**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**КАФЕДРА КОМП'ЮТЕРНИХ НАУК**

# КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

**на тему:**

# «Інформаційна технологія проектування інтерфейсу системи стрімінга сучасної музики Deezer»

**Завідувач випускаючої кафедри** **Довбиш А.С.**

**Керівник роботи** **Берест О.Б.**

**Студента групи ІН.м-02** **Васильєв Є.І.**

**СУМИ 2021**

*Сумський державний університет*
(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*
Спеціальність *«122 - Комп'ютерні науки»*

Затверджую:
зав.кафедрою ____________________
____________________
"________" ____________________ 20___р.

# ЗАВДАННЯ
# НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

*Васильєву Євгену Ігоровичу*
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія проектування інтерфейсу системи стрімінга сучасної музики Deezer*

затверджую наказом по інституту від "____" ________________ 20 ___ р. № _____________

2. Термін здачі студентом закінченого проекту (роботи)________________________

3. Вхідні данні до проекту (роботи) ____________________________________________
____________________________________________
____________________________________________
____________________________________________

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Вступ, опис технологічного прогресу у сфері обміну даними мережею інтернет; 2) Постановка завдання; 3) Загальні відомості про REST, систему Deezer API, а також процедура публікації бібліотек до репозиторію Maven; 4) Огляд існуючих інтерфейсів стрімінга сучасної музики Deezer; 5) Реалізація власного інтерфейсу стрімінга сучасної музики Deezer; 6) Висновки.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) ________
____________________________________________

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання ____________________________________________________

Керівник _________________________
(підпис)

Завдання прийняв до виконання _________________________
(підпис)

# КАЛЕНДАРНИЙ ПЛАН

| № п/п | Назва етапів дипломного проекту (роботи) | Термін виконання проекту (роботи) | Примітка |
|---|---|---|---|
| 1. | Огляд технологій, що застосовуються для передачі даних через Інтернет | | |
| 2. | Постановка задачі та формування завдань дослідження. | | |
| 3. | Опис REST архітектури, системи Deezer API, існуючих реалізацій інтерфейсів систем стрімінга сучасної музики. | | |
| 4. | Розробка інтерфейсу системи стрімінга сучасної музики Deezer | | |
| 5. | Оформлення пояснювальної записки до кваліфікаційної магістерської роботи | | |

Студент – дипломник _________________________
(підпис)

Керівник проекту _________________________
(підпис

# РЕФЕРАТ

**Записка:** 102 стор., 19 рис., 1 додаток, 9 джерел.

**Об’єкт дослідження** — система стрімінга сучасної музики Deezer.

**Мета роботи** — розробка інтерфейсу системи стрімінга сучасної музики Deezer.

**Методи дослідження** — аналіз існуючих інтерфейсів систем стрімінга сучасної музики.

**Результати** — створено Java бібліотеку для швидкої та зручної взаємодії з Deezer API. Бібліотека передбачає 80 варіантів використання Deezer API – від отримання інформації щодо того чи іншого музичного альбому до створення власних списків відтворення. Бібліотека написана з використанням мови програмування Java версії 8 та фреймворку Maven. Дана бібліотека є проектом з відкритим вихідним кодом та завантажена на GitHub. Також бібліотеку було завантажено у центральний репозиторій Maven, щоб кожен користувач міг легко підключити її у свій проект.

REST, HTTP, DEEZER API, АНАЛІЗ ІСНУЮЧИХ ІНТЕРФЕЙСІВ СТРІМІНГА МУЗИКИ, ПРОГРАМНА РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ СИСТЕМИ СТРІМІНГА DEEZER, ПУБЛІКАЦІЯ БІБЛІОТЕКИ У MAVEN CENTRAL, JAVA, JUNIT

# ЗМІСТ

ВСТУП .......... 5

1 ПОСТАНОВКА ЗАДАЧІ .......... 7

2 ЗАГАЛЬНІ ВІДОМОСТІ .......... 8

2.1 Принципи проектування REST .......... 9

2.2 Як працює REST API .......... 10

2.3 Використання протоколу HTTP .......... 11

2.4 Deezer API .......... 13

2.5 Публікація у Maven Central Repository .......... 22

3 ІСНУЮЧІ РІШЕННЯ .......... 25

4 РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ .......... 28

4.1 Дизайн інтерфейсу .......... 28

4.2 Програмна реалізація .......... 32

4.3 Приклад застосування .......... 57

ВИСНОВКИ .......... 59

СПИСОК ЛІТЕРАТУРИ .......... 61

ДОДАТОК – КОД ПРОГРАМИ .......... 62

# ВСТУП

В епоху інформаційних технологій найбільшою цінністю стала інформація. З 1990-х інтернет технології різко увійшли у практично всі сфери нашого життя. У 1989 році Тім Бернерс-Лі винайшов всесвітню мережу (World Wide Web), а в 1994 інтернет став фактично найбільш швидко розвиваючою сферою.[1] У 1996 було створено та популяризовано стандарт протоколу HTTP/1, що уніфікував обмін даними між усіма користувачами інтернету. Цей протокол (в новіших версіях) ми використовуємо й досі, коли переглядаємо інтернет-сторінки, дивимося фільми та серіали в онлайн кінотеатрах, слухаємо музику, граємо в ігри та ін.[2]

