УДК 004.42:78.06
DOI https://doi.org/10.32782/2663-5941/2025.3.2/43

Регенель Т.Ю.
Державний університет «Житомирська політехніка»

Фант М.О.
Державний університет «Житомирська політехніка»

Савіцький Р.С.
Державний університет «Житомирська політехніка»

РОЗРОБКА СИСТЕМИ КОНТРОЛЮ ВЕРСІЙ ДЛЯ МУЗИЧНИХ ПРОЄКТІВ

У статті розглянуто розробку системи контролю версій для працівників музичної індустрії. В розробку входить сама система контролю версій, а також екосистема програмних продуктів на її основі, зокрема, застосунку для роботи з файлами проєктів безпосередньо на комп'ютері користувача, а також веб-платформи для зберігання, керування та колективної роботи над проєктами. Не зважаючи на наявність великої кількості різноманітного програмного забезпечення, яке покриває практично всі основні потреби різних робочих процесів у музичній індустрії, все ще відсутні інструменти для організації та контролю файлів різних версій проєкту. Існуючі аналоги на кшталт Git та GitHub не набувають широкого використання в цій сфері з низки причин, серед яких погана оптимізація системи щодо роботи з великими файлами (понад 50мб, в той час як середній обсяг файлів однієї сесії запису в музикантів може сягати декількох гігабайт), вимоги до навичок користувача у роботі з командним рядком, засобами автентифікації та ін., а також орієнтованість функціоналу саме на роботу з кодом та специфіку робочого процесу програмістів, тестувальників, DevOps-інженерів тощо, яка має ряд відмінностей від робочого процесу в музичній індустрії. Більш розповсюдженими у використанні є файлообмінники, такі як Google Drive або DropBox, і деякі з них мають вбудоване версіювання файлів. Проте, це версіювання дуже обмежене по своєму функціоналу, а також відсутня можливість працювати декільком учасникам паралельно (історія версій лінійна та не підтримує розгалужень. В результаті дослідження, спроєктовано саму систему контролю версій, на базі якої продукт, що складається з клієнтського додатку та веб-платформи з підтримкою підключення користувацького сховища, які покривають всі ці недоліки, і водночас пропонують не гіршу функціональність, значно підвищуючи ефективність та якість робочого процесу працівників сфери.

Ключові слова: *система контролю версій, версіювання, збереження файлів, веб-платформа, програмне забезпечення.*

Постановка проблеми. На сьогоднішній день робота з музикою складається з різних етапів, на кожному з яких може бути залучено багато ролей одночасно, окрім самого виконавця: продюсери, мікс-інженери, саунд-дизайнери, запрошені виконавці, сесійні музиканти тощо. Організація робочих матеріалів та різних версій повністю покладається на версіювання файлообмінників, або домовленості у найменуваннях та зберігання декількох копій проєкту в сховищі. Відсутні інструменти та сервіси, які надають функціонал для організації та синхронізації матеріалів між різними учасниками процесу протягом усієї роботи над проєктом. Існують аналогічні системи для представників інших сфер, наприклад, система Git та платформа GitHub. Проте ці програмні продукти вимагають певного рівня знань і навичок в сфері інформаційних технологій, що суттєво ускладнює їх використання представниками інших сфер.

Аналіз останніх досліджень і публікацій. В дослідженні Зуєвського Д. Р. та Кулішової Н. Є. [1] описано розробку системи контролю версій інтегрованої в основне ПЗ, зі збереженням змін об'єктів всередині самих об'єктів. Такий підхід має низку переваг, проте у вказаному дослідженні розглядалася робота з електронними виданнями. В той час як розповсюджені програмні продукти в цьому напрямі підтримують аддони та надають програмний інтерфейс для взаємодії з об'єктами, програмне забезпечення в музичній індустрії, зокрема, такий вид ПЗ як

Digital Audio Workstation або DAW, не надають достатньо гнучкі можливості маніпуляції елементами проєкту, що робить цей підхід не найкращою опцією в рамках нашого дослідження, і клієнтську частину буде оформлено в окремий незалежний додаток, а не в плагін/інтеграцію.

