ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА МІКРОСЕРВІСУ ДЛЯ ПЛАНУВАННЯ СВЯТКОВИХ ПОДІЙ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ SPRING BOOT, МОВОЮ JAVA**»

Виконала: студентка 4 курсу, групи ПД-41

спеціальності

121 Інженерія програмного забезпечення

(шифр і назва спеціальності)

Ковалевська Ю. С.

(прізвище та ініціали)

Керівник Негоденко О. В.

(прізвище та ініціали)

Рецензент _____

(прізвище та ініціали)

Київ – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти – «Бакалавр»

Напрям підготовки – 121 – «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

О. В. Негоденко

«_____» _____ 2023 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

КОВАЛЕВСЬКОЇ ЮЛІЇ СЕРГІЇВНИ

(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка мікросервісу для планування святкових подій з використанням технології Spring Boot, мовою Java»

Керівник роботи _____,

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «24» лютого 2023 року № 26.

2. Строк подання студентом роботи «1» червня 2023 року

3. Вхідні дані до роботи:

3.1 Науково-технічна література, пов'язана із розробкою вебдодатків

3.2 Мова програмування Java

3.3 Фреймворки Spring Boot та Spring Data JPA

3.4 Фреймворки для тестування JUnit та Mockito

3.5 Система управління базами даних MySQL

3.6 Бібліотека Liquibase

3.7 Бібліотека Google Protocol Buffers

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).

4.1 Аналіз предметної області

4.2 Аналіз та дослідження існуючих аналогів

4.3 Аналіз засобів розробки

4.4 Аналіз розроблюваного мікросервісу

4.5 Розробка програмного забезпечення

4.6 Тестування розробленого програмного забезпечення

5. Перелік графічного матеріалу

5.1 Мета, об’єкт та предмет дослідження

5.2 Аналоги

5.3 Технічне завдання

5.4 Програмні засоби реалізації

5.5 Розроблений мікросервіс для планування святкових подій

5.6 Висновки

6. Дата видачі завдання «25» лютого 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	25.02.2023 – 27.02.2023	Виконано
2	Аналіз предметної області та дослідження існуючих аналогів	16.04.2023 – 17.04.2023	Виконано
3	Дослідження програмних засобів	17.04.2023 – 18.04.2023	Виконано
4	Проєктування мікросервісу	30.04.2023 – 01.05.2023	Виконано
5	Розробка мікросервісу	01.05.2023 – 15.05.2023	Виконано
6	Вступ, висновки, реферат	15.05.2023 – 18.05.2023	Виконано
7	Розробка обов’язкових демонстраційних матеріалів	18.05.2023	Виконано
8	Попередній захист роботи	22.05.2023	Виконано
9	Здача роботи	01.06.2023	Виконано

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина магістерської роботи 90 с., 20 рис., 35 джерел.

BACK-END, JAVA, МІКРОСЕРВІС, RESTful API, SPRING BOOT, СОЦІАЛЬНА МЕРЕЖА, ВЕБДОДАТОК

Об'єкт дослідження – процес планування святкових подій.

Предмет дослідження – програмне забезпечення для планування святкових подій.

Мета роботи – спрощення процесу планування святкових подій за рахунок мікросервісу, написаного мовою програмування Java.

Методи дослідження – методи проєктування та розробки програмного забезпечення, методи опрацювання та аналізу отриманих результатів, методи тестування програмного забезпечення.

Здійснено аналіз існуючих архітектур вебдодатків і для розробки обрано REST та мікросервісну архітектури. Проаналізовано існуючи аналоги програмного забезпечення. Обрано Spring Boot + Java для розробки мікросервісу та MySQL + Liquibase + Spring Data JPA для роботи із базою даних. Визначено структуру вебдодатку і візуалізовано з допомогою UML діаграм. Розроблено мікросервіс для планування святкових подій за допомогою попередньо обраних засобів розробки. Визначено взаємозв'язки між розробленим мікросервісом та іншими потенційними мікросервісами вебдодатку.

Розроблений мікросервіс може бути використаний для подальшого масштабування вебдодатку.

Галузі використання – комунікації, розваги, освіта, громадська діяльність/активізм.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення

IDE – Integrated Development Environment

HTTP – HyperText Transfer Protocol

JPA – Java Persistence API

API – Application Programming Interface

CI/CD – Continuous Integration/Continuous Deployment

XML – Extensible Markup Language

JAR – Java Archive

JSON – JavaScript Object Notation

YAML – YAML Ain’t Markup Language

SQL – Structured Query Language

СУБД – Система управління базами даних

ОС – Операційна система

DTO – Data Transfer Object

REST – Representational State Transfer

UUID – Universally Unique Identifier

UML – Unified Modeling Language

ВСТУП

Обґрунтування вибору теми та її актуальність: Соціальні мережі є невід'ємною частиною життя сучасного суспільства. Їх використання зручне, швидке та доступне, чим зумовлена їх велика популярність. Соціальна мережа може зібрати всю корисну інформацію в одному місці значно зручнішим способом.

Саме тому, для розробки вебдодатку з оголошення святкових подій, було обрано соціальну мережу. Вона дасть можливість зібрати будь-які заплановані події в одному місці, при цьому охопити більшу аудиторію в силу популярності соцмереж.

Такий тип додатку дасть можливість користувачам, що шукають події, легко відфільтрувати необхідну їм інформацію, знайти інформацію про подію своєчасно, а користувачам, що оголошують святкові події, знайти нових учасників, збільшити кількість відвідувачів, зібрати відгуки для потенційного покращення якості оголошених подій.

Розробка системи саме для оголошення подій може допомогти людям налагодити комунікації між собою, знайти нових друзів чи хобі.

Об'єктом дослідження є розробка мікросервісів з допомогою Spring Boot фреймворку.

Предмет дослідження є мікросервіс розроблений за допомогою Spring Boot фреймворку.

Метою роботи є розробка мікросервісу з допомогою Spring Boot фреймворку.

Методи дослідження: для початку треба було ознайомитись із існуючими системами для визначення вимог до власної системи. Визначитись із технологіями, які будуть використовуватись для проєктування і розробки, а саме Java, Spring Boot, Spring Data JPA, MySQL, Liquibase, Google Protocol Buffers, JUnit та Mockito. Далі були обрані архітектурні шаблони, відповідно до яких був спроєктований сам мікросервіс для оголошення святкових подій з допомогою UML діаграм.

Дослідження процесу розробки відбувалось саме під час розробки мікросервісу.

Наукова новизна роботи полягає в розробці нового мікросервісу для планування святкових подій та використанні сучасних технологій та архітектур для його реалізації.

Практична значущість результатів дослідження полягає в можливості використання розробленого мікросервісу як основи для подальшої розробки та розширення функціональності соціальної мережі. Мікросервіс надає можливість створити вебдодаток для планування святкових подій, що дозволить їм ефективно організовувати та контролювати події.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Соціальні мережі

Соціальна мережа – це термін, що використовується для позначення соціальних комунікацій між людьми в інтернеті. Вони використовуються для створення зв’язків між знайомими людьми, незнайомцями, що мають спільні інтереси чи мету, обміну інформацією, публікування контенту та отримання фідбеку щодо нього, створення користувацьких профілів.

Не зважаючи на широкий спектр використання соціальних мереж, саме зв’язки та стосунки між людьми є основною ідеєю соцмереж.

Соціальні мережі мають чотири основні цілі:

– зв’язки між людьми;

– обмін інформацією;

– навчання;

– маркетинг.

Соціальні мережі поділяються на такі найпоширеніші категорії:

– для соціальних зв’язків: використовується для того, аби підтримувати зв’язки між людьми, ділитись особистою інформацією через профілі. Приклади: Facebook, Instagram;

– для професійних зв’язків: використовується для ділових комунікацій або пошуку роботи чи спеціалістів. Приклади: LinkedIn;

– для обміну мультимедіа: використовується для обміну відео, фото чи музики. Приклади: YouTube;

– для обміну новинами чи інформацією: використовується для публікування новин, або інформаційного контенту, що дозволяє людям вирішити певні задачі чи знайти відповіді на певні питання. Приклади: Reddit, StackOverflow;

– для спілкування: використовуються для того, аби дати можливість користувачам спілкуватись між собою в окремих чи групових чатах. Переважно це програми для обміну миттєвими повідомленнями. Приклади: WhatsApp, Telegram;

– для освіти: використовуються для організації дистанційного навчання. Приклади: Google Classroom.[1]

Переваги соцмереж:

– безмежне спілкування;

– легке поширення контенту, новин;

– сприяння саморозвитку;

– дієвий маркетинг;

– легке створення бізнесу;

– притупляють відчуття самотності.

Недоліки соцмереж:

– збирають надто багато особистої інформації;

– розвинене шахрайство та крадіжки даних;

– реклама, спам та пропаганда;

– викликають залежність;

– сприяють асоціальності в реальному житті, заміняють живе спілкування;

– впливають на можливість грамотно висловлюватись і писати;

– поширення аморального контенту.[2]

1.2 Огляд та аналіз існуючих аналогів

Розглянемо два аналоги систем для оголошення подій. Це Eventbrite та Hopin.

1.2.1 Eventbrite

Eventbrite є глобальною платформою для самостійного продажу квитків для різноманітних подій, яка надає можливість створювати, ділитися, знаходити та відвідувати події.[3]

Функції даного ПЗ:

– управління та планування подій;

– сканування квитків;

– нагадування;

– автоматизоване просування подій;

– індивідуальні карти місць;

– онлайн продажі та резервування квитків;

– приватні події;

– фандрайзинг;

– опитування аудиторії;

– звітність та аналітика;

– профілі організаторів.[4]

Переваги:

– легко використовувати;

– є мобільні додатки та сайт;

– інтеграції з іншими системами;

– безкоштовно для безкоштовних подій;

– має стандартні шаблони для подій.

Недоліки:

– мобільний додаток розділений на два окремих додатки для організаторів і відвідувачів;

– складна для використання резервація місць;

– великий податок на дорогі квитки;

– деякі інтеграції з іншими системами погано протестовані;

– не вистачає демонстрації усього функціоналу, який часто є неочевидним.[5]

1.2.2 Hopin

Hopin – це платформа для організації онлайн, офлайн та гібридних подій.

Функції даного ПЗ:

– управління та планування подій;

– чати;

– бронювання та купівля квитків;

– онлайн кімнати для зустрічей;

– e-mail розсилка;

– хост для конференцій;

– просування подій;

– аналітика.

Переваги:

– легко використовувати;

– є мобільний додаток та сайт;

– інтеграції з іншими системами;

– чати;

– події проводяться прямо в додатку;

– якісна технічна підтримка;

– є спільнота.

Недоліки:

– доступно мало безкоштовних функцій;

– високі ціни;

– не можна створювати повторювані події;

– низька якість трансляцій;

– не вистачає більш деталізованих інструкцій з використання.[6]

2 ЗАСОБИ РОЗРОБКИ

2.1 Середовище розробки та засоби тестування

IntelliJ IDEA – це провідне інтегроване середовище розробки для написання коду мовами програмування Java та Kotlin, розроблене компанією JetBrains. Дане IDE має безліч інструментів для аналізу, налагодження та рефакторингу коду. Є можливість глибокої конфігурації даного середовища для підвищення ефективності написання коду, а також є можливість встановлення плагінів для розширення функціоналу.

Є дві версії IDE: безкоштовна – Community та платна, із розширеним функціоналом – Ultimate. Платна версія має такі переваги, як керування базами даних, профайлер, вбудований HTTP клієнт. Ця версія також має кращу інтеграцію із деякими фреймворками, як Spring і Spring Boot, Micronaut, Quarkus, JPA та Hibernate.

Окрім мов програмування Java та Kotlin, дана IDE розуміє багато інших мов, наприклад Groovy, Scala, JavaScript, SQL та інші.

Розвинена система навігації, пошуку та пошуку з заміною дозволяє полегшити ці завдання і більше зосередитись на самому написанні коду.

IntelliJ IDEA також підтримує найпопулярніші системи контролю версій Git, Subversion, Mercurial та Performance. Особливо зручно ними користуватись з допомогою вбудованого терміналу.[7]

Postman – це ПЗ, що використовується для тестування API. З допомогою цієї програми можна надсилати HTTP запити на вебсервер. Дана програма широко використовується розробниками та тестувальниками.

Postman легко інтегрується із CI/CD, дозволяє легко налаштовувати і автоматизовувати тести та писати набори тестів.[8]

2.2 Мова програмування Java

Java – це відома та популярна об’єктно-орієнтовна високорівнева мова програмування та програмна платформа. Написані нею програми використовуються мільярдами різних пристроїв, таких як ноутбуки, мобільні пристрої, ігрові консолі та багато іншого. Дана мова програмування є С-подібною, написаною на базі мов програмування С та С++.

Однією з переваг є дистрибутивність, адже код написаний на Java буде працювати на будь-якому пристрої. Головним гаслом є саме «написати один раз, запустити всюди» («Write once, run anywhere»). Таку мету собі ставив винахідник даної мови програмування Джеймс Ґослінґ із компанії Sun Microsystems.

Для того, щоб почати розробку треба мати Java Development Kit (JDK), який включає в себе Java Runtime Environment (JRE), що у свою чергу включає в себе Java Virtual Machine (JVM). Компілятор компілює написаний код в байткод, який потім інтерпретується з допомогою JVM. Саме завдяки віртуальній машині цей байткод може запускатись на будь-якому пристрої, головною умовою є саме наявність JVM.

Окрім дистрибутивності є ще багато переваг, таких як надійність, простота та безпека. Мова програмування Java є ідеальним вибором для створення вебдодатків, адже наділена такими можливостями, як керування транзакціями, кластеризація, продуктивність, стабільність та масштабованість.[9]

2.3 Фреймворки

Spring Boot – це фреймворк написаний на базі Spring фреймворку, що робить розробку вебдодатків і мікросервісів простішою та швидшою.

Від Spring фреймворку Spring Boot успадкував функцію ін’єкції залежностей («dependency injection») та контейнер інверсії управління («inversion of control

container»), що дозволяє легко управляти залежностями програми, що складаються із слабко зв’язаних компонентів, які легко масштабувати та тестувати.

Spring Boot має велику перевагу над Spring, тому що він автоматично налаштовує усі стандартні залежності, дає можливість швидко розпочати роботу і сконцентруватись саме на бізнес логікі. Також Spring Boot пропонує вбудований Tomcat вебсервер, з яким розгортання програми проходить швидше.[10]

Також можна виділити ще кілька переваг:

– для додаткових конфігурацій не потрібно використовувати XML, адже Spring Boot має багато корисних анотацій;

– достатньо створювати JAR файли, для того, щоб розгортати додатки;

– має велику спільноту.[11]

Spring Data JPA – це проект із сімейства Spring Data, що полегшує отримання доступу до даних, що зберігаються в реляційних базах даних. Даний фреймворк забезпечує абстракцію для рівня доступу до даних («domain layer»), що дозволяє писати менше шаблонного коду і більше зосередитись на бізнес задачах. Для цього нам потрібно лише створити інтерфейси із абстрактними методами та сутності із відповідними до потрібних таблиць структурами. Для доступу до джерела даних, потрібна мінімальна конфігурація, адже з за це відповідальний Spring Boot.[12]

JUnit – це фреймворк призначений для модульного тестування програм, написаних мовою програмування Java. Він використовується для написання та виконання автоматизованих тестів. Тестування з допомогою JUnit дозволяє швидко писати надійний код і перевіряти чи працює він належним чином.[13]

Mockito – це фреймворк для імітування залежностей, який використовується для тестування разом із фреймворками JUnit та TestNG. Внутрішньо даний фреймворк використовує Java Reflection API, що дозволяє йому створювати фіктивні об’єкти чи повертати фіктивні дані, уникаючи при цьому зовнішніх залежностей. Це спрощує розробку тестів, імітуючи зовнішні залежності та застосовуючи несправжні об’єкти у тестах, що забезпечує їх ізольованість.[14]

Liquibase – це фреймворк, що дозволяє визначати структуру бази даних і відстежувати зміни в цій структурі. З допомогою цього фреймворку можна

працювати із різними типами баз даних (реляційні або нереляційні), приймаючи різні формати файлів, що визначатимуть їх структуру. Такими форматами є: XML, JSON, YAML та SQL.

Liquibase дозволяє легко ініціалізувати, вносити зміни до бази даних, відкочувати їх на попередні версії у разі необхідності. Усе це можна робити у файлах наборів змін («changesets»). Коли до програми буде доданий новий набір змін, то Liquibase автоматично перевірить цілісність бази даних і внесе нові зміни, або видасть помилку, у випадку порушення цілісності.[15]

Google Protocol Buffers – це розширений механізм, що дозволяє серіалізувати структуровані дані незалежно від мови програмування. Буфери протоколів схожі на JSON, але є меншими та швидшими. Вони дозволяють швидко генерувати код обраною мовою програмування, шляхом визначення структури даних у файлах із розширенням *.proto*. Ці файли компілюються протокомпілятором – саме він генерує вихідні файли.

Буфери протоколів забезпечують серіалізацію даних, формат яких підходить як для тимчасового, так і для тривалого зберігання даних. Вони можуть розширюватись, не впливаючи на існуючі дані і не роблячи їх недійсними, також не вимагають подальшого оновлення коду.[16]

2.4 Система управління базою даних

MySQL — це система з відкритим кодом для управління реляційними базами даних. Подібно до інших систем управління реляційними базами даних, ця система зберігає дані в таблицях, що містять у собі рядки та стовпці. Користувачі системи керують даними за допомогою мови структурованих запитів SQL.[17]

Система MySQL швидка, надійна, масштабована і проста у використанні. Вона була розроблена спеціально для швидкої обробки великих баз даних і використовується у високовимогливих виробничих середовищах.

MySQL постійно розвивається та пропонує багатий набір функцій. Також ця СУБД є дуже зручною для доступу до баз даних, що знаходяться в Інтернеті.

Основні переваги:

– проста у використанні;

– надійна;

– масштабована;

– швидка;

– має високу доступність (надає повний набір власних інтегрованих технологій реплікації даних);

– безпечна;

– гнучка.[18]

2.5 Документування та система контролю версій

Jira – це програма, розроблена компанією Atlassian, яка дозволяє командам відстежувати проблеми, керувати проектами та автоматизувати робочі процеси.