У новому столітті активно розповсюдився підхід до архітектури мережевих протоколів під назвою REST. Працюючи на базі HTTP цей архітектурний стиль значною мірою уніфікував процес запитів та відповідей між серверною та клієнтською частинами веб застосунків. Даний підхід спонукав виникненню таких програмних рішень як сервіси та мікро-сервіси. Тепер більшість веб застосунків не створюють у вигляді громіздких проектів, що виконують усю бізнес логіку в одному місці. Зараз найбільш активно використовують сервіси – (відносно) невеликі програмні рішення, що виконують лише одну поставлену їм задачу. Таким чином декілька сервісів окремо виконують свою роботу і обмінюються між собою результати за допомогою HTTP запитів. Такий підхід спонукав створення документованого списку правил для створення HTTP запитів – REST API. Деякі такі API стали публічно відкритими, щоб будь-який користувач міг звертатись до інформаційної системи деякого ресурсу у зручному для користувача вигляді. Зараз найбільш популярним стало використання публічних REST API для створення чат-ботів та створення користувацьких застосунків, що модифікують оригінальний клієнт.

Самі лише знання відкритих REST API дозволяють використовувати їх для створення власних рішень, але залишають за собою багато ручної роботи для користувача. Зазвичай це виражається у монотонній роботі по створенню HTTP запитів, їх відправка та отримання. Ця робота корисна, але забирає немалу кількість часу. Тому разом з відкритим API компанії також створюють стандартний набір для створення користувацьких застосунків – SDK. SDK зазвичай являє собою готову бібліотеку, що приховує від користувача усю рутинну та (відносно) низькорівневу роботу з «голими» HTTP запитами. Але на практиці далеко не всі відкриті API мають відповідну бібліотеку для вашої улюбленої мови програмування. Тому у цій роботі буде розглянуто створення такої бібліотеки – інтерфейс доступу до системи стрімінга сучасної музики Deezer.

# 1 ПОСТАНОВКА ЗАДАЧІ

У даній роботі необхідно розглянути ідеї та принципи підходу до архітектури мережевих протоколів REST. Потрібно ознайомитись з чого він складається та які має обмеження. Також треба розглянути як працювати з даним стилем, які можуть буть сценарії використання, ознайомитись з прикладами використання.

Також необхідно дослідити відкриту документацію Deezer API. Необхідно дізнатись, з чого починається робота, як виглядають базові запити до сервісу Deezer, які є обмеження та відповіді. Варто розглянути, у якому форматі можна передавати дані, та визначити, який буде найбільш зручним для реалізації власної інформаційної системи. Треба ознайомитись, які об'єкти існують у зовнішній системі, та як взаємодіяти з ними. Нарешті, треба дізнатись, як саме можна отримати дані користувача та керувати його даними для реалізації користувацького клієнта. Необхідно створити програмний інтерфейс системи стрімінга Deezer.

Оскільки, на виході ми плануємо поширювати нашу бібліотеку серед усіх користувачів, ми маємо вирішити, до якого відкритого сховища ми можемо звантажити бібліотеку. Для бібліотек написаних мовою Java відповідь очевидна – Maven Central. Це є найпопулярніший відкритий репозиторій для Java проектів серед спільноти Java розробників. Тому необхідно ознайомитись з документацією та процедурою публікації бібліотеки у репозиторії Maven Central. Необхідно дізнатись, які є вимоги до бібліотек, як їх правильно оформлювати та як підготуватись до її завантаження у віддалений репозиторій.

# 2 ЗАГАЛЬНІ ВІДОМОСТІ

Акронім REST означає REpresentational State Transfer. Спочатку цей термін придумав Рой Філдінг, який також був винахідником протоколу HTTP. Вражаючою особливістю служб REST є те, що вони хочуть якнайкраще використовувати HTTP.[3]

REST використовується в веб-розробці і є широко прийнятим набором рекомендацій для створення програмних інтерфейсів (API). Веб-додатки, що відповідають обмеженням REST, ще називають RESTful. RESTful додатки зазвичай базуються на методах HTTP для доступу до ресурсів за допомогою параметрів, закодованих URL-адресами, і використання JSON або XML для передачі даних.

«Веб-ресурси» вперше були визначені у всесвітній мережі як документи або файли, ідентифіковані за їх URL-адресами. Сьогодні це визначення є набагато більш загальним і абстрактним і включає кожну річ, сутність або дію, яку можна ідентифікувати, назвати, адресувати, обробити або виконати будь-яким чином в Інтернеті. У веб-додатку REST запити до URI ресурсу повертають відповідь із даними, відформатованим у HTML, XML, JSON або іншому форматі. Наприклад, відповідь може підтвердити, що стан ресурсу було змінено. Відповідь також може включати гіпертекстові посилання на пов'язані ресурси. Найпоширенішим протоколом для цих запитів і відповідей є HTTP. Він забезпечує операції (методи HTTP), такі як GET, POST, PUT і DELETE та ін. Використовуючи протокол без стану та стандартні операції, RESTful додатки націлені на швидку продуктивність, надійність і здатність розвиватися за рахунок повторного використання компонентів, якими можна керувати та оновлювати їх, не впливаючи на систему в цілому, навіть під час її роботи.

Метою REST є підвищення продуктивності, масштабованості, простоти, модифікації, видимості, портативності та надійності. Це досягається завдяки

дотриманню принципів REST, таких як архітектура клієнт-сервер, відсутність стану, кешування, використання багатошарової системи, підтримка коду на вимогу та використання єдиного інтерфейсу. [4]

## 2.1 Принципи проектування REST

На найпростішому рівні API — це механізм, який дозволяє програмі чи службі отримати доступ до ресурсу в іншій програмі чи службі. Програма або служба, яка здійснює доступ, називається клієнтом, а програма або служба, що містить ресурс, називається сервером.