В дослідженні Александер Й. та ін. [2] розглянуто інший аспект систем контролю версій – синхронізацію даних на клієнтах. Зокрема, розглянуто розподілені системи контролю версій. Цей підхід має свої переваги. Наприклад, можливість побудувати систему без серверу взагалі, або із значно дешевшим у розробці та розгортанні серверним додатком. Проте, в роботі з музичними проєктами є певна специфіка: часто доведеться працювати з великими файлами, понад 50мб, в тому числі передачу цих даних по мережі. Також, нерідким явищем може бути співпраця фахівців з різних часових поясів, що може суттєво ускладнити синхронізацію даних на їх комп'ютерах (потрібен момент, у який обидва комп'ютери будуть доступні у мережі). Тому в межах нашого дослідження ми будемо використовувати клієнт-серверну архітектуру для синхронізації між клієнтами та збереження даних.

Постановка завдання. Необхідно розробити систему контролю версій, і на основі цієї системи розробити клієнтський додаток для безпосередньої взаємодії з файлами проєктів, а також веб-платформу для зберігання даних проєктів на віддаленому сервері, синхронізації даних між декількома клієнтами тощо. При розробці врахувати специфіку сфери застосування, включаючи тип і структуру файлів, вимоги щодо доступності і зручності інтерфейсу та ін.

Виклад основного матеріалу. *Розробка системи контролю версій.* Кінцеве запропоноване рішення складається з щонайменше двох програмних продуктів: клієнтський додаток і веб-платформа. Всі ці продукти працюватимуть з проєктами та їх елементами, тобто з функціоналом системи контролю версій (далі – СКВ). Щоб ефективніше організувати процес розробки, а також забезпечити відповідність функціоналу та імплементації СКВ між різними частинами рішення, СКВ буде реалізовано у вигляді npm-пакету, який встановлюватиметься та використовуватиметься у веб-платформі (NextJS), клієнтському додатку (Electron), та за потреби в інших продуктах, якщо буде потреба їх додати. При такому форматі роботи важливо, щоб пакет СКВ був типізованим, тому для його написання буде використано TypeScript. Оскільки надійність цього елемента розробки є вкрай критичною (адже інші елементи залежать від нього), пакет також буде покрито тестами за допомогою технології Jest. Це дозволить контролювати функціональність усієї СКВ після будь-яких внесених змін.

На відміну від програмістів, які працюють з кодом у файлах, тобто безпосередньо з вмістом файлу, музикантам для роботи потрібно декодувати цей вміст у відповідному ПЗ. А структура та кодування файлів проєктів цього ПЗ часто є закритим. До того ж, в залежності від обраного формату, зміна в конкретній ділянці аудіо файлу при конвертації може змінити значно більшу частину кінцевого вмісту файлу. Таким чином, розглядати конкретні байти вмісту файлу як атомарну сутність СКВ немає потреби. Замість цього, головною сутністю оберемо так званий Item – Елемент Проєкту. Кожен Item міститиме свій унікальний ідентифікатор, ідентифікатор контенту і шлях кінцевого файлу у проєкті. Контентом називаємо копію файлу з ідентичним вмістом, та ідентифікатором замість назви. Цей контент зберігатиметься в службовій директорії СКВ, і використовуватиметься для відтворення стану файлу на момент певної версії. Якщо якийсь файл зміниться, буде створено новий Item, з тим самим шляхом, що і попередній Item для цього файлу, але з іншим ідентифікатором контенту. Якщо файл буде перейменовано/переміщено – також створюватиметься новий Item, з тим самим ідентифікатором контенту, але з іншим шляхом. У такий спосіб, ми зберігатимемо мінімальну можливу кількість контенту, уникаючи зайвої дублікації даних, що зменшує обсяг диску необхідний для роботи, і трафіку відповідно.

Щоб репрезентувати зміну в проєкті, введемо сутність ItemChange – Зміну елемента. Кожен ItemChange може містити ідентифікатори Item-ів, які відповідать стану файлу до зміни та після зміни. Назвімо це полями from і to. Якщо існують обидва ідентифікатори – файл було змінено, або перейменовано/переміщено. Якщо існує тільки ідентифікатор to, а до змін у файлу не було ніякого стану – файл було вперше додано в проєкт. Якщо ж навпаки, до внесення змін існував певний стан файлу, а стану після змін немає (відсутній ідентифікатор to) – файл було видалено з проєкту.