Ключові концепції:

– проблема («issue») – це один робочий елемент, який ви відстежуєте від створення до завершення. Проблемою може бути будь-яке завдання, яке має виконати команда;

– проект («project») – це спосіб згрупувати проблеми за відповідним контекстом. Зазвичай створюються окремі проекти для кожного продукту;

– дошка («board») – це візуальне представлення робочого процесу, яке відображає усі наявні проблеми та стадії на яких вони знаходяться;

– робочий процес («workflow») – це шлях, яким проходять проблеми. Кожна мітка в робочому процесі, як-от «виконати», «виконується» та «готово», являє собою стадію, на якій знаходиться проблема.[19]

Confluence – це інстумент для співпраці розроблений компанією Atlassian, яке надає платформу для командної взаємодії. Дана платформа дозволяє структурувати

та організовувати роботу команди, завдяки зручному спільному доступу до важливої інформації щодо проєкту.

Ключові концепції:

– сторінка («page») – документ, що містить якусь конкретну інформацію, яка стосується проекту (вимоги, нотатки зустрічей, плани проекту та інше);

– простір («space») – робоча область, де упорядковано зберігаються сторінки;

– дерево сторінок («page tree») – ієрархічно організовані сторінки, для легкого пошуку.[20]

Git – це розподілена система керування версіями, яка дозволяє зберігати код, відстежувати версії коду, об’єднувати внесені зміни та повертатися до попередньої версії коду.

Git зберігає код і повну історію його розробки локально в репозиторії в гілках («branch»), зміни до яких можна вносити паралельно. Гілки можна зливати («merge»), або відкочувати зміни в них до попередньої версії («revert»). Кожного разу, коли відбувається зберігання змін («commit»), Git порівнює поточну та попередню версії коду, і якщо зміни відбулись, то він зберігає їх.[21]

Bitbucket – це сервіс для керування Git репозиторієм, призначене для спільної роботи над вихідним кодом.

Основні функції:

– контроль доступу до вихідного коду;

– керування робочим процесом;

– пул-запити («pull request»). Використовуються для перевірки написаного коду, що знаходиться в гілці, яка є кандидатом на злиття;

– інтеграція із Jira.[22]

3 ВИМОГИ ТА ПРОЄКТУВАННЯ

3.1 Загальний опис

Мікросервіс з планування святкових подій буде складатись із одного основного та трьох додаткових модулів, що будуть мати спільного батьківського модуля. Основний модуль відповідатиме за бізнес логіку, інші відповідатимуть за DTO, валідацію та обробку помилок.

На рисунку 3.1 зображено діаграму компонентів мікросервісу, яка містить у собі п’ять пов’язаних між собою модулів:

– vechornytsi-events: мікросервіс для планування подій;

– vechornytsi-events-service: основний модуль, що відповідатиме за бізнес логіку, контролери, домен та основні конфігурації. Він буде залежати від додаткових модулів;

– vechornytsi-events-api: батьківський модуль, що буде містити у собі інші додаткові модулі ;

– vechornytsi-events-api-api: модуль, що міститиме в собі DTO, згенеровані з допомогою Google Protocol Buffers;

– vechornytsi-events-api-validation: модуль, що міститиме в собі класи для валідації даних надісланих запитів. Він буде залежати від модулю vechornytsi-api-api;

– vechornytsi-events-api-exception: модуль, що міститиме класи помилок та класи для їх обробки.

За допомогою діаграми компонентів система, що розробляється, розбивається на окремі функціональні рівні. Кожен окремий компонент має чітку визначену мету в межах усієї системи та взаємодіє з іншими елементами системи лише за необхідності.[23]

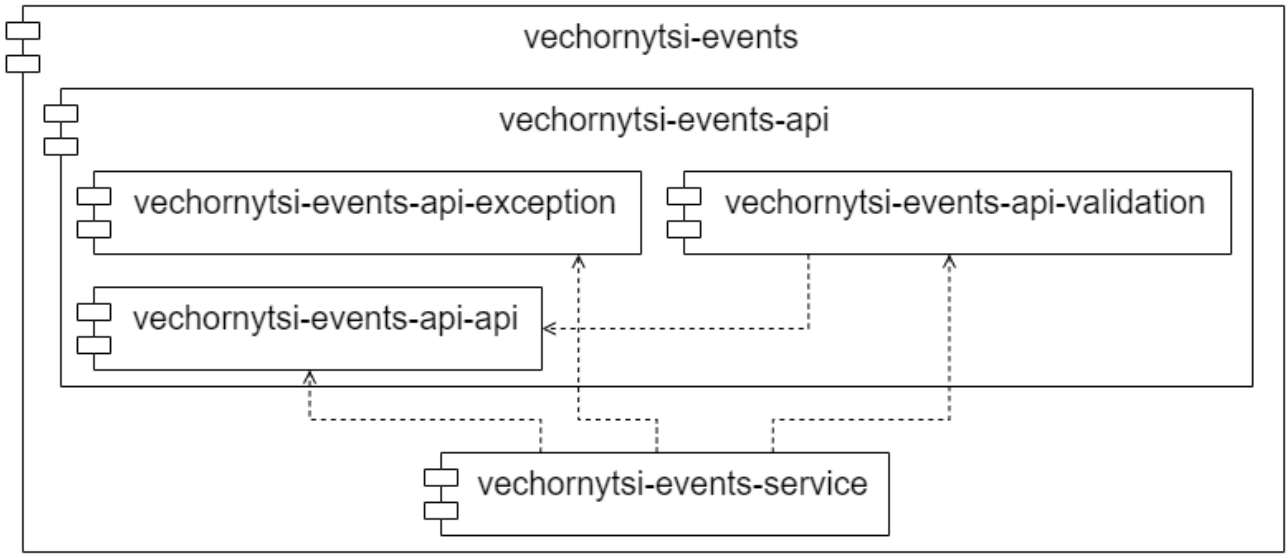


Рисунок 3.1 – Діаграма компонентів

3.2 Архітектура

У цьому розділі буде розглянуто типи архітектур, що були обрані для розробки мікросервісу.

3.2.1 Мікросервісна архітектура

Мікросервісна архітектура являє собою набір пов’язаних між собою автономних сервісів, кожен з яких виконує бізнес задачі свого субдомену.

Якщо брати за приклад систему для планування подій, то така система може мати наступні мікросервіси:

– для планування подій (саме цей мікросервіс буде розроблено);

– для бронювання та купівлі квитків на події;

– для реєстрації/автентифікації;

– для онлайн оплати квитків та інше.

На рисунку 3.2 зображена схема такої системи:

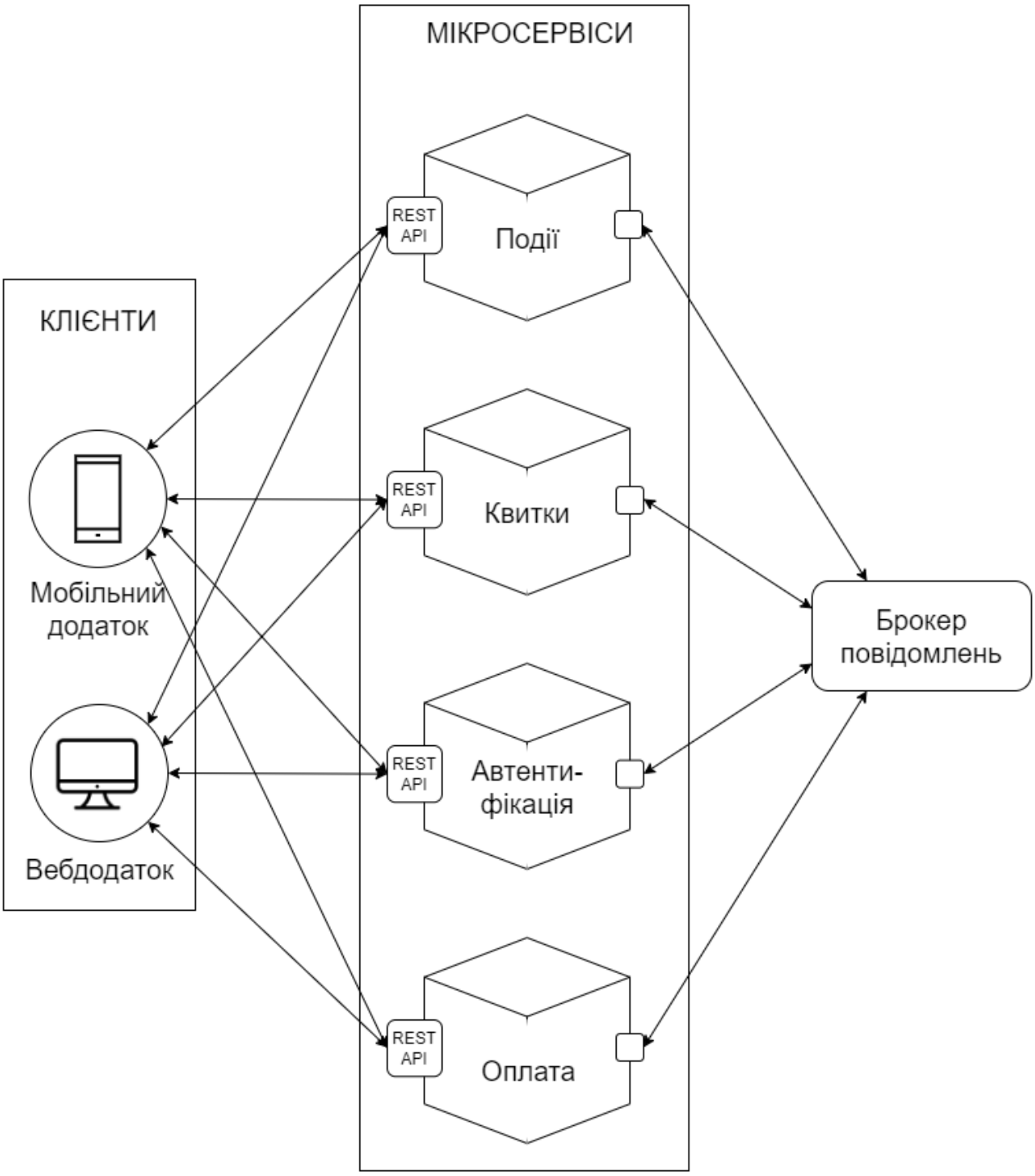


Рисунок 3.2 – Схема архітектури вебдодатку

Операції в подібних системах в основному будуть виконуватись локально, або будуть розподілені між пов’язаними мікросервісами. Такі розподілені операції можуть виконуватись синхронно з допомогою протоколу HTTP, або асинхронно з

допомогою брокерів повідомлень, як Apache Kafka, RabbitMQ, Apache ActiveMQ та інших.[24]

Перевагами мікросервісної архітектури є:

– швидкість розгортання;

– команди невеликі і розподілені між мікросервісами;

– слабка зв’язність коду;

– можливість вибору різних технологій для кожного мікросервісу;

– масштабованість: легко додавати нові мікросервіси;

– можливість ізоляції даних: створення різних баз даних для кожного мікросервісу;

– надійність: якщо один з мікросервісів вийде з ладу, інші залишаться справними і можуть далі виконувати свої функції.

Варто зазначити, що дана архітектура має свої недоліки:

– складність: система в цілому стає складнішою;

– складність тестування мікросервісів і їх зв’язків;

– важливо запроваджувати стандарти для усього проекту, аби полегшити подальшу підтримку;

– ланцюжок залежностей між мікросервісами може стати надто великим, повільним і важко підтримуваним;

– потрібно ретельно слідкувати за цілісністю даних;

– потрібна висококваліфіковані команди, знайомі з даною архітектурою.[25]

3.2.2 REST архітектура

Основною концепцією REST архітектури є представлення даних як ресурсів. Кожен ресурс має свій ідентифікатор та репрезентацію. Ресурс може бути репрезентовано з допомогою JSON, HTML, XML та інших форматів, які підтримуються HTTP.

Суть даної архітектури полягає у створенні RESTful API і отриманні доступу до ресурсів (створювати, редагувати, отримувати, видаляти) з допомогою HTTP запитів.

Перевагами REST архітектури є:

– продуктивність: системи комунікують за допомогою HTTP запитів, що дозволяє підвищити їх продуктивність;

– масштабованість: дозволяє масштабувати взаємодію між компонентами, які надають доступ до ресурсів;

– простота інтерфейсу: при розробці RESTful API для доступу до ресурсів використовуються ендпоінти (посилання, які будуть використовуватись в HTTP запитах для отримання доступу до ресурсів)[26];

– модифікованість компонентів: можна модифікувати ендпоінти незалежно один від одного;

– портативність: дану архітектуру можна реалізовувати незалежно від мови та технологій. Також доступ до ресурсів може мати будь-який пристрій, здатний надсилати HTTP запити;

– надійність: RESTful API не мають стану, тобто їм не потрібно зберігати стан клієнта, що надсилає запити. Тому при збою систему легко відновити.[27]

3.3 Вимоги до системи

Функціональні вимоги – це те, як повинна поводитись система та які дії вона повинна виконувати. Вони характеризують те, як система працює всередині.

Функціональні вимоги мають дві складові частини – це функція та поведінка. Функція є саме дією, яку виконує система, а поведінка це те, як система ту дію виконує.[28]

Було визначено вимоги до мікросервісу для планування подій:

– створення онлайн/офлайн подій;

– скасування подій;

– відновлення скасованих подій;

– оновлення інформації про події;

– пошук події за ID, ім’ям, місцем проведення та датою, а також пошук подій, що проводяться онлайн та повний список активних подій.

Графічно дані вимоги можна зобразити з допомогою діаграми прецедентів, зображеної на рисунку 3.3.

Діаграма прецедентів, або ж діаграма варіантів використання – це представлення вимог до системи у вигляді діаграми. Варіанти використання в даній діаграмі вказують саме на те, що система повинна робити, але не як вона повинна це робити. Ключовою концепцією є те, що дана діаграма допомагає проектувати систему з точки зору користувача (актора).[29]

Акторам в даному мікросервісі виступає вебклієнт.

Загалом мікросервіс для планування подій включатиме в себе одинадцять варіантів використання (не включаючи прецеденти «Створити подію» та «Пошук», адже вони є лише узагальненням для інших прецедентів).

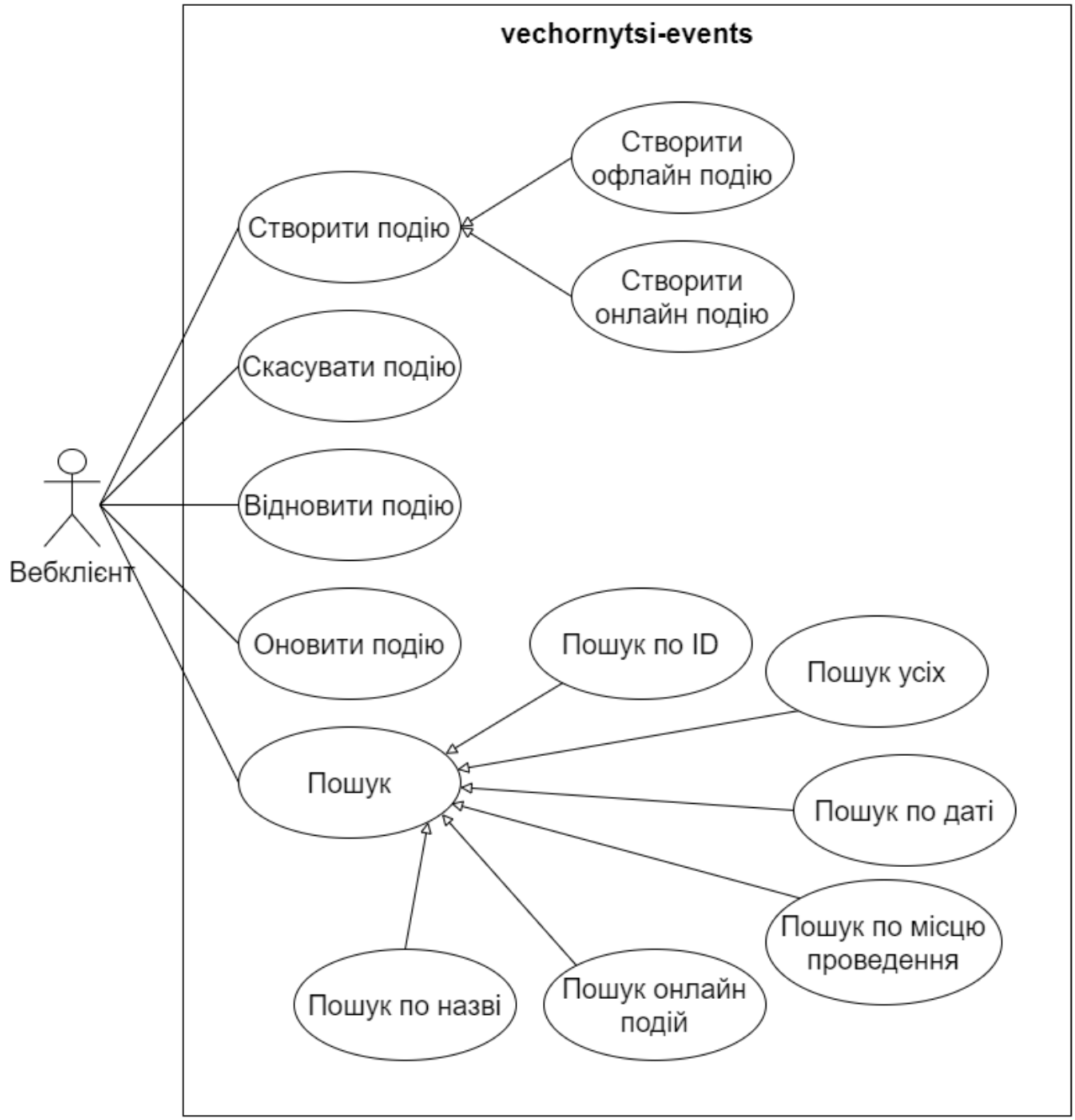


Рисунок 3.3 – Діаграма прецедентів

Нефункціональні вимоги – це вимоги, що допомагають визначити атрибути якості системи та забезпечити її відповідність потребам користувача.[30]

Нефункціональні вимоги:

– Масштабованість: мікросервіс повинен мати можливість масштабування шляхом додавання нових мікросервісів до вебдодатку;

– Портативність: мікросервіс повинен працювати незалежно від платформи.

На основі функціональних та нефункціональних вимог додаємо завдання в Jira. По ходу виконання цих завдань, вони будуть переходити із одного стану в інший, допоки не будуть завершені.