Деякі API, такі як SOAP або XML-RPC, накладають суворі рамки на розробників. Але REST API можна розробляти практично з використанням будь-якої мови програмування та підтримувати різноманітні формати даних. Єдина вимога полягає в тому, щоб вони відповідали наступним шести принципам проектування RESTful сервісів - також відомим як архітектурні обмеження:

1. Уніфікований інтерфейс. Усі запити API для одного ресурсу мають виглядати однаково, незалежно від того, звідки надходить запит. REST API повинен гарантувати, що один і той самий фрагмент даних, наприклад ім’я або адреса електронної пошти користувача, належить лише одному єдиному ідентифікатору ресурсу (URI). Ресурси не повинні бути занадто великими, але повинні містити всю інформацію, яка може знадобитися клієнту.
2. Розділення на клієнт-сервер. У дизайні REST API клієнтські та серверні програми повинні бути повністю незалежними один від одного. Єдина інформація, яку клієнтська програма має знати, це URI запитуваного ресурсу; він не може взаємодіяти із серверною програмою будь-якими іншими способами. Аналогічно, серверна

програма не повинна змінювати клієнтську програму, крім передачі її запитаним даним через HTTP.

3. Відсутність стану. REST API не мають стану, тобто кожен запит повинен містити всю інформацію, необхідну для його обробки. Іншими словами, REST API не вимагають жодних сеансів на стороні сервера. Серверним програмам не дозволяється зберігати будь-які дані, пов’язані з клієнтським запитом.
4. Можливість кешування. Якщо можливо, ресурси повинні бути кешовані на стороні клієнта або сервера. Відповіді сервера також повинні містити інформацію про те, чи дозволено кешування для доставленого ресурсу. Мета полягає в тому, щоб підвищити продуктивність на стороні клієнта, одночасно збільшуючи масштабованість на стороні сервера.
5. Багатошарова архітектура системи. У REST API виклики та відповіді проходять через різні рівні. Не допускайте, щоб клієнтські та серверні програми підключались безпосередньо один до одного. У комунікаційному циклі може бути кілька різних посередників. REST API повинні бути розроблені так, щоб ні клієнт, ні сервер не могли визначити, він спілкується з кінцевою програмою чи з посередником.
6. Код на вимогу (необов'язково). REST API зазвичай надсилає статичні ресурси, але в деяких випадках відповіді також можуть містити виконуваний код (наприклад, аплети Java). У цих випадках код повинен виконуватися лише на вимогу.[5]

## 2.2 Як працює REST API

API REST спілкуються через HTTP-запити для виконання стандартних функцій бази даних, таких як створення, читання, оновлення та видалення

записів (також відомих як CRUD) у ресурсі. Наприклад, REST API використовуватиме запит GET для отримання запису, запит POST для його створення, запит PUT для оновлення запису та запит DELETE для його видалення. Усі методи HTTP можна використовувати у викликах API. Добре розроблений API REST схожий на веб-сайт, що працює у веб-браузері з вбудованою функціональністю HTTP.

Стан ресурсу в будь-який конкретний момент або мітка часу відомий як представлення ресурсу. Цю інформацію можна надати клієнту практично в будь-якому форматі, включаючи JSON, HTML, XLT, Python, PHP або звичайний текст. JSON один з найбільш популярних форматів, оскільки його легко може прочитати як люди, так і машини, і він не залежить від мови програмування.

Заголовки та параметри запитів також важливі у викликах REST API, оскільки вони містять важливу інформацію про ідентифікатор, таку як метадані, авторизації, уніфіковані ідентифікатори ресурсів (URI), кешування, файли cookie тощо. Заголовки запитів і відповіді, а також звичайні коди статусу HTTP використовуються в добре розроблених API REST.

## 2.3 Використання протоколу HTTP

Типове використання HTTP методів:

Таблиця 2.1 HTTP методи

| URL | Колекція https://api.example.com/resources/ | Ресурс https://api.example.com/resources/item17 |
|---|---|---|
| GET | Повертає URI ресурсів колекції. | Повертає об’єкт колекції. |
| PUT | Змінює одну колекцію на іншу. | Замінює певний ресурс колекції, і якщо він не існує, створює його. |
| PATCH | Зазвичай не використовують | Оновлює певний об’єкт колекції. |

| POST | Створює новий ресурс в колекції. URI для нового ресурсу застосунок призначає автоматично та зазвичай повертає у відповіді. | Зазвичай не використовують |
|---|---|---|
| DELETE | Видаляє всю колекцію. | Видаляє певний ресурс колекції. |

Хоча ресурс може бути яким завгодно, дії, які можна виконувати над ресурсом, визначаються повідомленнями, які визначено стандартним протоколом. В системі WWW цей протокол — HTTP, але існують REST-архітектури на основі й інших протоколів.

Стандарт HTTP визначає 8 типів повідомлень.

Найчастіше використовують 4 з них:

• GET — отримати представлення ресурсу

• DELETE — знищити ресурс

• POST — створити новий ресурс на місці даного використавши передане представлення

• PUT — замінити стан поточного ресурсу станом, що описується переданим представленням

Ці використовуються, щоб дослідити API:

• HEAD — отримати заголовки, які б відсилались разом з представленням цього ресурсу, але не саме представлення.