Сутністю, яка відповідає певній версії проєкту, буде Commit. Commit включатиме в себе метадані (ідентифікатор, назва, опис, автор, дата

створення тощо), посилання на батьківський та дочірній коміти для роботи з історією комітів як з деревом, а також найголовніше – список змін, тобто список ItemChange-ів. Для гнучкої організації введемо сутність гілки, яка являє собою просто посилання на певний коміт.

А сутністю, яка відповідає проєкту музиканта, буде сам проєкт СКВ – Project. Project міститиме метадані (ідентифікатор, назва, опис, автор, дата створення тощо), поточний стан проєкту (поточний Commit, поточна гілка), список гілок, список Commit-ів та список усіх існуючих Item-ів. Саме цю сутність ми зберігатимемо у форматі JSON в службовому файлі та в БД.

Для колекцій сутностей використовуватимуться асоціативні масиви (Map). Збереження таких колекцій у вигляді пар ключ-значення дозволить зручно та ефективно отримувати сутність за її ідентифікатором.

Однією з найосновніших операцій в функціях СКВ буде порівняння станів проєкту. Порівнюватися можуть як стани проєкту з двох комітів, так і порівняння певного коміту з поточним станом файлів у директорії проєкту. Оскільки самі стани безпосередньо ніде не зберігаються, перед порівнянням їх потрібно буде сформувати.

Якщо потрібно отримати стан проєкту на момент певного коміту, це можна зробити в два етапи: визначення ланцюга комітів і накладання змін. Для визначення ланцюга коміті, нам потрібно обходити все дерево комітів від заданого коміта до його нащадка, поки ми не прийдемо до кореневого коміта проєкту, зберігаючи ідентифікатори всіх відвіданих комітів. В результаті отримаємо історію комітів для заданого коміту у вигляді переліку ідентифікаторів. Далі, для накладання змін, потрібно обійти отриманий перелік, формуючи стан проєкту у тимчасовій змінній. На кожній ітерації ми обходимо ItemChange-и коміту, і в залежності від типу змін (додавання, модифікація, видалення), відтворювати їх у тимчасовій змінній. В результаті обходу, отримаємо стан проєкту для цільового коміту у вигляді масиву Item-ів. Для зручності при подальшому порівнянні, до кожного Item-у додамо хеш його контенту.

Щодо отримання стану проєкту відповідно до поточних файлів у робочій директорії, ми обійдемо всі файли, пропускаючи службові файли (наприклад, саму службову директорію СКВ) і помічені для ігнорування користувачем. Для кожного файлу створюватимемо тимчасовий Item без контенту, але зі шляхом і хешом вмісту файлу. В результаті, незалежно від того, стани чого саме ми порівнюємо, ми завжди працюємо з масивами Item-ів із доданими хешами вмісту файлів.

Коли обидва стани отримані, можна проводити саме порівняння. Для цього попередньо представимо обидва стани у вигляді асоціативних масивів, де ключем буде шлях до файлу, а значенням – відповідний Item. Далі, достатньо обійти всі існуючі ключі для обох масивів. Якщо ключ існує лише в одному з них – фіксуємо це як відповідну зміну (видалення або додавання), якщо ж ключ присутній в обох масивах, але хеш вмісту відрізняється – фіксуємо це як модифікацію. Решта ключів вважаються не зміненими файлами. Усі зафіксовані зміни повертаємо як результат порівняння.

На основі порівняння реалізовується створення нового коміту: отримується стан на момент