3.4 Проєктування бази даних

Для роботи мікросервісу потрібно буде створити одну сутність «Подія», що включатиме в себе такі дані:

– ID: унікальний ідентифікатор події;

– назва;

– опис;

– дата початку;

– дата кінця;

– місце проведення: поле, що може містити null значення, якщо подія відбувається онлайн;

– онлайн: поле логічного типу, що вказує на те, чи ця подія відбувається онлайн;

– приватна: поле логічного типу, що вказує на те, чи ця подія приватна;

– вартість квитка: поле, що може містити нульове значення, якщо подія безкоштовна;

– кількість вільних місць: поле, яке при ініціалізації події містить початкову кількість вільних місць, яка зменшується кожен раз при бронюванні/купівлі квитка;

– ID організатора: унікальний ідентифікатор організатора. Організатори не зберігаються в базі даних мікросервісу для створення подій, але сутність «Подія» із даного мікросервісу залежить від сутності «Організатор» із іншого мікросервісу, що може бути створений при розробці цілого вебдодатку;

– активна: вказує на те, чи подія є активною чи деактивованою, з можливістю подальношо відновлення.

Таблиця «Подія» зображена за допомогою діаграми зв’язків сутностей на рисунку 3.4. У кожного поля визначено відповідний тип даних.

Діаграма зв’язків сутностей – це графічне представлення сутностей та зв’язків між ними, що використовується для ефективної розробки бази даних.[31]

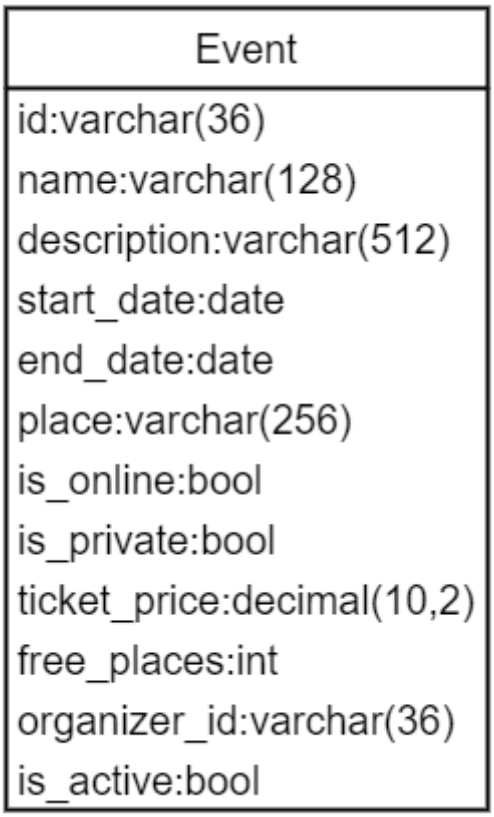


Рисунок 3.4 – Діаграма зв’язків сутностей

3.5 Структура класів мікросервісу

Структура основних класів представлена за допомогою діаграми класів.

Діаграма класів – це тип представлення структури системи, демонструючи її класи, атрибути та методи цих класів, а також зв’язки між ними. Клас являє собою опис атрибутів та поведінки об’єктів. Також класи можна структурувати за допомогою пакетів.[32]

На рисунку 3.5 зображені основні класи та інтерфейси мікросервісу, а саме класи Event, EventServiceImpl та інтерфейси EventService, EventRepository, EventMapper, що знаходяться у пакеті component.event. Також у пакеті controller знаходиться клас EventController. Усі зазначені компоненти знаходяться у пакеті com.vechornytsi.event.service.

Клас Event відповідає за зберігання інформації про події. Клас EventServiceImpl відповідає за бізнес-логіку і імплементує інтерфейс EventService. Також він має залежності від інтерфейсів EventRepository та EventMapper, що відповідають за роботу з базою даних і конвертування об’єктів одного класу в інший відповідно.

Клас EventController відповідає за отримання HTTP запитів і делегування їх обробки до EventService, від якого залежить контролер.

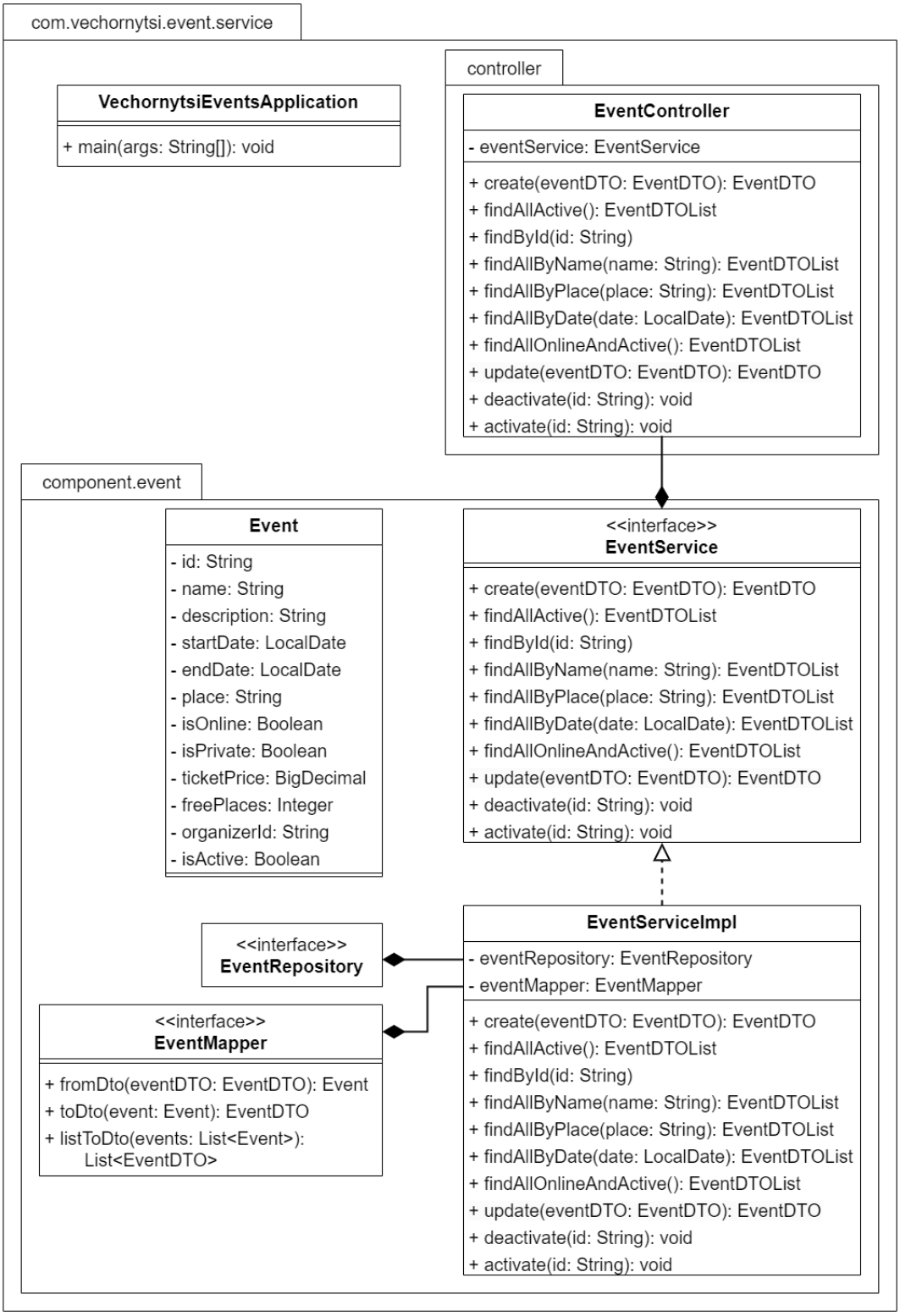


Рисунок 3.5 – Діаграма класів

Усі створені діаграми та описані вимоги заносяться до Confluence, аби мати можливість їх легко переглядати із будь-яких пристроїв і зберегти їх у хмарі.

Для цього створюємо нову сторінку та заносимо туди інформацію, яку потрібно зберегти. На рисунку 3.6 зображена вже збережена сторінка.

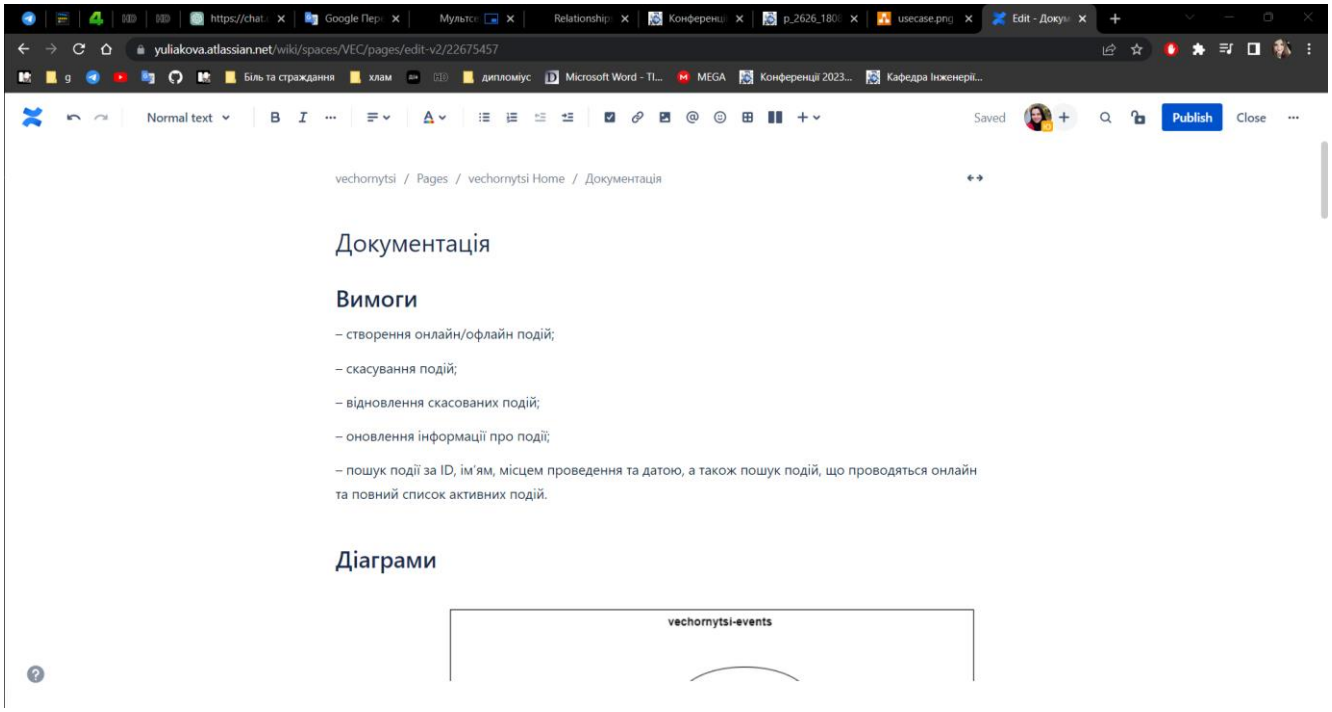


Рисунок 3.6 – Сторінка із документацією у Confluence

4 РОЗРОБКА МІКРОСЕРВІСУ

4.1 Розробка основних модулів

Для початку розробки на комп’ютері попередньо було встановлено OpenJDK, IntelliJ IDEA, Postman, MySQL Server, Git та Protoc.

Першим кроком буде створення проєкту. Для цього потрібно зайти на вебсайт «Spring Initializr» (https://start.spring.io/). Тип проєкту – Maven, мова програмування – Java, версія Spring Boot найновішу стабільну – це 3.0.6.

Далі потрібно заповнити поля із метаданими проєкту:

– *Group*: com.vechornytsi;

– *Artifact*: vechornytsi-events;

– *Name*: vechornytsi-events;

– *Description*: Microservice for events management;

– *Package name*: com.vechornytsi.vechornytsi.events;

– *Packaging*: Jar;

– *Java*: 17.

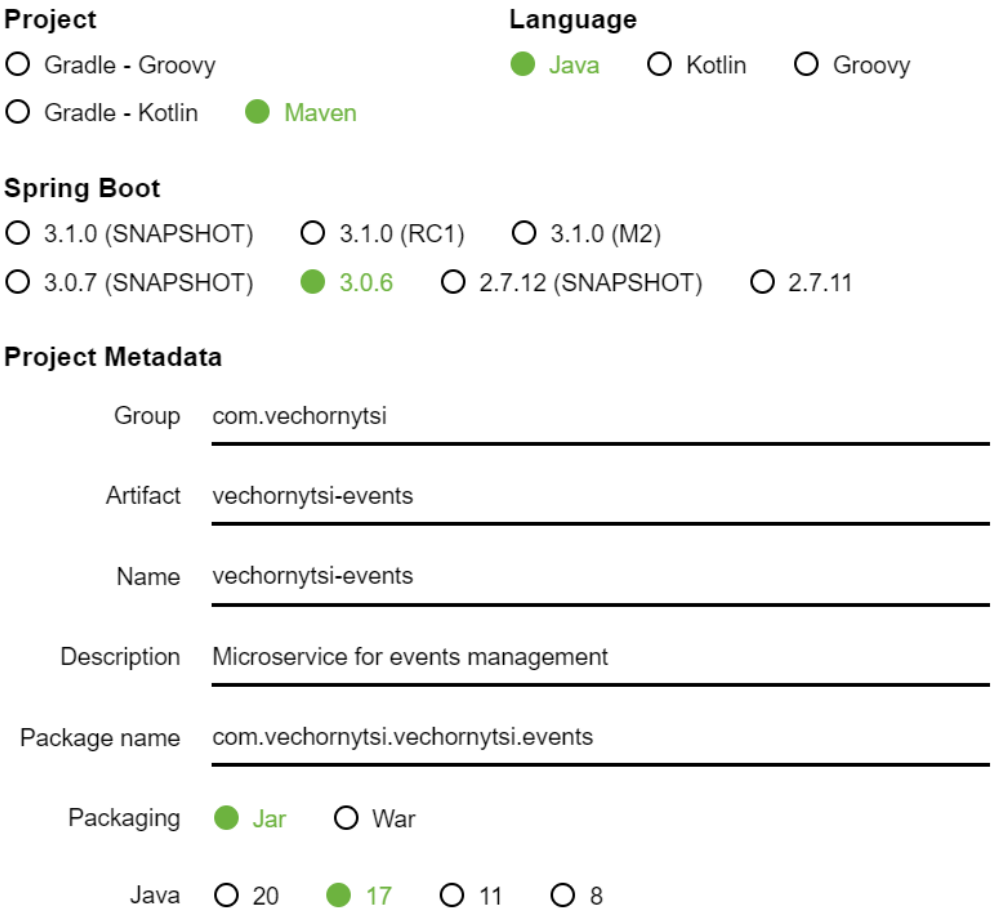


Рисунок 4.1 – Дані для створення проєкту

Після того, як усі дані були введені, потрібно натиснути кнопку «Generate» та зберегти архів із новим проєктом на комп’ютері. Далі потрібно розпакувати архів та запустити IntelliJ IDEA. Коли IDE запуститься, потрібно натиснути кнопку «Open», що знаходиться в правому верхньому кутку і обрати папку із новостворeним проєктом і дочекатись поки проєкт буде завантажено.

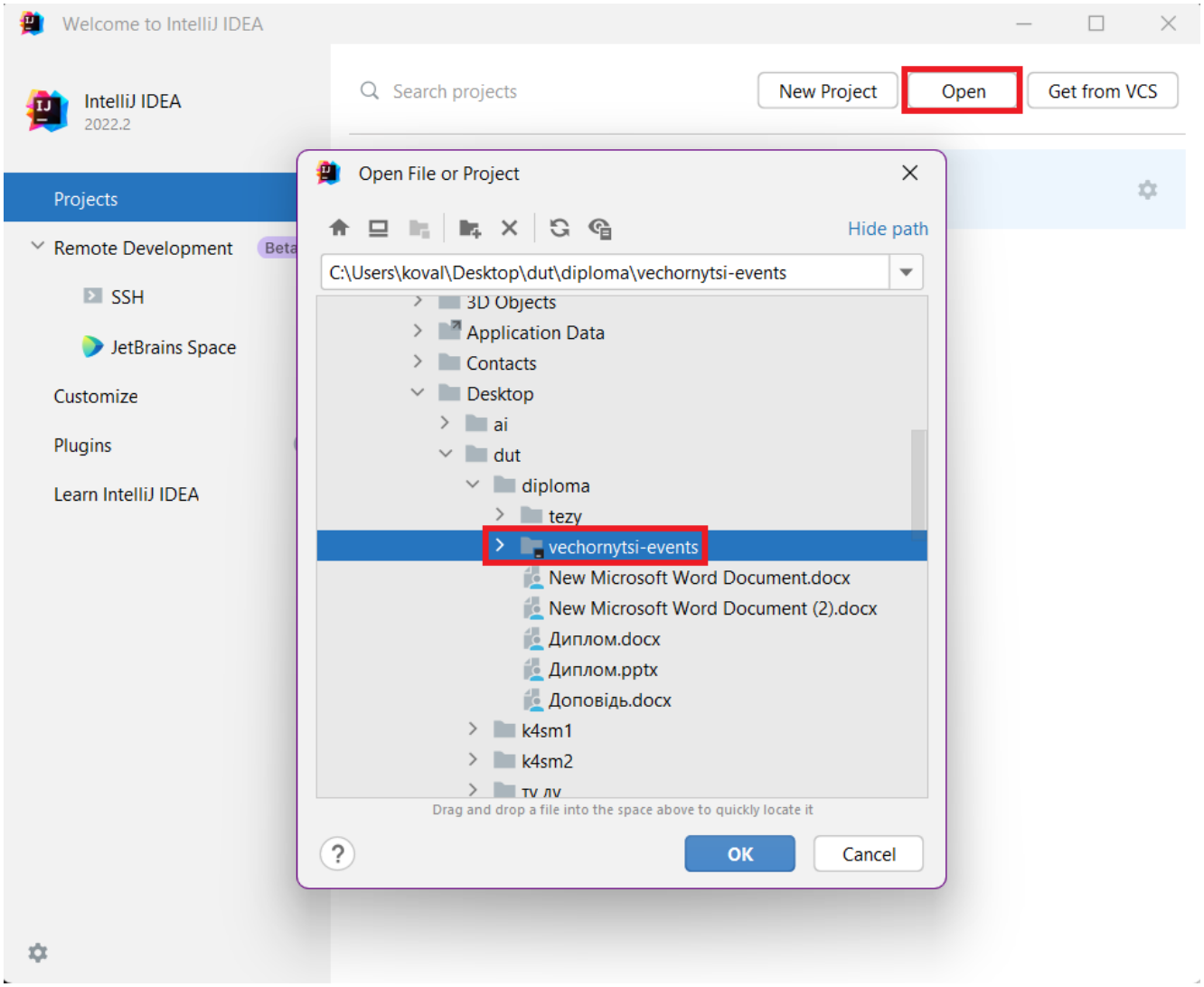


Рисунок 4.2 – Запуск проєкту в IDE

Наступним кроком буде створення структури проєкту. Для цього потрібно обрати файл *./pom.xml* та відредагувати його, вказавши необхідні модулі, пакування pom і видаливши зайві теги, а саме – dependencies, parent та build.

