"
        FAILED=1
    elif [ "$HEALTH" != "none" ] && [ "$HEALTH" != "healthy" ]; then
        echo "FAIL: $CONTAINER health is $HEALTH"
        FAILED=1
    else
        echo "OK: $CONTAINER ($STATUS, health: $HEALTH)"
    fi
done

if [ "$FAILED" -eq 1 ]; then
    echo "Deployment verification failed. Rolling back..."

    PREVIOUS_COMMIT=$(cat /tmp/mvcs_previous_commit)
    git checkout $PREVIOUS_COMMIT
    docker compose -f docker-compose.yml up -d --build

    echo "Rollback to $PREVIOUS_COMMIT complete."
    exit 1
fi

echo "All containers are healthy. Deployment successful."
```