• OPTIONS — визначити список методів, на які цей ресурс відповідає.

Методи CONNECT і TRACE, використовуються лише для HTTP-проксі.

Дев'ятий метод, описаний не в  стандарті HTTP, а в додатку RFC 5789: PATCH — змінити лише частину даного ресурсу на основі даного представлення. Якщо якась частина ресурсу не згадується в переданому представленні — не модифікувати її. Це знижує кількість інформації, що необхідно передавати.

Існує ще два методи, що описуються в пропозиції до стандарту «Internet-Draft „snell-link-method“»:

- LINK — прив'язати певний ресурс до цього
- UNLINK — відв'язати ресурс від цього

Методи GET, PUT і DELETE — ідемпотентні. Це значитьє, що незалежно від того, скільки разів ви виконаєте запит— ви отримаєте один і той же результат. Звичайно, DELETE спершу поверне 204 No Content, а потім 404 Not Found, але ресурсу не буде, як після одного видалення, так і після десяти. Ідемпотентність є дуже важливою в мережі інтернет, де ви не знаєте, чи досяг запит успіху, і, у разі відсутності відповіді, надсилаєте його ще раз. POST — не ідемпотентний, тобто, якщо ви відправите POST запит на створення об’єкту кілька разів, ви отримаєте декілька відповідей.[4]

## 2.4 Deezer API

Deezer API надає зручний набір сервісів REST API для створення веб-застосунків з доступом до музичного каталогу Deezer.

**Запити**

Більшість запитів виконуються за допомогою звичайного HTTP GET методу. Базове URL кожного API методу має наступний вигляд:

https://api.deezer.com/version/service/id/method/?parameters

Важливо: існує ліміт на кількість звернень до Deezer API – не більше 50 запитів за 5 секунд.

Декілька прикладів простих запитів:

1. https://api.deezer.com/user/2529
2. https://api.deezer.com/user/2529/playlists
3. https://api.deezer.com/album/302127

**Формат відповідей**

Щоб знайти виконаця, який має ім’я 'eminem' і отримати результат у форматі xml небхідно виконати наступний запит:

GET /search/artist/?q=eminem&index=0&limit=2&output=xml

Доступні наступні формати відповідей:

Таблиця 2.2 Формати відповідей

| **Формат** | **Accept Header** | **Розширення** |
|---|---|---|
| JSON | application/json | .json |
| JSONP | application/json | .json |
| XML | application/xml, text/xml | .xml |
| PHP | application/xml, application/json | .php |

**Кодування**

Усі запити та відповіді повинні бути у кодуванні UTF-8.

**Глобальні параметри**

Deezer API пропонує декілька глобальних параметрів, щоб спростити та упорядкувати запити.

**Пагінація**

Коли відповіддю на ваш запит є список об'єктів, ви можете не отримувати весь результат одразу, а отримати лише їх частку. Існують параметри, що регулюють кількість об'єктів, що повертаються.

Ці параметри наступні:

1. **index** – номер першого об'єкта зі списку, що бажаємо повернути
2. **limit** – максимальна кількість об'єктів у списку

Приклади таких запитів:

3. https://api.deezer.com/playlist/4341978/tracks?index=0&limit=10
4. https://api.deezer.com/playlist/4341978/tracks?index=3&limit=7
5. https://api.deezer.com/playlist/4341978/tracks?limit=2

**Токен доступу**

Deezer API використовує протокол OAuth 2.0 для аутентифікації та авторизації.

Цей протокол надає токен доступу відповідно до прав, які ви запитуєте у користувача та якими права користувач погодився поділитися з вашим додатком.

Як тільки ви отримали даний токен, ви можете використовувати його для запитів як звичайний параметр. Токен може мати певний термін дії. Якщо термін дії закінчився, його необхідно поновити (запросити новий токен).

**Методи запитів**

Deezer API – це RESTful API, отже, ви маєте виконувати запит GET HTTP, щоб отримувати інформацію, POST HTTP, щоб оновлювати або додавати дані, та DELETE HTTP, щоб видаляти дані.

Існує спеціальний параметр `request_`method, щоб перевизначати HTTP запит і виконувати запити POST або DELETE використовуючи GET HTTP.

Приклад такого запиту:

https://api.deezer.com/album/302127?note=5&request_method=POST

**OAuth**

Deezer API використовує протокол OAuth 2.0 для аутентифікації та авторизації. Сервіс підтримує два шляхи для авторизації, які можна використовувати для авторизації у вашому додатку.

**Авторизація користувача**

Deezer Api підтримує два шляхи для авторизації користувача: на боці сервера та на боці клієнта. Авторизація на боці сервера використовується, коли вам необхідно виконувати запити до Deezer з вашого веб-серверу.

Авторизація на боці клієнта використовується клієнтськими застосунками, такі як браузерні застосунки з використанням Javascript.

Авторизація складається з трьох кроків:

1. Авторизація користувача – користувач вводить дані свого профілю.
2. Авторизація застосунку – користувачеві надається перелік прав, які він має передати застосунку.
3. Аутентифікація застосунку – користувач погоджується з передачею прав вашому застосунку.

**Авторизація на боці сервера**

Перші два кроки доповнюються перенаправленням користувача на вікно авторизації до сервісу deezer.com. Це дозволяє авторизувати користувача (крок 1) та показати користувачеві які права він має передати застосунку, а також який саме застосунок запитує ці права (крок 2).