Тепер варто дозволити IDE створити необхідні модулі. Для цього потрібно поставити курсор на місце назви першого модуля, далі натиснути *Alt+Enter*, із випадаючого списку обрати пункт «Create module with parent». Таким же чином створити другий модуль.

Далі варто видалити зайві папки та файли в модулі *vechornytsi-events*, а саме – *./src* та *./HELP.md*.

Наступним кроком потрібно відредагувати *pom.xml* файли в модулях *vechornytsi-events-api* та *vechornytsi-events-service*. В модулі *vechornytsi-events-api* створено три додаткові модулі – *vechornytsi-events-api-api, vechornytsi-events-api-validation, vechornytsi-events-api-exception*. Теги ті самі, що й в файлі *./pom.xml*, але вказувати groupId та version не потрібно.

Редагуємо *./vechornytsi-events-service/pom.xml*. У parent тег додаємо артифакт spring-boot-starter-parent. Додаємо теги dependencies, build та properties.

Далі створюємо папки для зберігання вихідного основного коду *./vechornytsi-events-service/src/main/java/com/vechornytsi/events/service* та папки для зберігання вихідного коду тестів *./vechornytsi-events-service/src/test/java/com/vechornytsi/events/service*.

На рисунку 4.3 зображено початкову структуру проєкту.

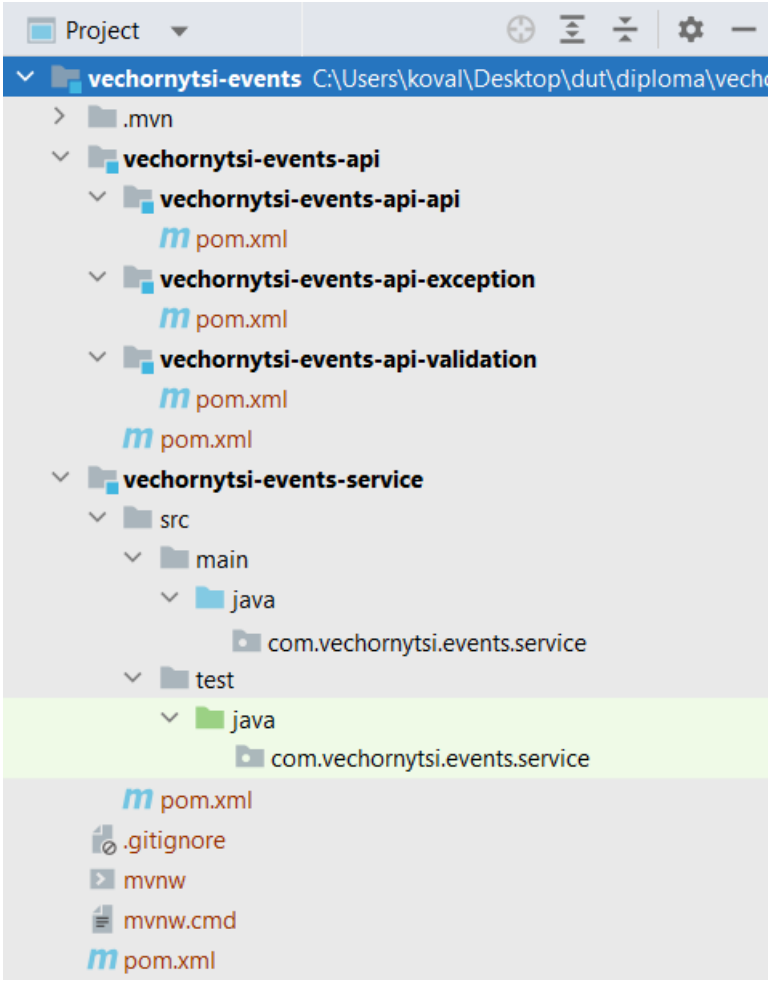


Рисунок 4.3 – Початкова структура проєкту

Тепер можна вперше зробити коміт, аби зберегти дані зміни. Для цього потрібно натиснути сполучення клавіш *Alt+F12*, після чого відкриється вбудована консоль.

Далі, по черзі, необхідно ввести наступні команди:

– git init – створює репозиторій;

– git add -A – індексуємо зміни;

– git commit -m “Init” – вносимо зміни до репозиторію;

– git remote add origin https://YuliaKova@bitbucket.org/yuliakova/vechornytsi.git – додати віддалений репозиторій;

– git push -u origin master – вносимо зміни до віддаленого репозиторію.

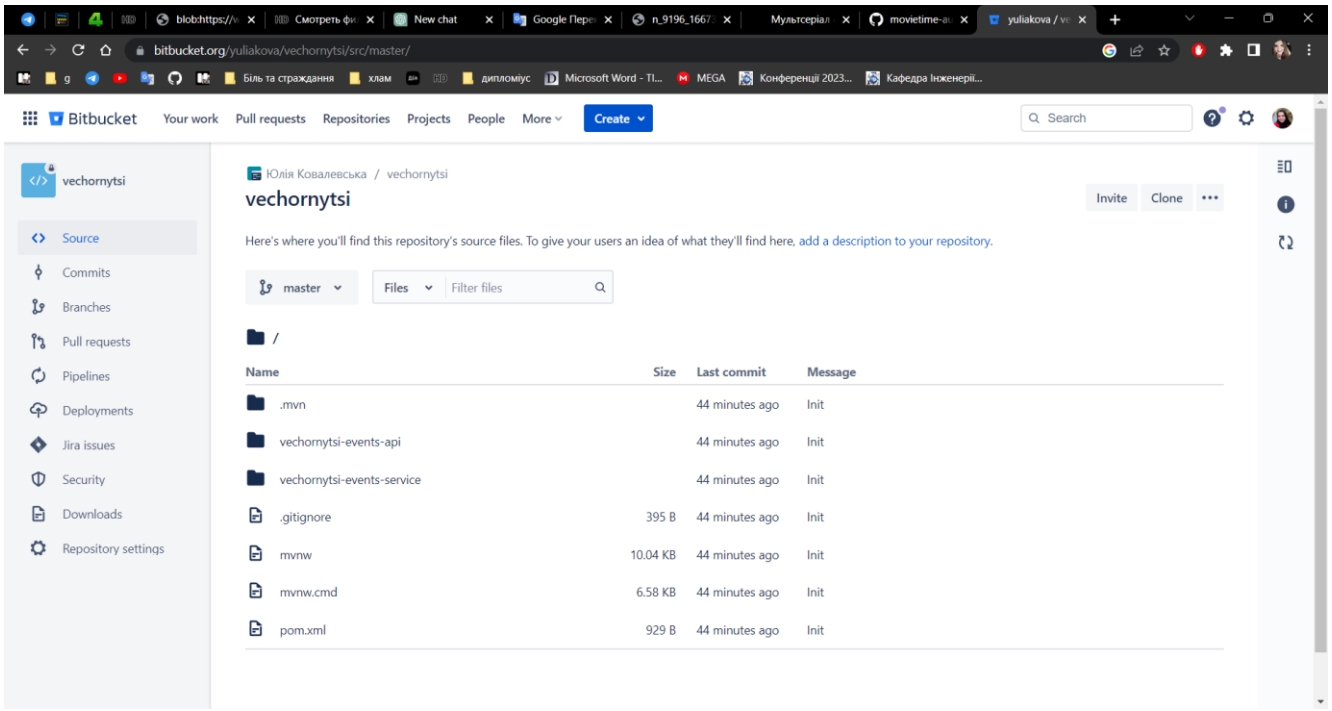


Рисунок 4.4 – Зміни, внесені до віддаленого репозиторію

Можна повертатись до створення структури мікросервісу. Наступним кроком буде створення основних пакетів у модулі *vechornytsi-events-service*, а саме:

– *com.vechornytsi.events.service.controller;*

– *com.vechornytsi.events.service.component.event;*

– *com.vechornytsi.events.service.config.*

Потім потрібно додати класи та інтерфейси:

– *com.vechornytsi.events.service.controller.EventController;*

– *com.vechornytsi.events.service.component.event.Event;*

– *com.vechornytsi.events.service.component.event.EventService;*

– *com.vechornytsi.events.service.component.event.EventServiceImpl;*

– *com.vechornytsi.events.service.component.event.EventRepository;*

– *com.vechornytsi.events.service.component.event.EventMapper;*

– *com.vechotnytsi.events.service.config.ApplicationConfig.*

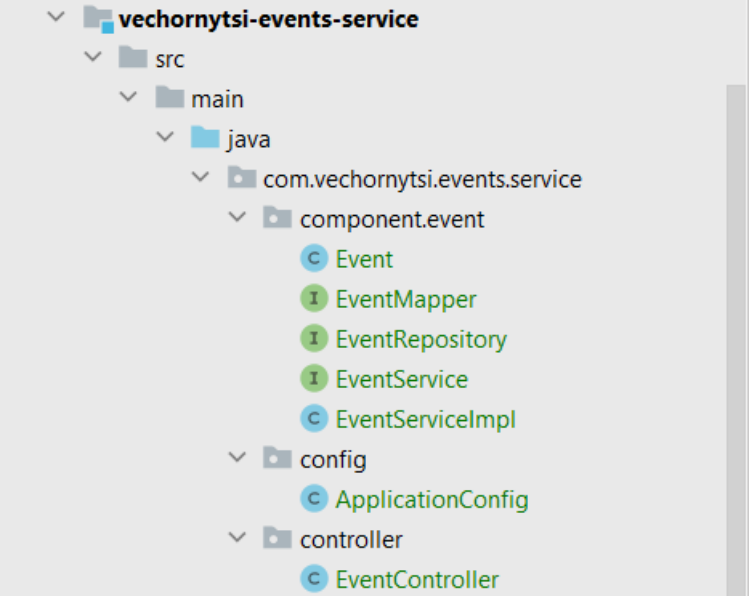


Рисунок 4.5 – Створені класи та інтерфейси у модулі vechornytsi-events-service

Для того аби мати можливість працювати із бібліотеками Spring Boot, потрібно додати необхідні залежності до *./vechornytsi-events-service/pom.xml*. Також варто одразу додати інші бібліотеки, що знадобляться при написанні бізнес-логіки.

У тіло тегу dependencies додаємо наступні залежності:

– spring-boot-starter-web;

– spring-boot-starter-test;

– spring-boot-starter-data-jpa;

– mysql-connector-j;

– liquibase-core;

– lombok;

– mapstruct;

– protobuf-java-util;

– vechornytsi-events-api-api;

– vechornytsi-events-api-exception;

– vechornytsi-events-api-validation.

Для сумісної роботи бібліотек lombok та mapstruct потрібно додати плагін maven-compiler-plugin із відповідною конфігурацією.

Створимо папку ./vechornytsi-events-service/src/main/resources. У ній потрібно створити файл *application.yml*, у якому будуть зберігатись властивості мікросервісу.

У дані файли потрібно додати властивості для підключення до бази даних.

Наступним кроком є створення бази даних. Для цього потрібно запустити MySQL Command Line Client, ввести пароль від серверу і ввести команду:

CREATE DATABASE vechornytsi_events;

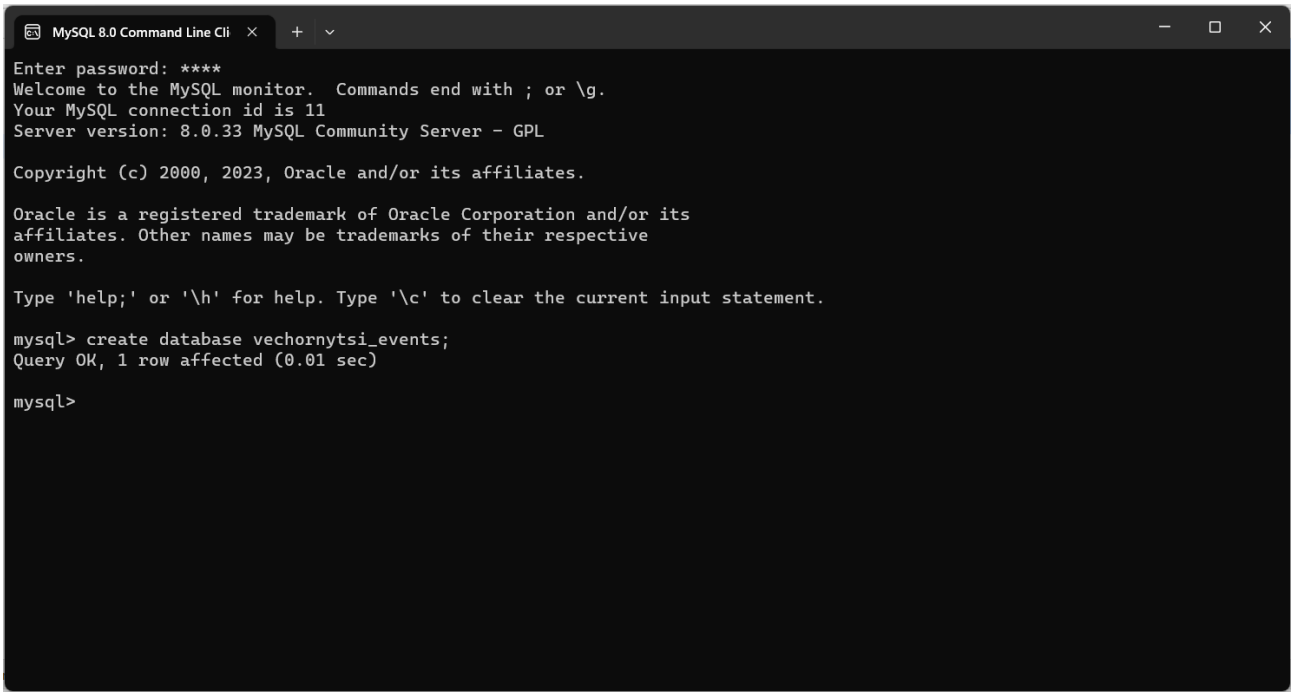


Рисунок 4.6 – Створення бази даних

Коли є існуюча база даних, то можна приступати до її ініціалізації за допомогою бібліотеки Liquibase. Для цього у папці з ресурсами потрібно створити файл *changelog.xml*. У файл додаємо кореневий тег databaseChangeLog, а в нього тег changeSet. Всередині останнього тегу ініціалізуємо таблицю event та її поля відповідно до діаграми зв'язків сутностей. Також потрібно додати тег loadData для того, щоб заповнити таблицю початковими даними із файлу events.csv, що потрібно створити у папці з ресурсами і заповнити тестовими даними.

Далі потрібно додати необхідні поля та анотації до класу Event. Поля будуть створені відповідно до діаграми зв'язків сутностей. Також додаємо анотації із бібліотеки Lombok для того, щоб згенерувати шаблонний код, а саме:

– @Entity – вказує на те, що це сутність;

– @Data – генерує методи для отримання та встановлення полів, а також конструктор, що ініціалізує final поля;

– @NoArgsConstructor – генерує конструктор без параметрів;

– @AllArgsConstructor – генерує конструктор із усіма параметрами;

– @Builder(toBuilder = true) – генерує код, що імплементує шаблон проектування «Будівельник».

Також над полем id потрібно додати анотації @Id, @GeneratedValue(generator = "UUID"), @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator"), що позначають це поле як ідентифікатор сутності і буде генерувати його при зберіганні сутності до бази даних.

UUID – це буквенно-цифровий текстовий ідентифікатор із 36 символів, що є дуже надійним способом ідентифікації, адже шанс згенерувати однакові ідентифікатори дорівнює 1 до 2^{128}.[33]

Для того, аби надсилати запити до бази даних необхідно аби інтерфейс EventRepository наслідувався від інтерфейсу JpaRepository<Event, String> (String в даному випадку це тип ідентифікатора сутності). Також він помічений анотацією @Repository. Методи для взаємодії із базою даних будуть згенеровані автоматично. Більшість із операцій уже наявна в базовому інтерфейсі, а додаткові можна згенерувати лише додавши методи до інтерфейсу. Проте, назви методів повинні

бути описані вірно, аби Spring Boot міг згенерувати їх. Методи, що додані до даного інтерфейсу:

– findAllByNameContains – для пошуку подій за іменем;

– findAllByStartDateBeforeAndEndDateAfter – для пошуку подій за датою;

– findAllByPlaceContains – для пошуку подій за місцем проведення;

– findAllByIsOnlineIsTrue – для пошуку онлайн подій;

– findAllByIsPrivateIsTrue – для пошуку приватних подій;

– findAllByIsActiveIsTrue – для пошуку усіх активних подій.

Переходимо до інших модулів мікросервісу. Спочатку варто завершити модуль із DTO, а саме *vechornytsi-events-api-api*. Перший крок – додати необхідні залежність до *pom.xml* у даному модулі, а саме protobuf-java, та плагін protobuf-maven-plugin, що буде використовуватись для генерування DTO класів. Після цього потрібно створити папку *src/main/proto* та файл *dto.proto*, який міститиме опис усіх класів, що повинні бути згенеровані.

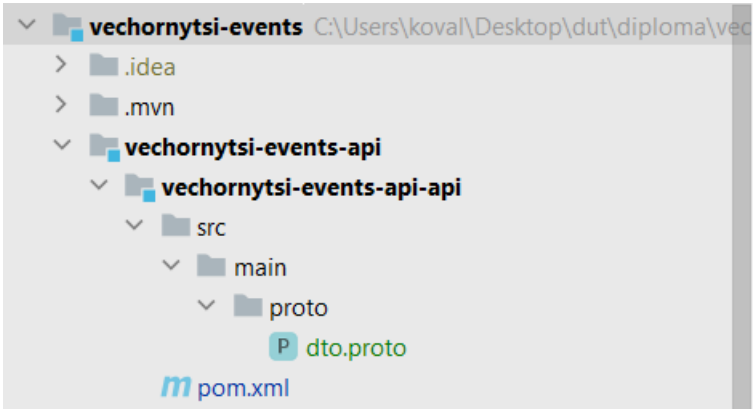


Рисунок 4.7 – Структура модулю vechornytsi-events-api-api

Для генерації потрібно запустити у терміналі команду *mvn compile*. Результат цієї команди зображений на рисунку 4.8.