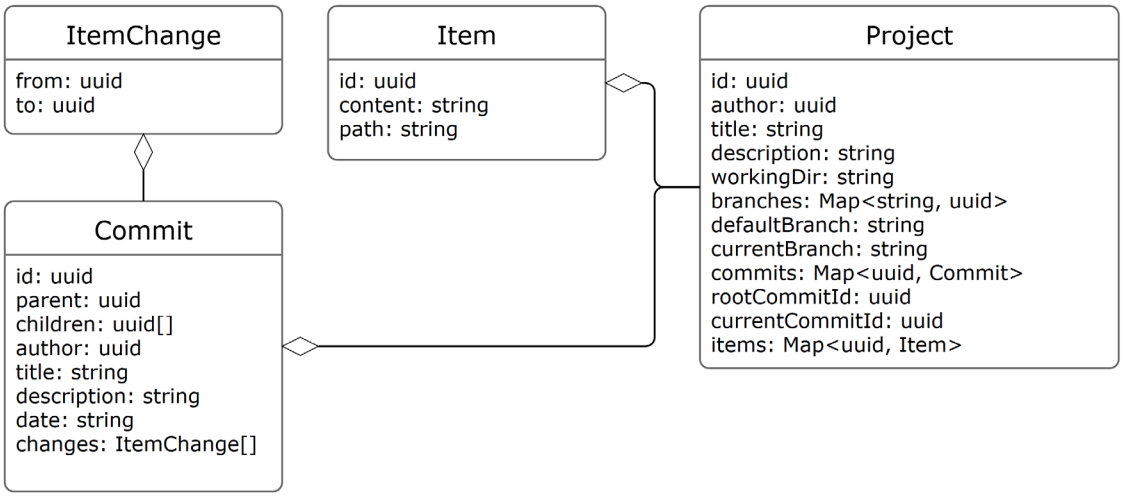

Рис. 1. Діаграма основних класів пакету СКВ

останнього коміту поточної гілки, поточний стан файлів у директорії, здійснюється порівняння, і для кожної з виявлених змін створюється ItemChange. Для нових та модифікованих файлів, за потреби, створюються нові Item-и, та зберігаються в проєкті, і зберігається новий контент відповідно.

Після реалізації інших функцій проєкту (збереження до файлу або JSON, відтворення з файлу або JSON, автодоповнення частини ідентифікатора коміту тощо), пакет готовий до використання. Разом з усіма класами проєкту, в ньому також буде присутній інтерфейс IStorageProvider, який передбачає усі основні операції з файловою системою (читання/створення/видалення файлів та директорій, перевірка існування файлу/директорії, копіювання файлу тощо), а також деякі імплементації цього інтерфейсу, завдяки чому пакет буде придатний до використання в різних середовищах: файлова система комп'ютера користувача, хмарне сховище, Google Диск користувача та інші сховища.

Розробка клієнтського додатку. Клієнтський додаток на період розробки Proof of Concept можна реалізувати у вигляді консольного додатку. Оскільки основна частина – СКВ – реалізована у вигляді окремого незалежного пакету, реалізація віконного додатку на її основі не вимагатиме значних ресурсів. Для розробки консольного додатку обрано npm-пакет commander, разом із додатковими пакетами для форматування і зручної взаємодії з користувачем (chalk, inquirer, nanospinner).

Перед здійсненням будь-яких операцій з проєктом, які стосуються веб-сервісу, клієнтський додаток вимагатиме автентифікації користувача. Задля забезпечення зручності і водночас надійності, реалізовується автентифікація згідно OAuth 2.0 Device Flow (Рис. 2). Цей механізм передбачає вхід користувача в його обліковий запис в браузері після ініціювання автентифікації додатку на пристрої. Після успішного входу користувача і його згоди, веб-платформа згенерує токен для авторизації, та відправить додатку, який в режимі очікування з певним інтервалом опитуватиме веб-платформу для отримання цього токену. До будь-яких подальших запитів клієнтський додаток додаватиме цей токен в заголовках запиту, отримуючи доступ до відповідних ресурсів.

Розробка веб-платформи. Веб-платформу буде реалізовано з використанням фреймворку NextJS. В якості БД обрано PostgreSQL, а для взаємодії з нею використовуватиметься Prisma ORM. Окрім користувацького інтерфейсу, засобами NextJS реалізується API для роботи з клієнтським додатком.

Сутності БД слід почати розглядати з сутності користувача – User. Вона міститиме ідентифікатор користувача, особисті дані, та інформа-

Рис. 2. Послідовність автентифікації клієнтського додатку

цію для авторизації. Другою важливою сутністю є сховище – Storage. Вона репрезентує сховище для файлів, доступне користувачеві. За замовчуванням користувачеві доступне власне сховище веб-платформи, проте він може додати власне: під'єднати власний Google-диск, Dropbox, або в перспективі навіть власний пристрій (наприклад, домашній сервер) в якості сховища за допомогою спеціального додатку. В цій сутності зберігатиметься вся необхідна інформація для під'єднання та отримання доступу до сховища.