**Параметри**

Коли ви виконуєте перенаправлення на вікно авторизації до сервісу deezer.com, ви маєте передати наступні параметри:

Таблиця 2.3 Параметри аутентифікації

| **Параметр** | **Опис** | **Обов'язковий** | **Значення за замовчуванням** |
|---|---|---|---|
| app_id | ID вашого застосунку, яке ви отримали на сторінці розробника. | Так | |
| redirect_uri | Посилання, на яке буде перенаправлено користувача після аутентифікації. `redirect_uri` повинно бути з того ж домену, що і ваш застосунок. | Так | |
| perms | Список прав, що необхідні вашому застосунку від користувача. | Ні | basic_access |

Приклад посилання на аутентифікацію користувача:

https://connect.deezer.com/oauth/auth.php?app_id=YOUR_APP_ID&redirect_uri=YOUR_REDIRECT_URI&perms=basic_access,email

Якщо користувач не авторизований, йому буде запропоновано ввести дані свого профілю і авторизуватися:

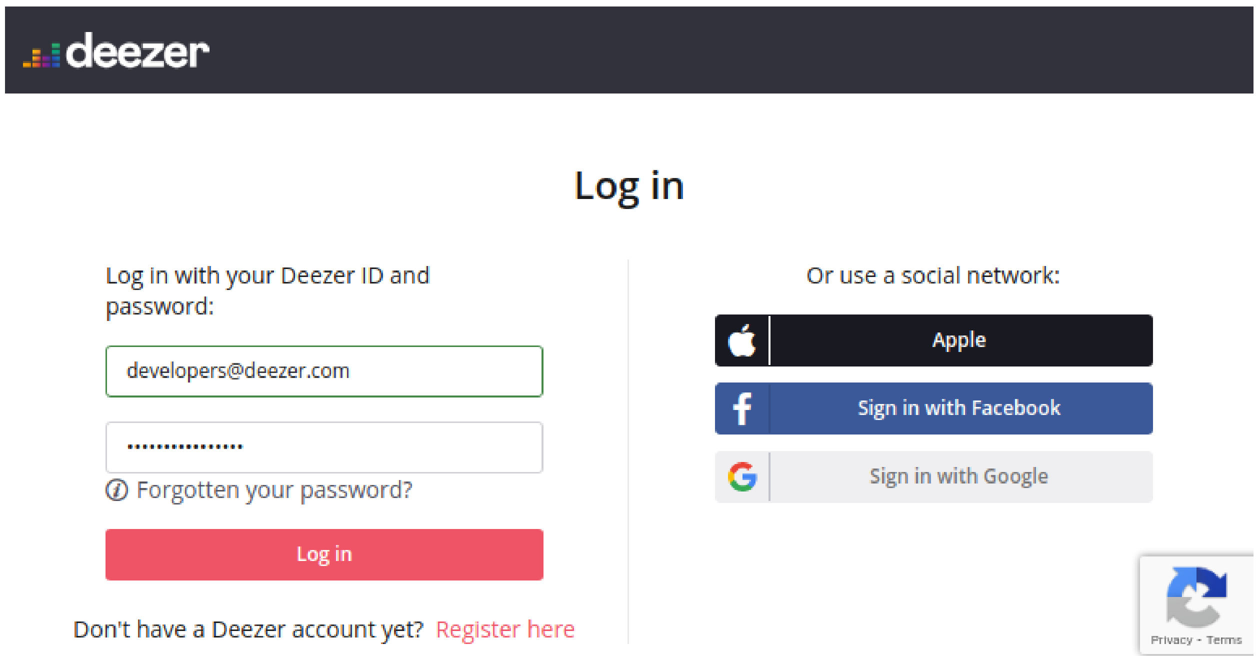


Рисунок 2.1 – Авторизація користувача[7]

Після авторизації користувачеві буде надано список прав, що необхідні застосунку:

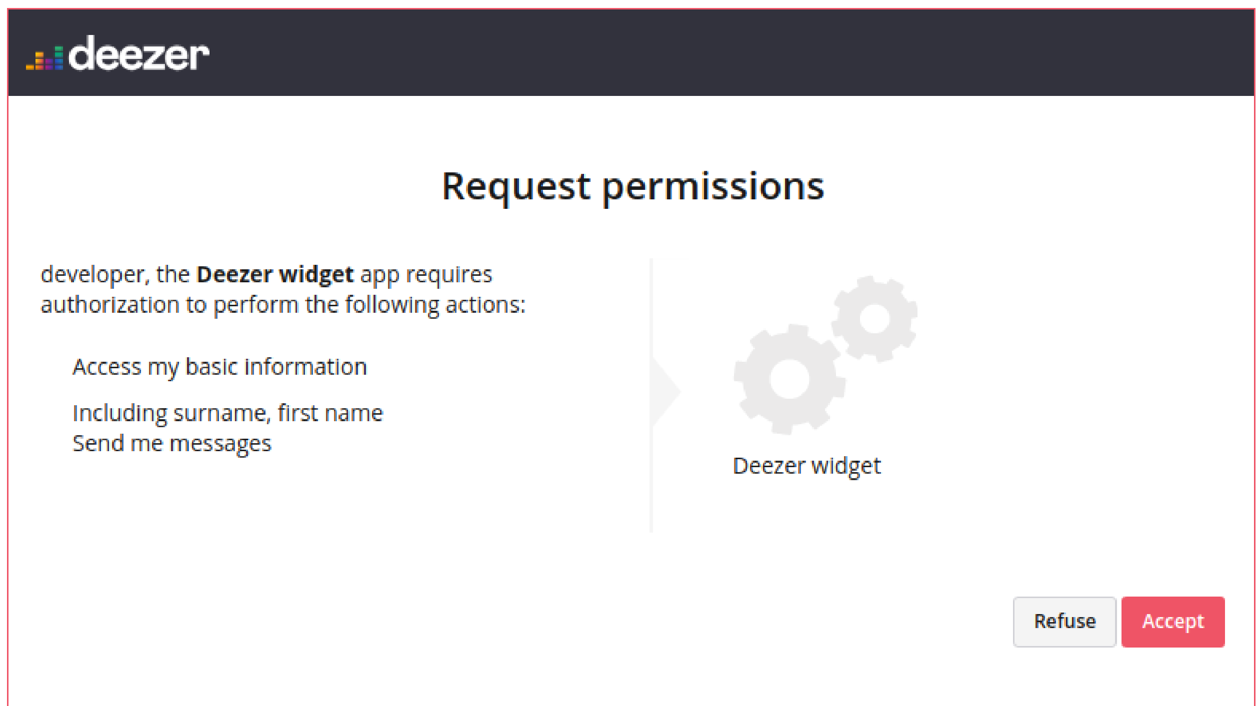


Рисунок 2.2 – Запит необхідних прав у користувача[7]

Якщо користувач натискає «Не дозволяю», ваш застосунок не буде авторизовано. Браузер перенаправить користувача на 'redirect_uri' (через HTTP 302) з параметром 'error_reason' = 'user_denied':

http://redirect_uri?error_reason=user_denied

Якщо користувач натисне «Дозволяю», ваш застосунок буде авторизовано. Браузер перенаправить користувача на 'redirect_uri' (через HTTP 302) з параметром 'code' = 'an authorization code ':

http://redirect_uri?code=A_CODE_GENERATED_BY_DEEZER

З цим кодом ви маєте можливість запросити токен доступу, який є обов’язковим для виконання дій, що відповідають наданим вам правам. Цей крок необхідний для аутентифікації застосунку. Що запросити токен доступу, вам потрібно передати отриманий код і секретний код застосунку до запиту:

https://connect.deezer.com/oauth/access_token.php?app_id=YOU_APP_ID &secret=YOU_APP_SECRET&code=THE_CODE_FROM_ABOVE

Параметри:

Таблиця 2.4 Параметри запиту

| **Параметр** | **Опис** | **Обов’язковий** | **Значення за замовчуванням** |
|---|---|---|---|
| app_id | ID вашого застосунку, яке ви отримали на сторінці розробника. | Так | |
| secret | Секретний код доступний на сторінці налаштувань вашого застосунку, і не повинен бути переданий стороннім особам. | Так | |
| code | Код, що було отримано на попередньому кроці. | Так | |

| output | Спосіб, яким ви хочете вивести access_token. | Ні | json<br>xml |
|---|---|---|---|

Якщо додаток успішно аутентифіковано і авторизаційний код є валідним, тоді авторизаційний сервер поверне токен доступу.

Рисунок 2.3 – Токен доступу[7]

Окрім токена доступу буде також повернуто число, що означає кількість секунд, протягом яких цей токен буде дійсний. Коли строк дії закінчиться, ми маємо повторити процес, щоб згенерувати новий код, та отримати новий токен.

Якщо користувач уже авторизувався у вашому додатку, йому не потрібно знову виконувати вхід. Якщо нам потрібен токен доступу, що має необмежений строк дії, щоб виконувати деякі дії навіть тоді, коли користувач не авторизований, нам потрібно запросити права «offline_access». У цьому випадку запит на токен доступу поверне `expires=0`.

**Авторизація на боці клієнта**

Авторизація на боці клієнта також використовує OAuth вікно для аутентифікації та авторизації з єдиним доповненням: ми маємо передати параметр `response_type=token` до запиту.

Так само, як і у авторизації на боці сервера, ми можемо запитувати додаткові права, використовую параметр `perms`. Як тільки користувач буде аутентифікований і додаток буде авторизований, браузер перенаправе користувача на `redirect_uri`. Єдина різниця буде у тому, що `access_token` буде записаний прямо в адресному рядку браузера (з ключом `access_token`)

**Дозволи**

Як було вказано вище, під час процесу аутентифікації необхідно запитувати у користувача дозволи. Далі представлено повний список дозволів:

Таблиця 2.5 Дозволи користувача

| **Назва** | **Дозвіл** | **Опис** |
|---|---|---|
| basic_access | Доступ до базової інформації | Включає логін, ім’я, зображення профілю користувача |
| email | Доступ до поштової скриньки користувача | Отримує поштову скриньку користувача |
| offline_access | Доступ до даних користувача у будь-який час | Додаток може звертатись до даних користувача у будь-який час |
| manage_library | Керування бібліотекою користувача | Додавання/перейменування списку відтворення. Додавання/сортування пісень у списку |
| manage_community | Керування друзями користувача | Додати/видалити друга |
| delete_library | Видалити пісні з бібліотеки | Дозволяє додатку видаляти пісні з бібліотеки користувача |
| listening_history | Історія прослуховувань користувача | Дає додатку доступ до історії прослуховувань користувача |

## 2.5 Публікація у Maven Central Repository

У результаті виконання цієї роботи ми отримаємо інтерфейс системи стрімінга сучасної музики Deezer. Даний модуль буде доступний для всіх охочих та буде поширюватися за допомогою центрального репозиторію Maven. Наш проект буде створювати спеціальний артефакт, який ми будемо завантажувати на віддалений репозиторій. Інші користувачі зможуть завантажувати артефакт у свої проекти через віддалений репозиторій.