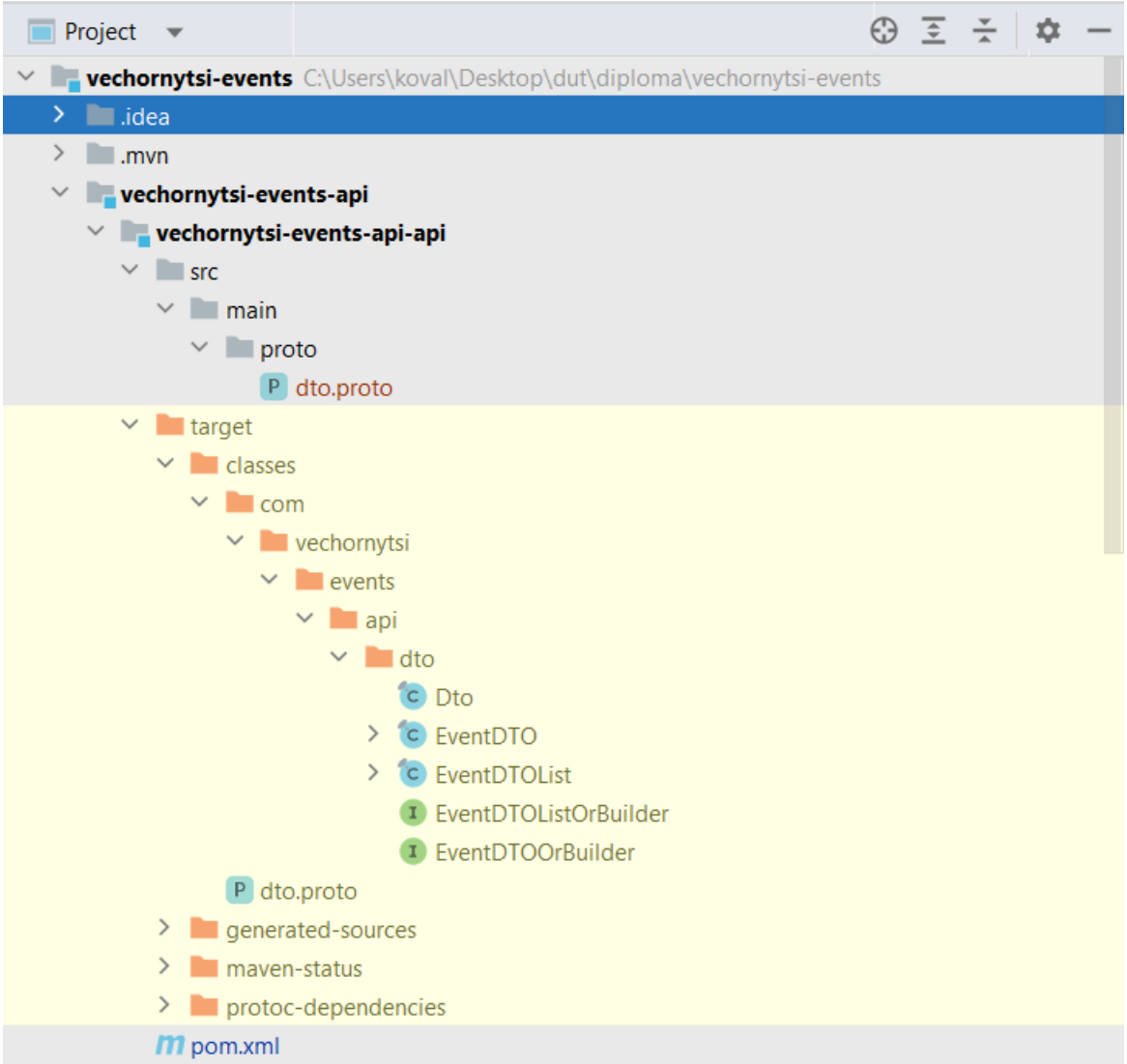


Рисунок 4.8 – Згенеровані класи DTO

Наступним модулем буде *vechornytsi-events-api-exception.* Додаємо залежності:

– spring-boot-starter-web;

– lombok.

У пакеті *com.vechornytsi.events.api.exception* створюємо класи InvalidInputException та RecordNotFoundException, які наслідуються від класу RuntimeException та мають анотацію @StandartException. У пакеті *com.vechornytsi.events.api.exception.handler* створюємо клас GlobalControllerExceptionHandler, що відповідатиме за обробку помилок. Даний клас наслідується від класу ResponseEntityExceptionHandler та має анотацію

@ControllerAdvice. Клас містить у собі два методи для обробки попередньо створених помилок.

Останнім буде опрацьований модуль *vechornytsi-events-api-validation*. Даний модуль залежить від двох попередніх модулів і ці залежності потрібно додати до *pom.xml*. У пакеті *com.vechornytsi.events.api.validation* створюємо клас DTOValidator, що імплементує патерн проєктування «Будівельник» і перевіряє валідність полів DTO об’єкту, і у випадку порушення умови валідності буде викинута помилка InvalidInputException.

Тепер можна повернутись до модулю *vechornytsi-events-service.* Першою справою варто створити ендпоінти у класі EventController. Кожен ендпоінт відповідає окремому прецеденту із діаграми прецедентів. Клас помічений анотацією @RestController, що позначає його як контролер та створює Spring Bean.

Далі потрібно додати конфігурації, що буду створювати Sprign Beans класів, що знаходяться поза межами даного модулю і конвертер для протобафів. Клас помічений анотацією @Configuration, що вказує на те, що клас містить у собі методи, відмічені анотацією @Bean, яка і створює дані Spring Beans.

Spring Beans – це об’єкти, які утворюють основу програми та які створюються, збираються та іншим чином керуються контейнером Spring IoC.[34]

Останнім кроком, що необхідно зробити перед тим, як приступити до бізнес логіки, є створення мапперу, що буде об’єкти класу Event перетворювати у об’єкти класу EventDTO та навпаки. Для цього інтерфейс EventMapper помічаємо анотацією @Mapper бібліотеки Mapstruct. Потім додаємо три методи, що будуть конвертувати об’єкти класів один в одного, а також буде перетворювати список об’єктів класу Event в список об’єктів класу EventDTO.

Коли усі попередні кроки виконані можна переходити до класу EventServiceImpl. Даний клас імплементує інтерфейс EventService та помічений анотацією @Service. На кожен прецедент потрібно створити відповідний метод. Принцип роботи сервісу зводиться до того, що або потрібно валідувати вхідні дані, конвертувати їх в об’єкт класу Event, виконати потрібні операції та зберегти зміни, конвертувати назад у EventDTO та повернути контролеру (останні два кроки

опціональні), або витягнути дані із бази даних, конвертувати у EventDTO та повернути контролеру.

Перед тим, як почати тестування залишився фінальний крок – це створення класу, що містить у собі метод main, який запускає створений мікросервіс. Клас VechornytsiEventsApplication помічений анотацією @SpringBootApplication, що запускає автоконфігурацію та скан компонентів.

Після цього можна знову зберегти усі зміни внесені до мікросервісу за допомогою Git, назвавши коміт «final».

4.2 Тестування

Першим буде тестування за допомогою інструменту Postman. Для цього потрібно запустити програму, авторизуватись та створити нову колекцію, як показано на рисунку 4.9.

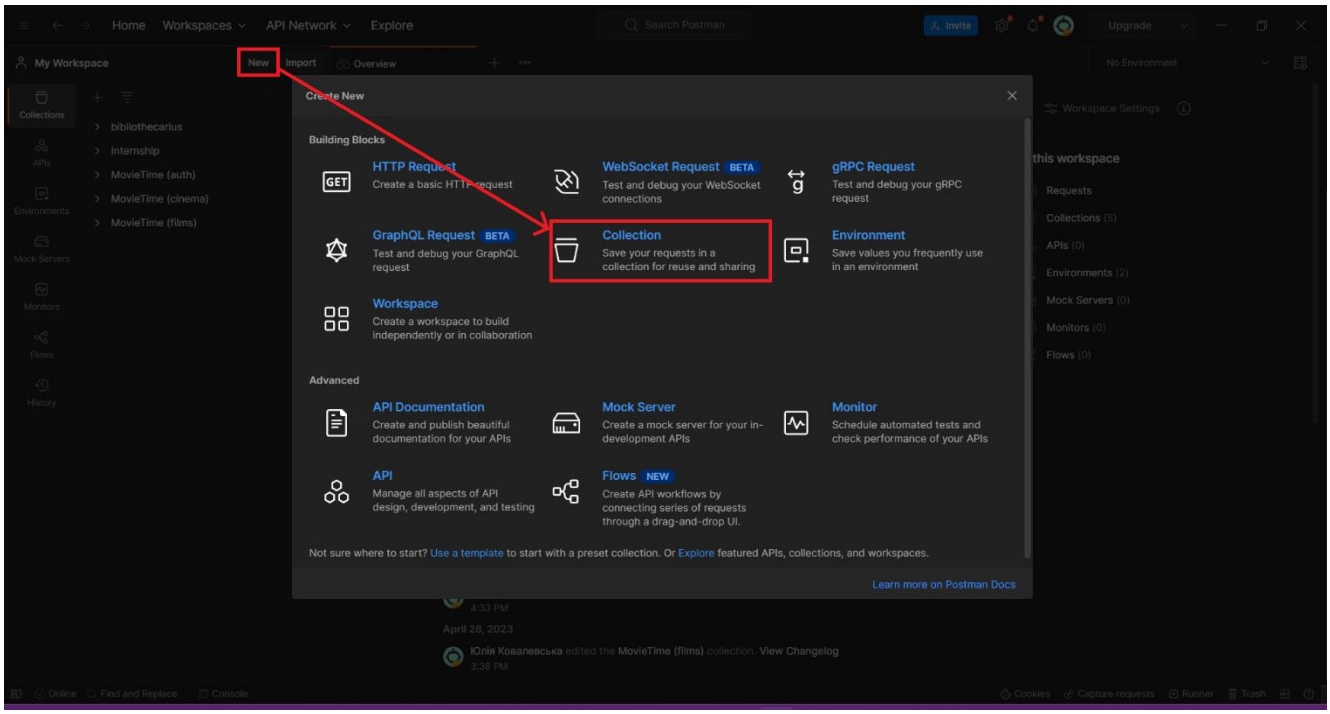


Рисунок 4.9 – Створення колекції

Далі відкриється вкладка колекції, де можна буде змінити її назву на VechornytsiEvents. Після цього можна буде приступити до створення запитів, що будуть надсилатись до створеного мікросервісу. Для цього у боковій вкладці із колекціями знаходимо новостворену колекцію, натискаємо на неї правою кнопкою миші та в меню, що з'явилось, обираємо кнопку «Add request».

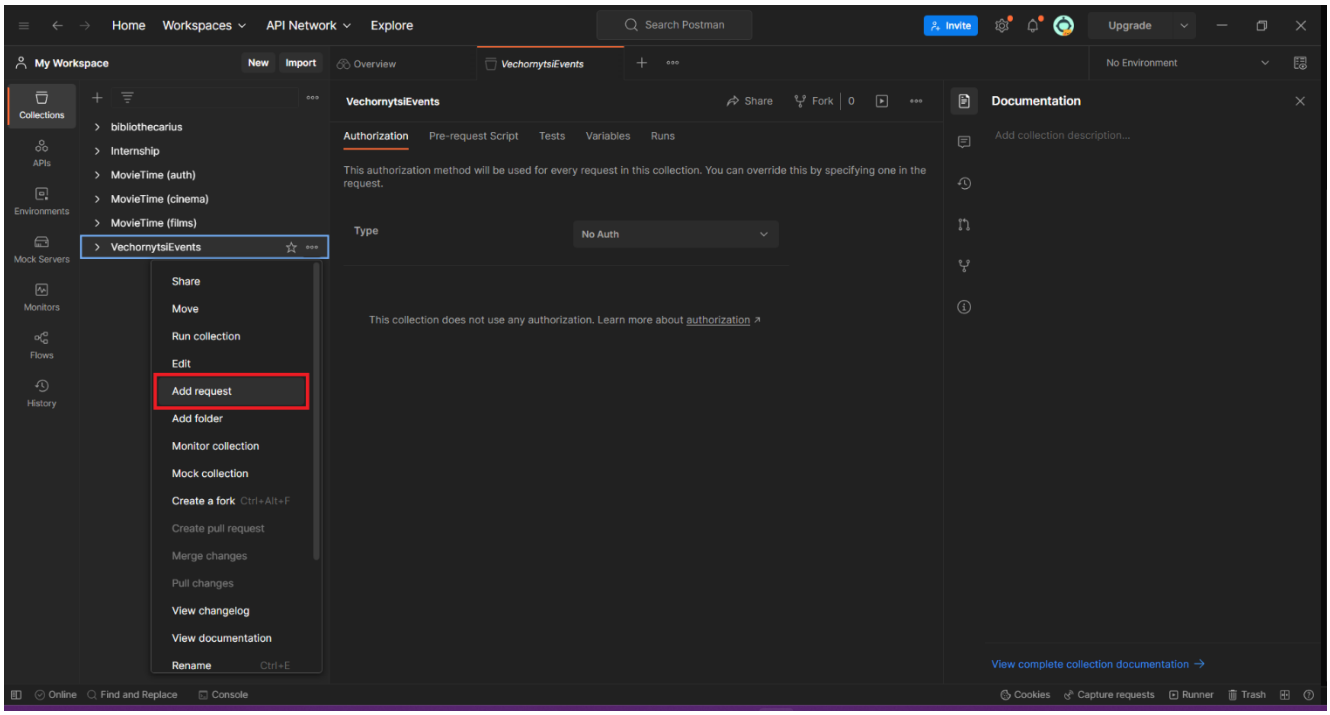


Рисунок 4.10 – Створення запиту

Першим кроком змінюємо назву запиту на «getAllActive». Далі потрібно обрати тип HTTP запиту. Він за замовчуванням GET, тому змінювати нічого не потрібно. Вводимо сам запит: http://localhost:8080/vechornytsi-events/events/.

Для того, аби перевірити чи працює даний запит, потрібно запустити створений мікросервіс. Для цього треба поставити курсор на назву метода main класу VechornytsiEventsApplication та натиснути на клавіші *Ctrl+Shift+F10*. Тепер можна повертатись до Postman та натиснути кнопку «Send».

Після цього отримуємо відповідь із списком усіх активних подій, що були додані до бази даних із файлу *events.csv*.

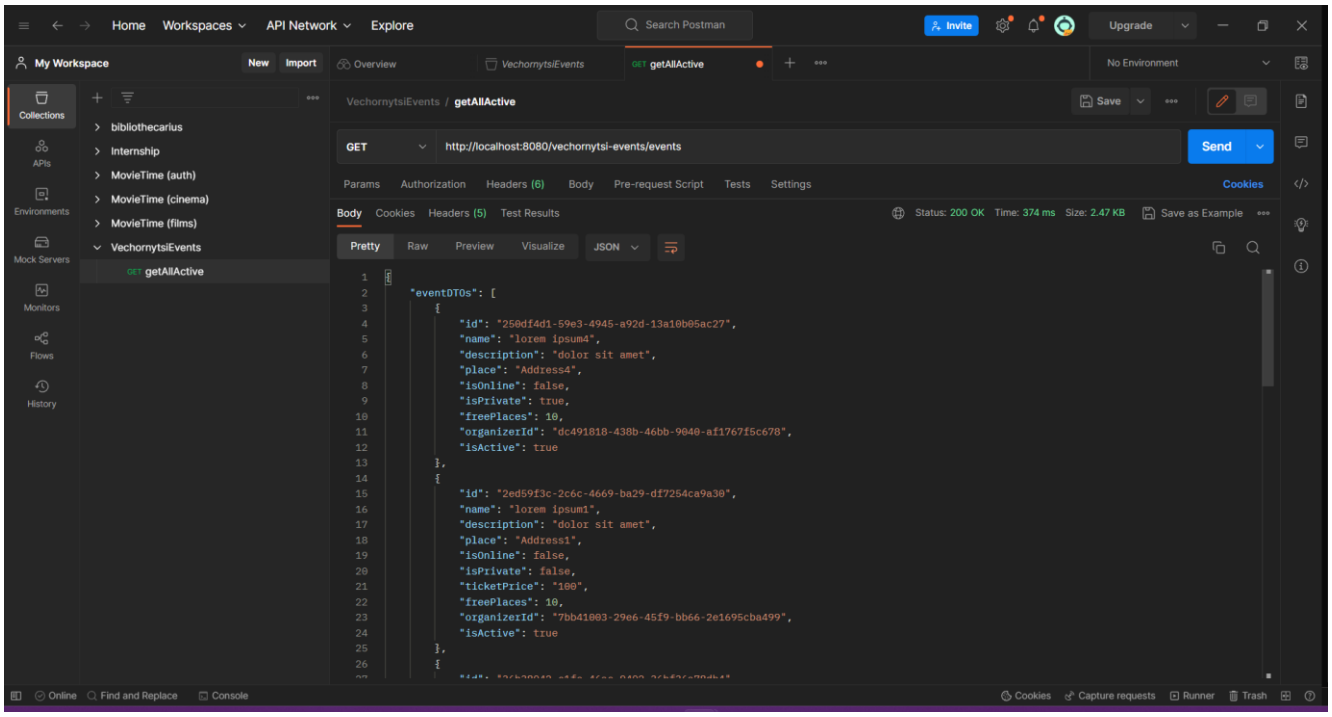


Рисунок 4.11 – Результат надсилання запиту

Після цього можна приступати до створення інших запитів. На рисунку 4.12 зображена готова колекція.