Проєкту відповідатиме сутність Project. В ній зберігатиметься JSON-представлення проєкту, сховище для зберігання, налаштування публічності тощо. Між сутностями Project, User та Storage будуть також проміжкові таблиці, визначаючі права доступу користувача до певного сховища та певного проєкту.

Через веб-інтерфейс користувач зможе керувати своїм профілем, своїми проєктами (переглядати структуру та історію змін, змінювати налаштування, надавати доступ іншим користувачам тощо), своїми сховищами (переглядати статус, налаштовувати доступ користувачів тощо), та інші функції які в перспективі можна додати на веб-платформу. Решта функцій, зокрема, безпосередня взаємодія з файлами проєктів, покладається на клієнтський додаток.

Отже, одна з найголовніших функцій веб-платформи – передача даних між сховищем і користувачем. Це включає в себе роботу з конкретним сховищем, приєднаним до проєкту, перевірку прав доступу користувача до запитаного ресурсу та інші операції.

Оскільки система працюватиме безпосередньо з даними для автентифікації на користувацькому сховищі, а доступ до проєкту може бути навіть відкритий для всіх користувачів, потрібно унеможливити розкриття вразливої інформації стороннім особам. При цьому вибудовування складного ланцюга перевірок та запитів може суттєво сповільнити передачу файлів, що також не є бажаним. Для забезпечення одночасно безпеки і швидкості роботи, буде використано механізм підписаних посилань Signed URLs.

Одна з основних переваг Signed URLs – пряме з'єднання клієнта і сховища без посередників та проміжкових обробок і відповідно навантажень на сервер. Проте, не всі сховища підтримують цей механізм. Зокрема, одна із найближчих середньостатистичному користувачеві опцій – Google-диск. На момент дослідження, API Google-диску не підтримує видачу Signed URLs, і аналогічна ситуація може виникати з іншими сховищами, підтримка яких додаватиметься в перспективі.

Щоб вирішити цю проблему, на веб-платформі введемо певний фасад, який буде відповідати за всі запити пов'язані із файлами. Якщо клієнтському додатку або веб-інтерфейсу потрібно завантажити файл зі сховища або на сховище, ініціюється передача файлу надсиланням запиту на відповідний ендпоінт. В цьому запиті передається перелік файлів, з якими планується затребувана операція. Цей запит потрапляє в обробку фасаду, після чого перевіряються права доступу користувача до проєкту, якому належить ресурс, а також сховище на якому він зберігається.

Якщо це сховище має вбудовану підтримку підписаних посилань – ці посилання отримуються напряму від сховища і передаються у відповіді клієнтському додатку, надаючи можливість обміну даними зі сховищем напряму, без викриття вразливих даних при цьому. Якщо ж сховище, на якому зберігається проєкт, не підтримує підписані посилання, тоді цю відповідальність бере на себе фасад, видаючи клієнту власні підписані посилання, і самостійно реалізовуючи їх обробку і роботу зі сховищем. Таке рішення може вимагати додаткових ресурсів серверу та дещо ускладнює розробку, проте досягнуті безпека та швидкість нівелюють це.

При проєктуванні веб-платформи також варто приділити увагу аспектам інфраструктури. Одна з найкращих опцій для розгортання NextJS додатків – платформа Vercel. У такому випадку будуть доступні помірна вартість розгортання (включаючи безкоштовний план), тісна інтеграція із самим фреймворком, логування та моніторинг тощо. Проте, цінова політика Vercel не є оптимальною для активного навантаження, такого як, наприклад, фасад, що відповідатиме за передачу файлів. Подібне навантаження буде оптимальніше розгортати на віртуальному приватному сервері (VPS). Таким чином, для найбільш оптимального використання платформою доступної інфраструктури, ці дві частини – фасад і NextJS додаток, варто розділити.