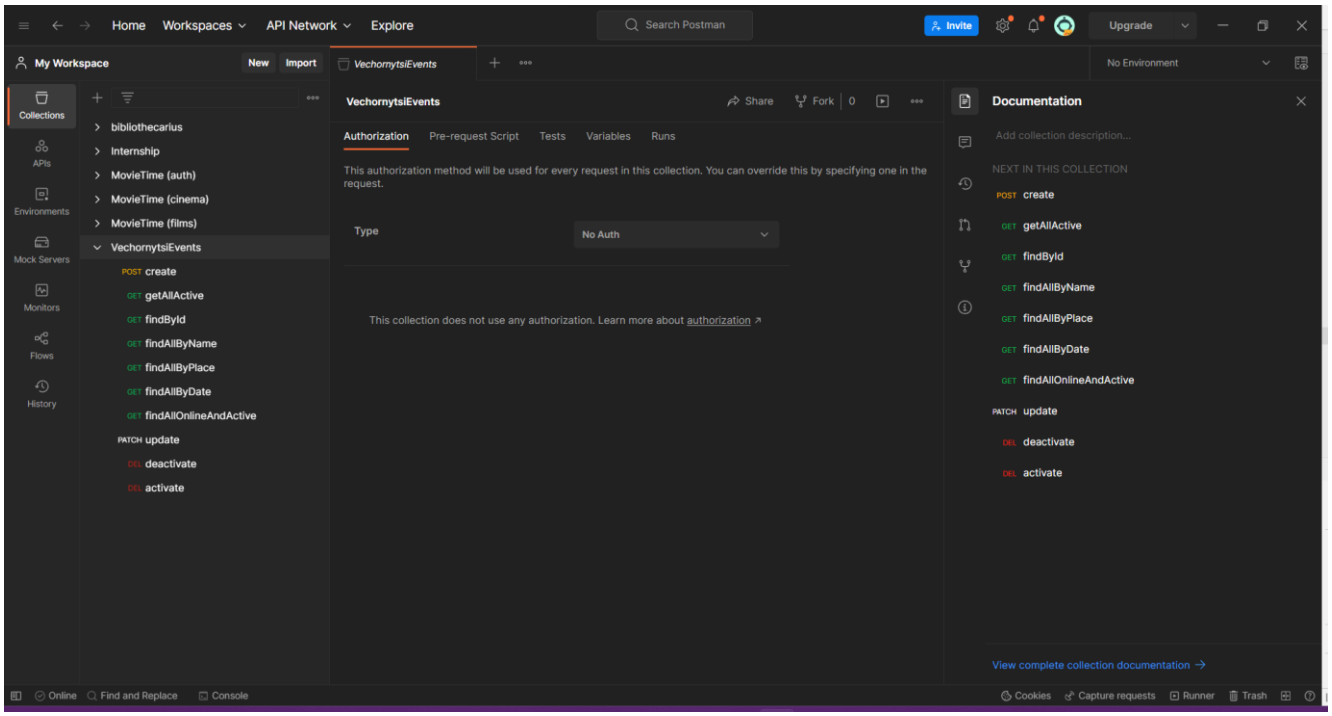


Рисунок 4.12 – Готова колекція

Тепер можна приступати до модульного тестування.

Модульне тестування – це тип тестування програмного забезпечення, який зосереджується на окремих модулях або компонентах програмної системи. Метою модульного тестування є перевірка того, що кожна одиниця програмного забезпечення працює належним чином і відповідає вимогам. Модульні тести призначені для перевірки найменшої можливої одиниці коду, наприклад функції або методу, і тестування її окремо від решти системи. Це дозволяє розробникам швидко виявляти та виправляти будь-які проблеми на ранній стадії розробки, покращуючи загальну якість програмного забезпечення та скорочуючи час, необхідний для подальшого тестування.[35]

Тестуватись буде клас EventServiceImpl. Аби створити тест потрібно поставити курсор на назву класу та натиснути клавіші *Ctrl+Shift+T*. Натискаємо кнопку «Create New Test», називаємо клас EventServiceImplTests та обираємо у списку усі методи, що будуть тестуватись, ставимо галочку напроти поля «setUp/@Before», натискаємо «ОК». Помічаємо новостворений клас анотацією @SprignBootTest.

Додаємо поля та позначаємо їх анотаціями із бібліотеки Mockito, як показано на рисунку 4.13.

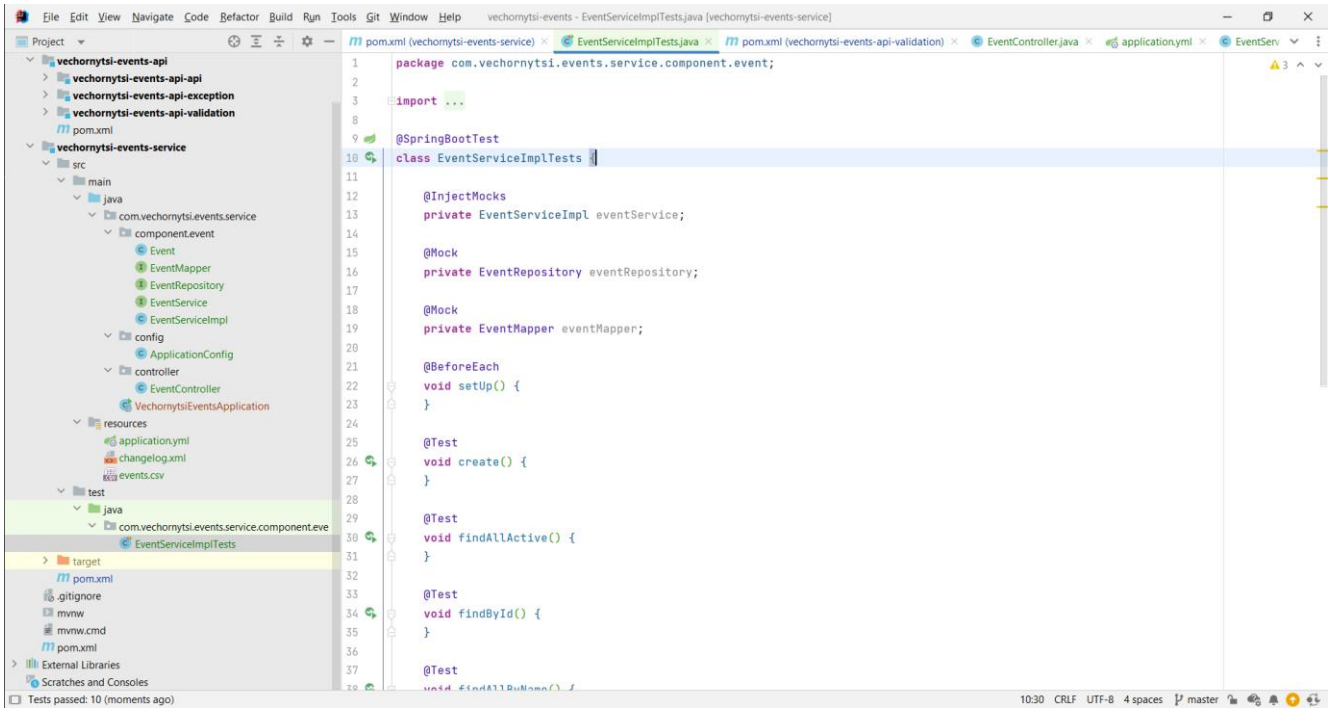


Рисунок 4.13 – Поля у класі EventServiceImplTests

Mockito потрібен для того, аби тести проводились у ізоляції, саме тому ми заміняємо об’єкти EventRepository та EventMapper і вводимо їх у об’єкт класу EventServiceImpl, що буде тестуватись.

Далі потрібно ініціалізувати в методі «setUp» тестові об’єкти класів Event та EventDTO, що будуть використовуватись у тестах. Після цього вже можна приступати до написання самих тестів. Спочатку потрібно визначити за допомогою метода Mockito.when() що будуть робити методи класів EventRepository та EventMapper. Далі викликатиметься метод, який потрібно протестувати, на об’єкті класу EventServiceImpl і результат виклику потрібно порівняти із тим, що очікувалось від даного виклику. Якщо очікуваний та дійсний результати збігаються, то значить, що тест пройдений успішно. Аби це перевірити, потрібно поставити курсор на назву класу EventServiceImplTests та натиснути клавіші *Ctrl+Shift+F10.*

Результат такого виклику можна побачити на рисунку 4.14. Усі тести пройдено успішно.

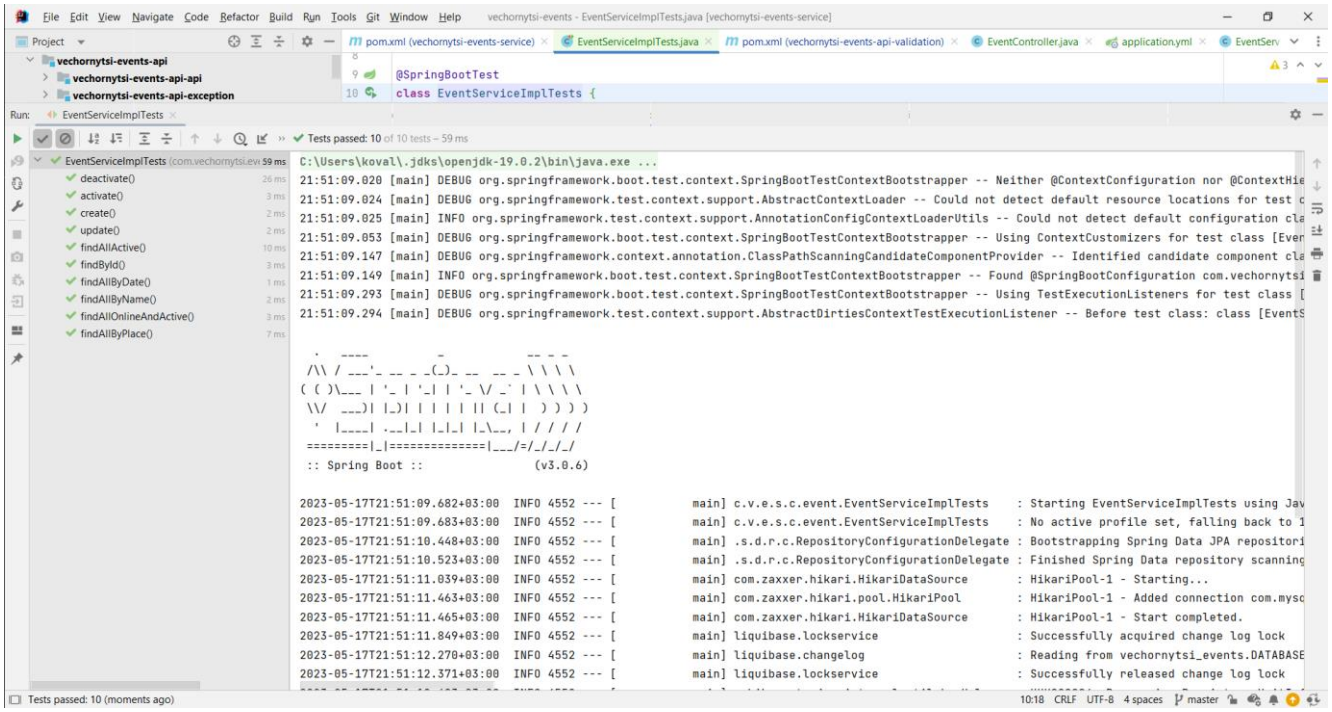


Рисунок 4.14 – Результат виконання тестів

ВИСНОВКИ

У результаті виконання роботи було спрощено процес планування святкових подій за рахунок мікросервісу, написаного мовою програмування Java.

Було виконано такі задачі:

– обрано та досліджено програмні засоби реалізації, що використовувались для розробки мікросервісу: Java, Spring Boot – для реалізації мікросервісу, JUnit, Mockito – для тестування, MySQL, Liquibase – для роботи з базою даних, Google Protocol Buffers – для генерування DTO;

– визначено функціональні вимоги, а саме менеджмент подій та їх пошук за критеріями;

– спроєктовано мікросервіс за допомогою UML діаграм, а саме діаграми компонентів, прецедентів, зв’язків сутностей та класів. Також створено схему вебдодатку, що включає у себе розроблений мікросервіс;

– розроблено мікросервіс для планування святкових подій з використанням технології Spting Boot мовою програмування Java;

– протестовано мікросервіс за допомогою інструменту Postman, а також написано модульні тести з використанням фреймворків Junit та Mockito.

ВИКОРИСТАНІ ДЖЕРЕЛА

1. Wright G., Yasar K. What is social networking and how does it work? – TechTarget Definition. *WhatIs.com*. URL: https://www.techtarget.com/whatis/definition/social-networking.

2. Юрценюк О. Дві протилежні сторони соцмереж – коментар фахівця. URL: https://www.bsmu.edu.ua/blog/6425-dvi-protilezhni-storoni-sotsmerezh-komentar-fahivtsya/.

3. Eventbrite - About Us. *Eventbrite.* URL: https://www.eventbrite.com/about/.

4. Eventbrite Review. *CompareCamp*. URL: https://comparecamp.com/eventbrite-review-pricing-pros-cons-features/.

5. Eventbrite. *TrustRadius*. URL: https://www.trustradius.com/products/eventbrite/reviews?qs=pros-and-cons#overview.

6. Hopin. *TrustRadius*. URL: https://www.trustradius.com/products/hopin/reviews#product-details.

7. Features - IntelliJ IDEA. *JetBrains*. URL: https://www.jetbrains.com/idea/features/.

8. What is Postman & How to Use Postman to Test API. *TestRig*. URL: https://www.testrigtechnologies.com/what-is-postman-and-how-to-use-postman-to-test-api/.

9. What is Java?. *IBM*. URL: https://www.ibm.com/topics/java.

10. What is Java Spring Boot?. *IBM*. URL: https://www.ibm.com/topics/java-spring-boot.

11. Advantages of Spring Boot. *Adservio*. URL: https://www.adservio.fr/post/advantages-of-spring-boot.

12. JPA and Spring Data JPA. *amitph*. URL: https://www.amitph.com/jpa-and-spring-data-jpa/.

13. What is Junit and How it works? An Overview and Its Use Cases. *DevOpsSchool*. URL: https://www.devopsschool.com/blog/what-is-junit-and-how-it-works-an-overview-and-its-use-cases/.

14. Mockito Tutorial. *DigitalOcean*. URL: https://www.digitalocean.com/community/tutorials/mockito-tutorial.

15. Ismail O. What is Liquibase?. *LinkedIn*. URL: https://www.linkedin.com/pulse/what-liquibase-omar-ismail/?trk=articles_directory.

16. Overview. *Protocol Buffers Documentation*. URL: https://protobuf.dev/overview/.

17. Drake M. What is MySQL?. *DigitalOcean*. URL: https://www.digitalocean.com/community/tutorials/what-is-mysql.

18. What is MySQL?. *Oracle*. URL: https://www.oracle.com/mysql/what-is-mysql/.

19. Jira. *ProductPlan*. URL: https://www.productplan.com/glossary/jira/.

20. Confluence: A Brief Overview. *Atlassian*. URL: https://www.atlassian.com/software/confluence/guides/get-started/confluence-overview#about-confluence.

21. What is Git?. *AWS*. URL: https://aws.amazon.com/devops/source-control/git/.

22. Bitbucket: What is Bitbucket?. *Atlassian Documentation*. URL: https://confluence.atlassian.com/confeval/development-tools-evaluator-resources/bitbucket/bitbucket-what-is-bitbucket.

23. What is Component Diagram?. *Visual Paradigm*. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/.

24. What are microservices?. *microservices.io*. URL: https://microservices.io/.

25. Microservice architecture style. *Microsoft Learn*. URL: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices.

26. Hays N. What is a restful API, how it works, advantages, and examples | mailgun. *Mailgun*. URL: https://www.mailgun.com/blog/it-and-engineering/restful-api/.

27. Patni S. Pro RESTful APIs: Design, Build and Integrate with REST, JSON, XML and JAX-RS. Apress, 2017. 126 с.

28. Що таке функціональні вимоги: приклади, визначення, повний посібник. *Visure Solutions*. URL: https://visuresolutions.com/uk/blog/functional-requirements/.

29. What is Use Case Diagram?. *Visual Paradigm*. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/.

30. Що таке нефункціональні вимоги: приклади, визначення, повний посібник. *Visure Solutions*. URL: https://visuresolutions.com/uk/blog/non-functional-requirements/.

31. What is Entity Relationship Diagram (ERD)?. *Visual Paradigm*. URL: https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/.

32. What is Class Diagram?. *Visual Paradigm*. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/.

33. What is a UUID, and why should you care?. *Cockroach Labs*. URL: https://www.cockroachlabs.com/blog/what-is-a-uuid/.

34. What Is a Spring Bean?. *Baeldung*. URL: https://www.baeldung.com/spring-bean.

35. Unit Testing | Software Testing. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/unit-testing-software-testing/.

ДОДАТОК А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«Розробка мікросервісу для планування святкових подій з використанням технології Spring Boot, мовою Java»

Виконала студентка 4 курсу
Групи ПД-41
Ковалевська Юлія Сергіївна
Керівник роботи
К.т.н, доцент Негоденко Олена Василівна

Київ – 2023

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** - спрощення процесу планування святкових подій за рахунок мікросервісу для планування святкових подій, написаний мовою програмування Java.
- **Об’єкт дослідження –** процес планування святкових подій.
- **Предмет дослідження –** програмне забезпечення для планування святкових подій.

2

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Проаналізувати предметну область та дослідити існуючі аналоги.
2. Дослідити програмні засоби реалізації.
3. Спроєктувати мікросервіс за допомогою UML діаграм та визначити функціональні вимоги.
4. Розробити мікросервіс для планування святкових подій з використанням технології Spring Boot, мовою Java.
5. Протестувати розроблений мікросервіс за допомогою інструменту Postman та фреймворків JUnit та Mockito.

3

АНАЛІЗ АНАЛОГІВ

Eventbrite.com

Переваги:	Недоліки:
– легко використовувати; – є мобільні додатки та сайт; – інтеграції з іншими системами; – безкоштовно для безкоштовних подій; – має стандартні шаблони для подій.	– мобільний додаток розділений на два окремих додатки для організаторів і відвідувачів; – складна для використання резервація місць; – великий податок на дорогі квитки; – деякі інтеграції з іншими системами погано протестовані; – не вистачає демонстрації усього функціоналу, який часто є неочевидним.

Hopin

Переваги:	Недоліки:
– легко використовувати; – є мобільний додаток та сайт; – інтеграції з іншими системами; – чати; – події проводяться прямо в додатку; – якісна технічна підтримка; – є спільнота.	– доступно мало безкоштовних функцій; – високі ціни; – не можна створювати повторювані події; – низька якість трансляцій; – не вистачає більш деталізованих інструкцій з використання.

4

ВИМОГИ ДО ДОДАТКУ

Функціональні вимоги:

- Створення онлайн/офлайн подій.
- Скасування подій.
- Відновлення скасованих подій.
- Оновлення інформації про події.
- Пошук події за ID, ім'ям, місцем проведення та датою, а також пошук подій, що проводяться онлайн та повний список активних подій.

Нефункціональні вимоги:

- Масштабованість: мікросервіс повинен мати можливість масштабування шляхом додавання нових мікросервісів до вебдодатку.
- Портативність: мікросервіс повинен працювати незалежно від платформи.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

IntelliJ IDEA

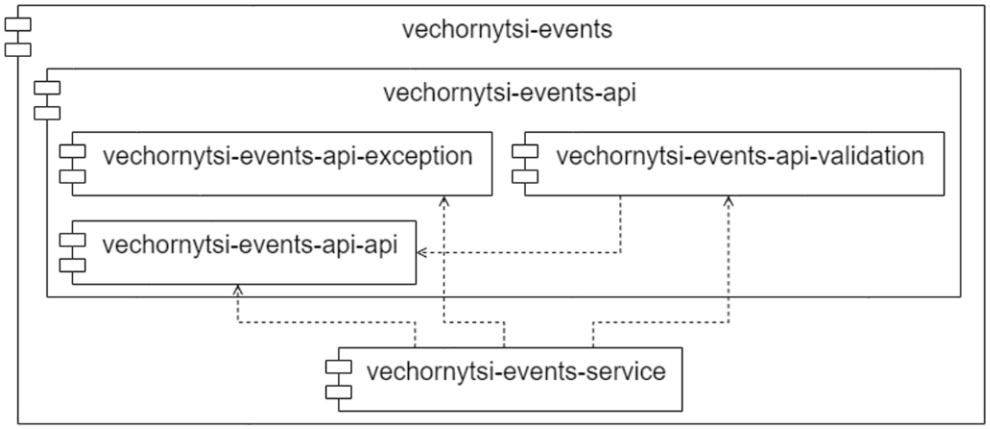
Postman

6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

7

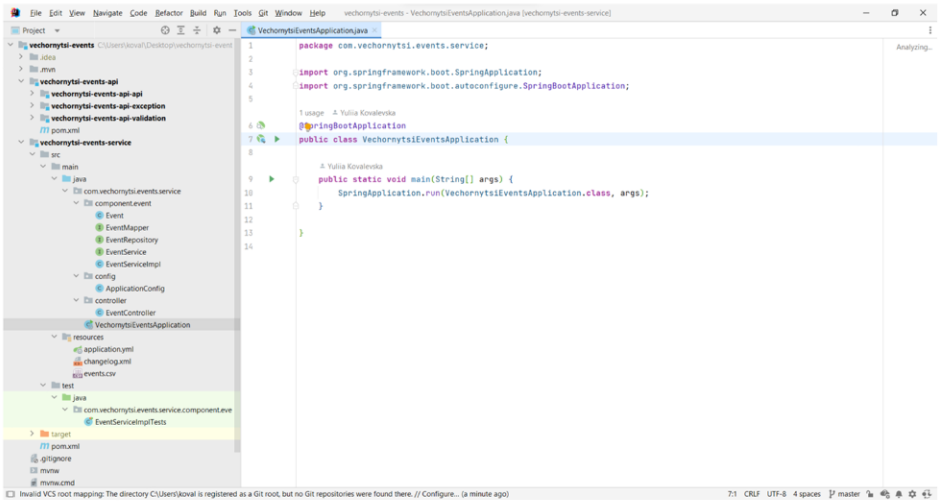
ДІАГРАМА КОМПОНЕНТІВ



8

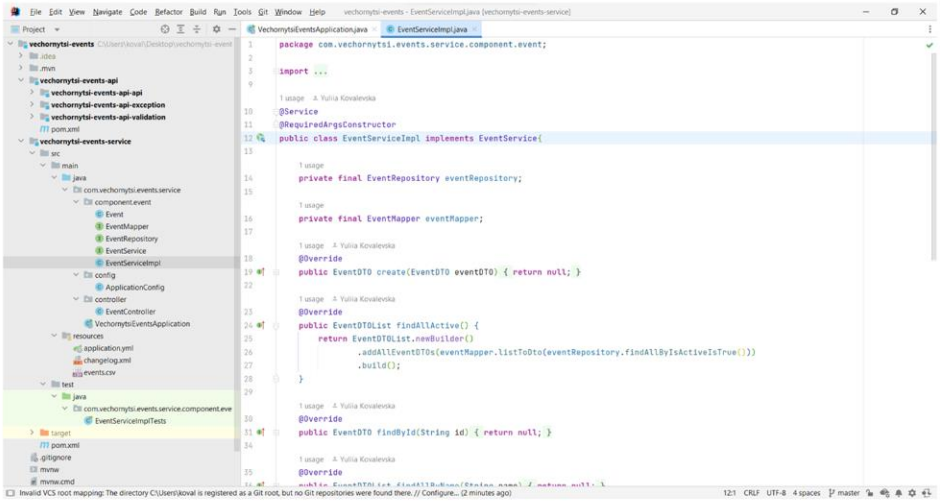
ДІАГРАМА КЛАСІВ

ЕКРАННІ ФОРМИ МІКРОСЕРВІСУ VECHORNYTSI



Клас VechornytsiEventsApplication

ЕКРАННІ ФОРМИ МІКРОСЕРВІСУ VECHORNYTSI



Клас EventServiceImpl

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Ковалевська Ю.С. Актуальність використання REST архітектури для розробки web-сервісів : Матеріали IV всеукраїнської науково-технічної конференції «Сучасний стан та перспективи розвитку IoT». Збірник тез. – 07.04.2023, ДУТ, м. Київ. – С. 15
2. Ковалевська Ю.С. Актуальність використання мікросервісної архітектури для розробки web-сервісів : Матеріали науково-технічної конференції «Застосування програмного забезпечення в інфокомунікаційних технологіях». Збірник тез. – 20.04.2023, ДУТ, м. Київ. – С. 137

ВИСНОВКИ

1. Обрано та досліджено програмні засоби реалізації, що використовувались для розробки мікросервісу: Java, Spring Boot – для реалізації мікросервісу, JUnit, Mockito – для тестування, MySQL, Liquibase – для роботи з базою даних, Google Protocol Buffers – для генерування DTO.
2. Визначено функціональні вимоги, а саме менеджмент подій та їх пошук за критеріями.
3. Спроєктовано мікросервіс за допомогою UML діаграм, а саме діаграми компонентів, прецедентів, зв'язків сутностей та класів. Також створено схему вебдодатку, що включає у себе розроблений мікросервіс.
4. Розроблено мікросервіс для планування святкових подій з використанням технології Spting Boot мовою програмування Java.
5. Протестовано мікросервіс за допомогою інструменту Postman, а також написано модульні тести з використанням фреймворків Junit та Mockito.

13

ДЕМОНСТРАЦІЯ РОБОТИ МІКРОСЕРВІСУ

14

ДЯКУЮ ЗА УВАГУ!

ДОДАТОК Б

Публікації

1. Ковалевська Ю.С. Актуальність використання REST архітектури для розробки web-сервісів : Матеріали IV всеукраїнської науково-технічної конференції «Сучасний стан та перспективи розвитку IoT». Збірник тез. – 07.04.2023, ДУТ, м. Київ. – С. 15

2. Ковалевська Ю.С. Актуальність використання мікросервісної архітектури для розробки web-сервісів : Матеріали науково-технічної конференції «Застосування програмного забезпечення в інфокомунікаційних технологіях». Збірник тез. – 20.04.2023, ДУТ, м. Київ. – С. 137

ДОДАТОК В

Вихідний код мікросервісу

```
package com.vechornytsi.events.service.component.event;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.hibernate.annotations.GenericGenerator;

import java.math.BigDecimal;
import java.time.LocalDate;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder(toBuilder = true)
public class Event {

    @Id
    @GeneratedValue(generator = "UUID")
    @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")
    private String id;

    private String name;

    private String description;

    private LocalDate startDate;

    private LocalDate endDate;

    private String place;

    private Boolean isOnline;
```

```
    private Boolean isPrivate;

    private BigDecimal ticketPrice;

    private Integer freePlaces;

    private String organizerId;

    private Boolean isActive;

}
```

```
package com.vechornytsi.events.service.component.event;

import com.vechornytsi.events.api.dto.EventDTO;
import org.mapstruct.Mapper;
import org.mapstruct.Mapping;

import java.util.List;

@Mapper
public interface EventMapper {

    @Mapping(target = "id", ignore = true)
    @Mapping(target = "startDate", ignore = true)
    @Mapping(target = "endDate", ignore = true)
    Event fromDto(EventDTO employeeDTO);

    @Mapping(target = "startDate", ignore = true)
    @Mapping(target = "endDate", ignore = true)
    EventDTO toDto(Event employee);

    @Mapping(target = "startDate", ignore = true)
    @Mapping(target = "endDate", ignore = true)
    List<EventDTO> listToDto(List<Event> employees);

}
```

```
package com.vechornytsi.events.service.component.event;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.time.LocalDate;
```

```
import java.util.List;

@Repository
public interface EventRepository extends JpaRepository<Event, String> {

    List<Event> findAllByNameContains(String name);

    List<Event> findAllByStartDateBeforeAndEndDateAfter(LocalDate date, LocalDate
date1);

    List<Event> findAllByPlaceContains(String place);

    List<Event> findAllByIsOnlineIsTrue();

    List<Event> findAllByIsPrivateIsTrue();

    List<Event> findAllByIsActiveIsTrue();

}

package com.vechornytsi.events.service.component.event;

import com.vechornytsi.events.api.dto.EventDTO;
import com.vechornytsi.events.api.dto.EventDTOList;

import java.time.LocalDate;

public interface EventService {

    EventDTO create(EventDTO eventDTO);

    EventDTOList findAllActive();

    EventDTO findById(String id);

    EventDTOList findAllByName(String name);

    EventDTOList findAllByPlace(String place);

    EventDTOList findAllByDate(LocalDate date);

    EventDTOList findAllByIsOnlineIsTrueAndIsActiveIsTrue ();

    EventDTO update(EventDTO eventDTO);
```

```
    void deactivate(String id);

    void activate(String id);

}

package com.vechornytsi.events.service.component.event;

import com.vechornytsi.events.api.dto.EventDTO;
import com.vechornytsi.events.api.dto.EventDTOList;
import com.vechornytsi.events.api.exception.RecordNotFoundException;
import com.vechornytsi.events.api.validation.DTOValidator;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.math.BigDecimal;
import java.time.Instant;
import java.time.LocalDate;
import java.time.ZoneId;

@Service
@RequiredArgsConstructor
public class EventServiceImpl implements EventService {

    private final EventRepository eventRepository;

    private final EventMapper eventMapper;

    @Override
    public EventDTO create(EventDTO eventDTO) {
        DTOValidator.newValidator()
                .checkName(eventDTO.getName())
                .checkDescription(eventDTO.getDescription())

.checkDates(LocalDate.ofInstant(Instant.ofEpochSecond(eventDTO.getStartDate()),
ZoneId.systemDefault()),
                        LocalDate.ofInstant(Instant.ofEpochSecond(eventDTO.getEndDate()),
ZoneId.systemDefault()))
                .checkPlace(eventDTO.getPlace())
                .checkTicketPrice(new BigDecimal(eventDTO.getTicketPrice()))
                .checkFreePlaces(eventDTO.getFreePlaces())
                .validate();
        Event toSave = eventMapper.fromDto(eventDTO);
```

```
toSave.setStartDate(LocalDate.ofInstant(Instant.ofEpochSecond(eventDTO.getStartDat
e()), ZoneId.systemDefault()));

toSave.setEndDate(LocalDate.ofInstant(Instant.ofEpochSecond(eventDTO.getEndDate(
)), ZoneId.systemDefault()));
        toSave.setIsOnline(eventDTO.getIsOnline());
        toSave.setIsPrivate(eventDTO.getIsPrivate());
        toSave.setIsActive(eventDTO.getIsActive());

        return eventMapper.toDto(eventRepository.save(toSave)).toBuilder()
                .setStartDate(eventDTO.getStartDate())
                .setEndDate(eventDTO.getEndDate())
                .build();
    }

    @Override
    public EventDTOList findAllActive() {
        return EventDTOList.newBuilder()

.addAllEventDTOs(eventMapper.listToDto(eventRepository.findAllByIsActiveIsTrue()
))
                .build();
    }

    @Override
    public EventDTO findById(String id) {
        return eventMapper.toDto(findByIdInternal(id));
    }

    @Override
    public EventDTOList findAllByName(String name) {
        return EventDTOList.newBuilder()

.addAllEventDTOs(eventMapper.listToDto(eventRepository.findAllByNameContains(n
ame)))
                .build();
    }

    @Override
    public EventDTOList findAllByPlace(String place) {
        return EventDTOList.newBuilder()
```

```
.addAllEventDTOs(eventMapper.listToDto(eventRepository.findAllByPlaceContains(p
lace)))
            .build();
    }

    @Override
    public EventDTOList findAllByDate(LocalDate date) {
        return EventDTOList.newBuilder()
                .addAllEventDTOs(

eventMapper.listToDto(eventRepository.findAllByStartDateBeforeAndEndDateAfter(d
ate, date)))
                .build();
    }

    @Override
    public EventDTOList findAllOnlineAndActive() {
        return EventDTOList.newBuilder()

.addAllEventDTOs(eventMapper.listToDto(eventRepository.findAllByIsOnlineIsTrueA
ndIsActiveIsTrue()))
                .build();
    }

    @Override
    public EventDTO update(EventDTO eventDTO) {
        Event event = findByIdInternal(eventDTO.getId());

        String name = eventDTO.getName();
        String description = eventDTO.getDescription();
        long startDate = eventDTO.getStartDate();
        long endDate = eventDTO.getEndDate();
        String ticketPrice = eventDTO.getTicketPrice();
        int freePlaces = eventDTO.getFreePlaces();

        if (!name.isEmpty() || !description.isEmpty() || startDate == 0 || endDate == 0
                || !ticketPrice.isEmpty() || freePlaces == 0) {
            event = event.toBuilder()
                    .id(event.getId())
                    .name(name.isEmpty() ? event.getName() : name)
                    .description(description.isEmpty() ? event.getDescription() : description)
                    .startDate(startDate == 0 ? event.getStartDate() :
```

```
LocalDate.ofInstant(Instant.ofEpochSecond(eventDTO.getStartDate()),
ZoneId.systemDefault()))
                .endDate(endDate == 0 ? event.getEndDate() :
                        LocalDate.ofInstant(Instant.ofEpochSecond(eventDTO.getEndDate()),
ZoneId.systemDefault()))
                .ticketPrice(ticketPrice.isEmpty() ? event.getTicketPrice() : new
BigDecimal(ticketPrice))
                .freePlaces(freePlaces == 0 ? event.getFreePlaces() : freePlaces)
                .build();

            DTOValidator.newValidator()
                .checkName(event.getName())
                .checkDescription(event.getDescription())
                .checkDates(event.getStartDate(), event.getEndDate())
                .checkPlace(event.getPlace())
                .checkTicketPrice(event.getTicketPrice())
                .checkFreePlaces(event.getFreePlaces())
                .validate();

            return eventMapper.toDto(eventRepository.save(event));
        }

        return eventMapper.toDto(event);
    }

    @Override
    public void deactivate(String id) {
        Event event = findByIdInternal(id);
        event.setIsActive(false);
        eventRepository.save(event);
    }

    @Override
    public void activate(String id) {
        Event event = findByIdInternal(id);
        event.setIsActive(true);
        eventRepository.save(event);
    }

    private Event findByIdInternal(String id) {
        return eventRepository
                .findById(id)
```

```
            .orElseThrow(() -> new RecordNotFoundException("Event{id=%s} not
found".formatted(id)));
    }

}

package com.vechornytsi.events.service.config;

import
com.vechornytsi.events.api.exception.handler.GlobalControllerExceptionHandler;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.converter.protobuf.ProtobufHttpMessageConverter;
import
org.springframework.http.converter.protobuf.ProtobufJsonFormatHttpMessageConverte
r;

@Configuration
public class ApplicationConfig {

    @Bean
    public GlobalControllerExceptionHandler exceptionHandler() {
        return new GlobalControllerExceptionHandler();
    }

    @Bean
    public ProtobufHttpMessageConverter protobufHttpMessageConverter() {
        return new ProtobufJsonFormatHttpMessageConverter();
    }

}

package com.vechornytsi.events.service.controller;

import com.vechornytsi.events.api.dto.EventDTO;
import com.vechornytsi.events.api.dto.EventDTOList;
import com.vechornytsi.events.service.component.event.EventService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PatchMapping;
```

```
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

import java.time.Instant;
import java.time.LocalDate;
import java.time.ZoneId;

@RestController
@RequestMapping(value = "/events", produces = MediaType.APPLICATION_JSON_VALUE)
@RequiredArgsConstructor
public class EventController {

    private final EventService eventService;

    @PostMapping
    public ResponseEntity<EventDTO> create(@RequestBody EventDTO eventDTO) {
        return ResponseEntity.ok(eventService.create(eventDTO));
    }

    @GetMapping
    public ResponseEntity<EventDTOList> findAllActive() {
        return ResponseEntity.ok(eventService.findAllActive());
    }

    @GetMapping("/{id}")
    public ResponseEntity<EventDTO> findById(@PathVariable String id) {
        return ResponseEntity.ok(eventService.findById(id));
    }

    @GetMapping(params = "name")
    public ResponseEntity<EventDTOList> findAllByName(@RequestParam String name) {
        return ResponseEntity.ok(eventService.findAllByName(name));
    }

    @GetMapping(params = "place")
    public ResponseEntity<EventDTOList> findAllByPlace(@RequestParam String place) {
```

```
        return ResponseEntity.ok(eventService.findAllByPlace(place));
    }

    @GetMapping(params = "date")
    public ResponseEntity<EventDTOList> findAllByDate(@RequestParam Long date) {
        return ResponseEntity.ok(eventService.findAllByDate(
                LocalDate.ofInstant(Instant.ofEpochSecond(date), ZoneId.systemDefault())));
    }

    @GetMapping("/online")
    public ResponseEntity<EventDTOList> findAllOnlineAndActive(){
        return ResponseEntity.ok(eventService.findAllOnlineAndActive());
    }

    @PatchMapping
    public ResponseEntity<EventDTO> update(@RequestBody EventDTO eventDTO){
        return ResponseEntity.ok(eventService.update(eventDTO));
    }

    @DeleteMapping("/deactivate/{id}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void deactivate(@PathVariable String id){
        eventService.deactivate(id);
    }

    @DeleteMapping("/activate/{id}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void activate(@PathVariable String id){
        eventService.activate(id);
    }

}
```

```
package com.vechornytsi.events.service;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class VechornytsiEventsApplication {

    public static void main(String[] args) {
        SpringApplication.run(VechornytsiEventsApplication.class, args);
    }
```

```
}

package com.vechornytsi.events.service.component.event;

import com.vechornytsi.events.api.dto.EventDTO;
import com.vechornytsi.events.api.dto.EventDTOList;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.springframework.boot.test.context.SpringBootTest;

import java.math.BigDecimal;
import java.time.LocalDate;
import java.time.LocalTime;
import java.time.ZoneOffset;
import java.util.Collections;
import java.util.List;
import java.util.Optional;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.ArgumentMatchers.anyList;
import static org.mockito.ArgumentMatchers.anyString;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;

@SpringBootTest
class EventServiceImplTests {

    @InjectMocks
    private EventServiceImpl eventService;

    @Mock
    private EventRepository eventRepository;

    @Mock
    private EventMapper eventMapper;

    private EventDTO eventDTO;

    private Event event;
```

```
    private List<Event> events;

    private EventDTOList eventDTOList;


    @BeforeEach
    void setUp() {
        eventDTO = EventDTO.newBuilder()
                .setId("ID")
                .setName("Name")
                .setDescription("Description")
                .setStartDate(LocalDate.MIN.toEpochSecond(LocalTime.MIDNIGHT,
ZoneOffset.UTC))
                .setEndDate(LocalDate.MAX.toEpochSecond(LocalTime.MIDNIGHT,
ZoneOffset.UTC))
                .setPlace("Place")
                .setTicketPrice("1")
                .setFreePlaces(10)
                .setIsActive(true)
                .setOrganizerId("ID1")
                .setIsPrivate(false)
                .build();

        event = Event.builder()
                .id("ID")
                .name("Name")
                .description("Description")
                .startDate(LocalDate.MIN)
                .endDate(LocalDate.MAX)
                .place("Place")
                .ticketPrice(BigDecimal.ONE)
                .freePlaces(10)
                .isActive(true)
                .organizerId("ID1")
                .isPrivate(false)
                .build();

        events = Collections.nCopies(10, event);

        eventDTOList = EventDTOList.newBuilder()
                .addAllEventDTOs(Collections.nCopies(10, eventDTO))
                .build();
    }
```

```
    @Test
    void create() {
        when(eventMapper.fromDto(any(EventDTO.class))).thenReturn(event);
        when(eventRepository.save(any(Event.class))).thenReturn(event);
        when(eventMapper.toDto(any(Event.class))).thenReturn(eventDTO);

        EventDTO actualEventDTO = eventService.create(eventDTO);

        assertEquals(eventDTO, actualEventDTO);
    }

    @Test
    void findAllActive() {
        when(eventRepository.findAllByIsActiveIsTrue()).thenReturn(events);

when(eventMapper.listToDto(anyList())).thenReturn(eventDTOList.getEventDTOsList
());

        EventDTOList actualEventDTOList = eventService.findAllActive();

        assertEquals(eventDTOList.getEventDTOsList(),
actualEventDTOList.getEventDTOsList());
    }

    @Test
    void findById() {
        when(eventRepository.findById(anyString())).thenReturn(Optional.of(event));
        when(eventMapper.toDto(any(Event.class))).thenReturn(eventDTO);

        EventDTO actualEventDTO = eventService.findById("ID");

        assertEquals(eventDTO, actualEventDTO);
    }

    @Test
    void findAllByName() {
        when(eventRepository.findAllByNameContains(anyString())).thenReturn(events);

when(eventMapper.listToDto(anyList())).thenReturn(eventDTOList.getEventDTOsList
());

        EventDTOList actualEventDTOList = eventService.findAllByName("Name");
```

```
        assertEquals(eventDTOList.getEventDTOsList(),
actualEventDTOList.getEventDTOsList());
    }

    @Test
    void findAllByPlace() {
        when(eventRepository.findAllByPlaceContains(anyString())).thenReturn(events);

when(eventMapper.listToDto(anyList())).thenReturn(eventDTOList.getEventDTOsList
());

        EventDTOList actualEventDTOList = eventService.findAllByPlace("Place");

        assertEquals(eventDTOList.getEventDTOsList(),
actualEventDTOList.getEventDTOsList());
    }

    @Test
    void findAllByDate() {

when(eventRepository.findAllByStartDateBeforeAndEndDateAfter(any(LocalDate.clas
s), any(LocalDate.class)))
                .thenReturn(events);

when(eventMapper.listToDto(anyList())).thenReturn(eventDTOList.getEventDTOsList
());

        EventDTOList                actualEventDTOList                =
eventService.findAllByDate(LocalDate.now());

        assertEquals(eventDTOList.getEventDTOsList(),
actualEventDTOList.getEventDTOsList());
    }

    @Test
    void findAllOnlineAndActive() {

when(eventRepository.findAllByIsOnlineIsTrueAndIsActiveIsTrue()).thenReturn(event
s);

when(eventMapper.listToDto(anyList())).thenReturn(eventDTOList.getEventDTOsList
());

        EventDTOList actualEventDTOList = eventService.findAllOnlineAndActive();
```

```
        assertEquals(eventDTOList.getEventDTOsList(),
actualEventDTOList.getEventDTOsList());
    }

    @Test
    void update() {
        when(eventRepository.findById(anyString())).thenReturn(Optional.of(event));
        when(eventMapper.fromDto(any(EventDTO.class))).thenReturn(event);
        when(eventRepository.save(any(Event.class))).thenReturn(event);
        when(eventMapper.toDto(any(Event.class))).thenReturn(eventDTO);

        EventDTO actualEventDTO = eventService.update(eventDTO);

        assertEquals(eventDTO, actualEventDTO);
    }

    @Test
    void deactivate() {
        when(eventRepository.findById(anyString())).thenReturn(Optional.of(event));
        when(eventRepository.save(any(Event.class))).thenReturn(event);

        eventService.deactivate("ID");

        verify(eventRepository, times(1)).save(event);
    }

    @Test
    void activate() {
        when(eventRepository.findById(anyString())).thenReturn(Optional.of(event));
        when(eventRepository.save(any(Event.class))).thenReturn(event);

        eventService.activate("ID");

        verify(eventRepository, times(1)).save(event);
    }

}

server:
  servlet:
    context-path: /vechornytsi-events

spring:
```

```
  datasource:
    url: jdbc:mysql://localhost:3306/vechornytsi_events
    username: root
    password: toor
  liquibase:
    change-log: classpath:changelog.xml
```

```
<databaseChangeLog
        xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
        xmlns:ext="http://www.liquibase.org/xml/ns/dbchangelog-ext"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog-ext
    http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-ext.xsd
    http://www.liquibase.org/xml/ns/dbchangelog
    http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.4.xsd">

    <changeSet id="events_v1" author="yuliia.kovalevska">
        <createTable tableName="event">
            <column name="id" type="varchar(36)">
                <constraints primaryKey="true"/>
            </column>
            <column name="name" type="varchar(128)">
                <constraints nullable="false"/>
            </column>
            <column name="description" type="varchar(512)">
                <constraints nullable="false"/>
            </column>
            <column name="start_date" type="date">
                <constraints nullable="false"/>
            </column>
            <column name="end_date" type="date">
                <constraints nullable="false"/>
            </column>
            <column name="place" type="varchar(256)"/>
            <column name="is_online" type="boolean">
                <constraints nullable="false"/>
            </column>
            <column name="is_private" type="boolean">
                <constraints nullable="false"/>
            </column>
            <column name="ticket_price" type="currency"/>
            <column name="free_places" type="int"/>
            <column name="organizer_id" type="varchar(36)">
                <constraints nullable="false"/>
```

```
            </column>
            <column name="is_active" type="boolean">
                <constraints nullable="false"/>
            </column>
        </createTable>
        <loadData tableName="event" file="events.csv" separator=";"/>
    </changeSet>

</databaseChangeLog>
```

```
id;name;description;start_date;end_date;place;is_online;is_private;ticket_price;free_places;organizer_id;is_active
2ed59f3c-2c6c-4669-ba29-df7254ca9a30;lorem ipsum1;dolor sit amet;2023-05-01;2024-05-01;Address1;false;false;100;10;7bb41003-29e6-45f9-bb66-2e1695cba499;true
d9a0acc8-d522-40d5-8f79-504f8c07e3a6;lorem ipsum2;dolor sit amet;2023-05-01;2024-05-01;Address2;false;false;;10;11c49676-d570-4f17-8b82-80305a940e8c;true
36b38042-c1fe-46ac-9402-36bf36a78db4;lorem ipsum3;dolor sit amet;2023-06-01;2024-06-01;Address3;false;false;100;10;47dacdd6-2e75-485c-99f2-70b31bfbe730;true
250df4d1-59e3-4945-a92d-13a10b05ac27;lorem ipsum4;dolor sit amet;2023-06-01;2024-06-01;Address4;false;true;;10;dc491818-438b-46bb-9040-af1767f5c678;true
7e8e54a6-f9e3-49cc-83f0-8cc3ab74243d;lorem ipsum5;dolor sit amet;2023-07-01;2024-07-01;Address5;false;true;100;10;409b2c15-bac3-4042-825a-605952eb4d8f;false
3fe41e23-a6f0-4b91-9d4b-e4b52e9a94e0;lorem ipsum6;dolor sit amet;2023-07-01;2024-07-01;;true;false;100;;7bb41003-29e6-45f9-bb66-2e1695cba499;true
4ca48f6d-a14d-4be6-9cc0-79b0a23b9c3e;lorem ipsum7;dolor sit amet;2023-08-01;2024-08-01;;true;false;;;11c49676-d570-4f17-8b82-80305a940e8c;true
971b896f-f0e1-41f0-83f8-9360f2a94c14;lorem ipsum8;dolor sit amet;2023-08-01;2024-08-01;;true;false;100;;47dacdd6-2e75-485c-99f2-70b31bfbe730;true
4e29a3e7-89ba-4096-bcfd-ce0d9a90c906;lorem ipsum9;dolor sit amet;2023-09-01;2024-09-01;;true;true;;;dc491818-438b-46bb-9040-af1767f5c678;true
b755117b-e28d-45df-bb31-2a6b9aac00a6;lorem ipsum10;dolor sit amet;2023-09-01;2024-09-01;;true;true;100;;409b2c15-bac3-4042-825a-605952eb4d8f;false
```