В той самий час, потрібно забезпечити комунікацію цих двох частин, адже дані, потрібні фасаду (наприклад, конфігурація певного сховища), зберігатимуться в БД, до якої має доступ лише NextJS додаток. Налагодити роботу двох різних додатків з однією БД і спільною схемою вкрай важко: є ризики конфліктів при невідповід-

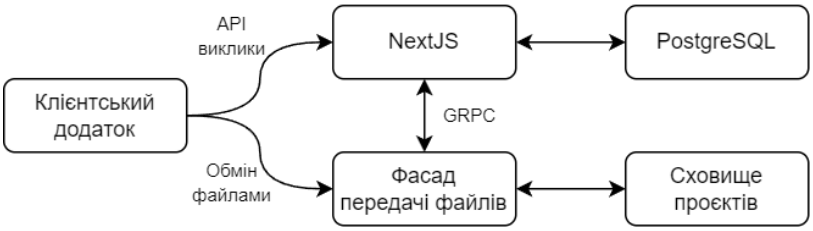

Рис. 3. Загальна архітектура системи

ності схем, складніше керувати доступом до БД тощо. Тому доступ до даних необхідних фасаду буде реалізовано саме через комунікацію із NextJS додатком. Для цієї ролі можна застосувати технологію gRPC. Вона забезпечить водночас зручність розробки (типізація, робота із окремим сервісом через виклики методів у програмному коді, а не прості HTTP запити тощо) і швидкість роботи.

Висновки. В цій статті було розглянуто розробку рішення із системою контролю версій для працівників музичної індустрії. Рішення представляє собою готову до використання та масштабування систему з клієнтського додатку, веб-платформи та фасаду для передачі файлів. Хмарна частина рішення є модульною та готовою до роботи на оптимальній інфраструктурі, забезпечуючи високу ефективність та помірну вартість розгортання, можливість балансування навантаження та обох видів масштабування (горизонтальне і вертикальне). Порядок обміну файлами передбачає ізоляцію вразливих даних, таких як облікові дані від сховища проєктів, від сторонніх осіб. Функціонал розробленої СКВ та можливості веб-платформи у сукупності покривають всі основні потреби різних ролей у робочому процесі виробництва музики, а під час самої розробки було досліджено аспекти архітектури ПЗ, безпеки даних та оптимізації роботи з файлами.

Список літератури:

1. Зуєвський Д.Р., Кулішова Н.Є. Розробка внутрішньопрограмної системи контролю версій: Радіоелектроніка та молодь у XXI столітті: ХНУРЕ, 2023, Том 6 ч. 2, 182 с.

2. Alexander Y., Benjie C., Robert M. Pastwatch: a Distributed Version Control System: NSDI ’06: 3rd Symposium on Networked Systems Design & Implementation, 381 p.

Rehenel T.Yu., Fant M.O., Savitskyi R.S. DEVELOPMENT OF A VERSION CONTROL SYSTEM FOR MUSIC PROJECTS

The article describes the development of a version control system for music industry workers. The development includes the version control system itself, as well as an ecosystem of software products based on it, including a client application for working with project files directly on the user’s computer, as well as a web platform for storing, managing, and collaborating on projects. Despite the availability of a wide variety of software that covers almost all the basic needs of various workflows in the music industry, there are still no tools for organizing and controlling files of different versions of a project. Existing alternatives like Git and GitHub are not widely used in this area for a number of reasons, including poor system optimization for working with large files (over 50 MB, while the average file size of a recording session for musicians can reach several gigabytes), requirements for user skills in working with the command line, authentication tools, etc, as well as the focus of the functionality on working with code and the specifics of the workflow of programmers, testers, DevOps engineers, etc., which has a number of differences from the workflow in the music industry. File sharing services such as Google Drive or DropBox are more commonly used, and some of them have built-in file versioning. However, this versioning is very limited in its functionality, and there is no possibility for several participants to work in parallel (the version history is linear and does not support branching). As a result of the study, the version control system itself was designed, based on which the product, consisting of a client application and a web platform with support for connecting user storage, covers all these shortcomings, and at the same time offers no worse functionality, significantly increasing the efficiency and quality of the workflow of workers in the field.

Key words: *version control system, versioning, file storage, web platform, software.*