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
```

```
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.0.6</version>
    <relativePath/>
</parent>

<groupId>com.vechornytsi</groupId>
<artifactId>vechornytsi-events-service</artifactId>
<version>0.0.1-SNAPSHOT</version>

<properties>
    <java.version>17</java.version>
    <org.mapstruct.version>1.5.3.Final</org.mapstruct.version>
    <org.projectlombok.version>1.18.26</org.projectlombok.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
    </dependency>
    <dependency>
        <groupId>org.liquibase</groupId>
        <artifactId>liquibase-core</artifactId>
    </dependency>

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
    </dependency>
    <dependency>
```

```
        <groupId>org.mapstruct</groupId>
        <artifactId>mapstruct</artifactId>
        <version>1.5.3.Final</version>
    </dependency>
    <dependency>
        <groupId>com.google.protobuf</groupId>
        <artifactId>protobuf-java-util</artifactId>
        <version>3.22.2</version>
    </dependency>

    <dependency>
        <groupId>com.vechornytsi</groupId>
        <artifactId>vechornytsi-events-api-api</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </dependency>
    <dependency>
        <groupId>com.vechornytsi</groupId>
        <artifactId>vechornytsi-events-api-exception</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </dependency>
    <dependency>
        <groupId>com.vechornytsi</groupId>
        <artifactId>vechornytsi-events-api-validation</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
                <annotationProcessorPaths>
                    <path>
                        <groupId>org.mapstruct</groupId>
                        <artifactId>mapstruct-processor</artifactId>
                        <version>${org.mapstruct.version}</version>
                    </path>
                    <path>
```

```
                    <groupId>org.projectlombok</groupId>
                    <artifactId>lombok</artifactId>
                    <version>${org.projectlombok.version}</version>
                  </path>
                  <path>
                    <groupId>org.projectlombok</groupId>
                    <artifactId>lombok-mapstruct-binding</artifactId>
                    <version>0.2.0</version>
                  </path>
                </annotationProcessorPaths>
                <compilerArgs>
                  <compilerArg>
                    -Amapstruct.defaultComponentModel=spring
                  </compilerArg>
                </compilerArgs>
              </configuration>
            </plugin>
          </plugins>
        </build>

</project>
```

```
syntax = "proto2";

package proto;

option java_multiple_files = true;
option java_package = "com.vechornytsi.events.api.dto";

message EventDTO {
  optional string id = 1;
  optional string name = 2;
  optional string description = 3;
  optional int64 startDate = 4;
  optional int64 endDate = 5;
  optional string place = 6;
  optional bool isOnline = 7;
  optional bool isPrivate = 8;
  optional string ticketPrice = 9;
  optional int32 freePlaces = 10;
  optional string organizerId = 11;
  optional bool isActive = 12;
}
```

```
message EventDTOList{
  repeated EventDTO eventDTOs = 1;
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>com.vechornytsi</groupId>
        <artifactId>vechornytsi-events-api</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <relativePath>../pom.xml</relativePath>
    </parent>

    <artifactId>vechornytsi-events-api-api</artifactId>

    <dependencies>
        <dependency>
            <groupId>com.google.protobuf</groupId>
            <artifactId>protobuf-java</artifactId>
            <version>3.22.2</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.xolstice.maven.plugins</groupId>
                <artifactId>protobuf-maven-plugin</artifactId>
                <version>0.6.1</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>compile</goal>
                            <goal>test-compile</goal>
                        </goals>
```

```
            </execution>
          </executions>
        </plugin>
      </plugins>
    </build>

</project>

package com.vechornytsi.events.api.exception;

import lombok.experimental.StandardException;

@StandardException
public class InvalidInputException extends RuntimeException{
}

package com.vechornytsi.events.api.exception;

import lombok.experimental.StandardException;

@StandardException
public class RecordNotFoundException extends RuntimeException{
}

package com.vechornytsi.events.api.exception.handler;

import com.vechornytsi.events.api.exception.InvalidInputException;
import com.vechornytsi.events.api.exception.RecordNotFoundException;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;
import
org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHan
dler;

@ControllerAdvice
public class GlobalControllerExceptionHandler extends
ResponseEntityExceptionHandler {

    @ExceptionHandler(RecordNotFoundException.class)
```

```
    protected ResponseEntity<Object> handleNotFoundException(RuntimeException e,
WebRequest req) {
        return handleExceptionInternal(e, e.getMessage(), new HttpHeaders(),
HttpStatus.NOT_FOUND, req);
    }

    @ExceptionHandler(InvalidInputException.class)
    protected ResponseEntity<Object> handleInvalidInputException(RuntimeException
e, WebRequest req) {
        return handleExceptionInternal(e, e.getMessage(), new HttpHeaders(),
HttpStatus.UNPROCESSABLE_ENTITY, req);
    }

}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>com.vechornytsi</groupId>
        <artifactId>vechornytsi-events-api</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <relativePath>../pom.xml</relativePath>
    </parent>

    <artifactId>vechornytsi-events-api-exception</artifactId>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <version>3.0.4</version>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.24</version>
        </dependency>
    </dependencies>
```

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <version>2.7.7</version>
            </plugin>
        </plugins>
    </build>

</project>
```

```
package com.vechornytsi.events.api.validation;

import com.vechornytsi.events.api.exception.InvalidInputException;

import java.math.BigDecimal;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

public class DTOValidator {

    public static Validator newValidator() {
        return new Validator();
    }

    public static class Validator {

        private final List<String> invalidData = new ArrayList<>();

        public Validator checkName(String name) {
            if (name.isEmpty() || name.length() > 128) {
                invalidData.add("name:%s".formatted(name));
            }
            return this;
        }

        public Validator checkDescription(String description) {
            if (description.isEmpty() || description.length() > 512) {
                invalidData.add("description:%s".formatted(description));
            }
            return this;
        }
```

```
        public Validator checkDates(LocalDate startDate, LocalDate endDate) {
            if (startDate.isAfter(endDate)) {
                invalidData.add("start date:%s, end date:%s".formatted(startDate, endDate));
            }
            return this;
        }

        public Validator checkPlace(String place) {
            if (place!=null && (place.isEmpty() || place.length() > 256)) {
                invalidData.add("place:%s".formatted(place));
            }
            return this;
        }

        public Validator checkTicketPrice(BigDecimal price) {
            if (price!=null && (price.compareTo(BigDecimal.ZERO) < 0)) {
                invalidData.add("price:%s".formatted(price));
            }
            return this;
        }

        public Validator checkFreePlaces(Integer freePlaces) {
            if (freePlaces!=null && freePlaces < 0) {
                invalidData.add("freePlaces:%s".formatted(freePlaces));
            }
            return this;
        }

        public void validate() {
            if (!invalidData.isEmpty()) {
                throw new InvalidInputException("Invalid input data: " + invalidData);
            }
        }

    }

    private DTOValidator() {
    }

}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
```

```
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <parent>
      <groupId>com.vechornytsi</groupId>
      <artifactId>vechornytsi-events-api</artifactId>
      <version>0.0.1-SNAPSHOT</version>
      <relativePath>../pom.xml</relativePath>
   </parent>

   <artifactId>vechornytsi-events-api-validation</artifactId>

   <dependencies>
      <dependency>
         <groupId>com.vechornytsi</groupId>
         <artifactId>vechornytsi-events-api-api</artifactId>
         <version>0.0.1-SNAPSHOT</version>
      </dependency>
      <dependency>
         <groupId>com.vechornytsi</groupId>
         <artifactId>vechornytsi-events-api-exception</artifactId>
         <version>0.0.1-SNAPSHOT</version>
      </dependency>
   </dependencies>

</project>

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <parent>
      <groupId>com.vechornytsi</groupId>
      <artifactId>vechornytsi-events</artifactId>
      <version>0.0.1-SNAPSHOT</version>
      <relativePath>../pom.xml</relativePath>
   </parent>

   <artifactId>vechornytsi-events-api</artifactId>
```

```
    <packaging>pom</packaging>

    <modules>
        <module>vechornytsi-events-api-api</module>
        <module>vechornytsi-events-api-validation</module>
        <module>vechornytsi-events-api-exception</module>
    </modules>

</project>

<?xml version="1.0" encoding="UTF-8"?>
<project                                    xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>

        <groupId>com.vechornytsi</groupId>
        <artifactId>vechornytsi-events</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <packaging>pom</packaging>

        <description>Microservice for events management</description>

        <properties>
                <java.version>17</java.version>
                <maven.compiler.source>17</maven.compiler.source>
                <maven.compiler.target>17</maven.compiler.target>
                <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
                <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
        </properties>

        <modules>
                <module>vechornytsi-events-api</module>
                <module>vechornytsi-events-service</module>
        </modules>

</project>
```