Та спершу треба правильно підготувати наш проект до публікації.

**Вимоги**

Наш проект має відповідати наступним вимогам:

1. Релізи (releases): до центрального репозиторію можна завантажувати лише релізи, тобто файли, які не зміняться та залежать лише від інших файлів, які вже випущені та доступні в репозиторії.
2. Javadoc та source: ми повині надавати документацію (javadoc) та вихідний код (source).
3. Підпис GPG
4. Мінімальна інформація в POM: існують деякі вимоги до мінімальної інформації в POM, які знаходяться в центральному репозиторії, які буде розглянуто нище.
5. Координати: важливо вибрати відповідні координати для нашого проекту. Зокрема це стосується groupId та права власності на домен.

Також необхідно вказати тип ліцензії, що застосовується до нашого проекту.[6]

Приклад базового налаштування POM:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.apache.maven</groupId>
  <artifactId>maven</artifactId>
  <version>2.0</version>
  <packaging>jar</packaging>

  <name>Maven core</name>
  <description>The maven main core project description</description>
  <url>http://maven.apache.org</url>

  <licenses>
    <license>
      <name>Apache License, Version 2.0</name>
      <url>http://www.apache.org/licenses/LICENSE-2.0.txt</url>
      <distribution>repo</distribution>
    </license>
  </licenses>

  <scm>
    <url>https://svn.apache.org/viewvc/maven</url>
  </scm>

  <dependencies>
    <dependency>
      <groupId>...</groupId>
      <artifactId>...</artifactId>
      <version>...</version>
    </dependency>
    ...
  </dependencies>
</project>
```

Коли люди завантажують артефакти з центрального репозиторію, вони можуть захотіти перевірити підписи цих артефактів PGP на сервері відкритих ключів. Якщо підписів немає, користувачі не мають гарантії, що вони завантажують оригінальний артефакт.

Центрального репозиторій Maven вимагає від авторів надати підписи PGP для всіх артефактів (усі файли, крім контрольних сум), і розповсюджувати відкритий ключ на сервер ключів, наприклад http://pgp.mit.edu.[8]

**Sonatype OSSRH**

Для публікація бібліотеки в Maven Central необхідно завантажити бінарні файли на хостинг Sonatype OSSRH. Sonatype OSSRH (OSS Repository Hosting) використовує Sonatype Nexus Repository Manager для надання послуг хостингу репозиторію для бінарних файлів проекту з відкритим. OSSRH використовує формат репозиторію Maven і дозволяє:

- завантажувати бінарні файли версії для розробки (snapshots)
- бінарні файли стадії релізу
- просувати двійкові файли випуску та синхронізувати їх із центральним репозиторієм

Початкове налаштування для нашого сховища OSSRH вимагає деяких дій вручну, після чого процес розгортання артефактів зазвичай автоматизуються, щоб передати компоненти в OSSRH. Це все одноразові кроки.

Після початкового налаштування публікація з OSSRH до Maven Central — це тривіальна дія, яку можна зробити через браузер або програмно.

**Початкові налаштування**

Треба створити тікет для створення репозиторію Sonatype.

Sonatype використовує JIRA для керування запитами.

Тож треба:

1. Створити свій обліковий запис JIRA.
2. Створити тікет нового проекту.

Це ініціює створення нового репозиторію. Зазвичай процес займає менше 2 робочих днів.

Далі треба правильно налаштувати проект Maven, щоб під час компіляції створений артефакт завантажувався до репозиторію Sonatype.

На цьому етапі наш проект буде знаходитись лише в приватному сховищі, доступне лише для учасників вашого проекту.

Щоб випустити свій проект у відкритий доступ, ми можемо або створити реліз безпосередньо з командного рядка, якщо ми використовуємо Nexus Staging Maven Plugin або Ant Tasks, або відкрити свій веб-браузер та створити реліз вручну, перейшовши за посиланням https://s01.oss.sonatype.org/.

# 3 ІСНУЮЧІ РІШЕННЯ

Подібні до нашої теми програмні рішення уже існують і користуються великою популярністю. У даному розділі розглянемо один з таких прикладів - Spotify Web API Java.

Даний проект є відкритим, він реалізує інтерфейс доступу до системи стрімінгово сервісу Spotify та доступний за посиланням https://github.com/spotify-web-api-java/spotify-web-api-java.

Сервіс Spotify, як і Deezer, теж побудований на системі REST запитів для отримання та маніпуляцією інформації з сервісу. Подібним же чином будуються HTTP запити з деякими URI та з необхідними параметрами. Аналогічно, деякі запити потребують наявності access_token, тож існує процедура по авторизації користувача, авторизації та аутентифікації додатку. Процедура аналогічна тій, що існує в Deezer – необхідно створити посилання, яке буде містити redirect_uri, користувач повинен перейти за посиланням та авторизуватися, після чого користувач буде перенаправлений на redirect_uri, а наш додаток зможе отримати code, за допомогою якого можна запросити access_token. Процедура практично ідентична, тому зупинятися на деталях не будемо.

Розглянемо програмну реалізацію. Дана бібліотека вже завантажена до репозиторію Maven, тому нам достатньо буде прописати відповідну залежність у pom.xml:

```
<dependency>
  <groupId>se.michaelthelin.spotify</groupId>
  <artifactId>spotify-web-api-java</artifactId>
  <version>6.5.4</version>
</dependency>
```

Вхідною точкою цього рішення є клас SpotifyApi. Він містить усі необхідні методи для авторизації користувача та додатку, та методи для звернень до сервісу Spotify. Приклад роботи з даною бібліотекою[9]:

```
// For all requests an access token is needed
SpotifyApi spotifyApi = new SpotifyApi.Builder()
        .setAccessToken("taHZ2SdB-bPA3FsK3D7ZN5npZS47cMy-
IEySVEGttOhXmqaVAIo0ESvTCLjLBifhHOHOIuhFUKPW1WMDP7w6dj3MAZdWT8CLI2MkZaXbYLTeoDvXesf2e
eiLYPBGdx8tIwQJKgV8XdnzH_DONk")
        .build();

// Create a request object with the optional parameter "market"
final GetSomethingRequest getSomethingRequest =
spotifyApi.getSomething("qKRpDADUKrFeKhFHDMdfcu")
        .market(CountryCode.SE)
        .build();

void getSomething_Sync() {
  try {
    // Execute the request synchronous
    final Something something = getSomethingRequest.execute();

    // Print something's name
    System.out.println("Name: " + something.getName());
  } catch (Exception e) {
    System.out.println("Something went wrong!\n" + e.getMessage());
  }
}

void getSomething_Async() {
  try {
    // Execute the request asynchronous
    final Future<Something> somethingFuture = getSomethingRequest.executeAsync();

    // Do other things...

    // Wait for the request to complete
```

```
    final Something something = somethingFuture.get();

    // Print something's name
    System.out.println("Name: " + something.getName());
  } catch (Exception e) {
    System.out.println("Something went wrong!\n" + e.getMessage());
  }
}
```

У прикладі показано ініціалізацію об’єкта класу SpotifyApi та два методи – отримання випадкових музичних записів синхронно та асинхронно. Як бачимо застосування я дуже простим та зручним для програміста. Отже, створення власних додатків проходитиме значно швидше. На даному прикладі будемо базувати власну програмну реалізацію.

# 4 РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ

## 4.1 Дизайн інтерфейсу

Оскільки сервіс Deezer є RESTful, працювати з ним будемо за наступною схемою:

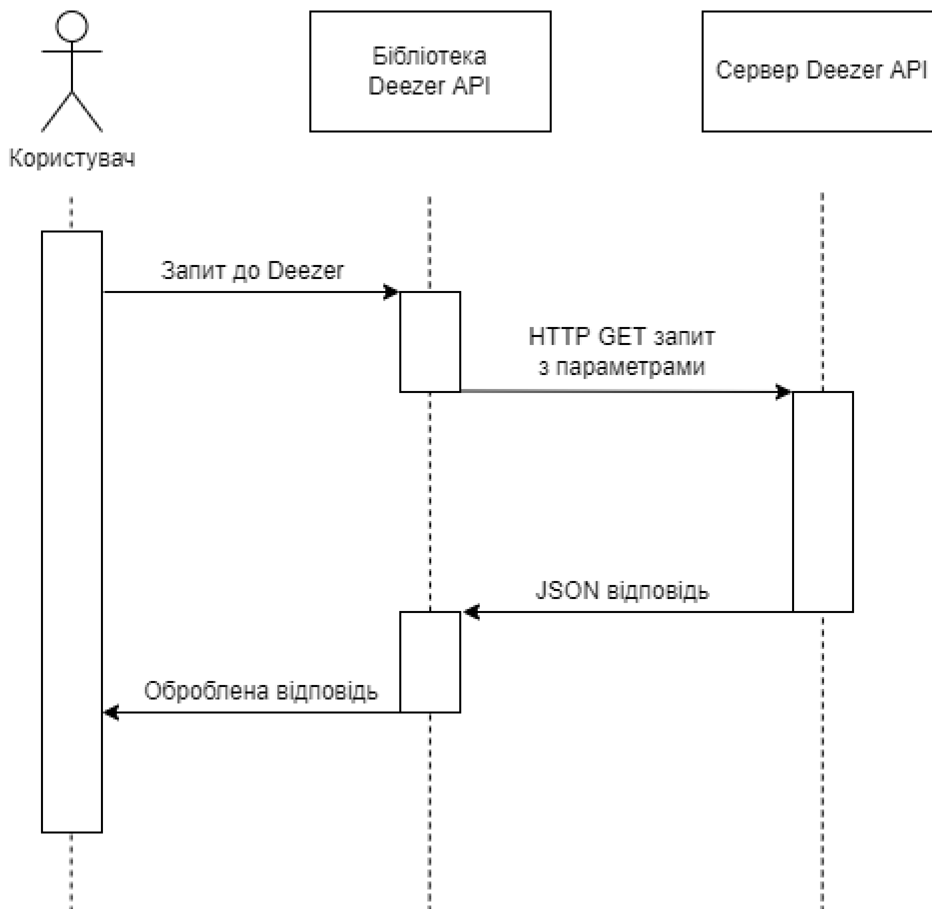


Рисунок 4.1 – Діаграма процесів взаємодії з Deezer API

Користувач, використовуючи класи з бібліотеки Deezer API створює запит до сервісу Deezer. Наша бібліотека переводить запит користувача у коректний HTTP запит та відправляє його на сервер Deezer API. Сервер оброблює запит та повертає відповідь у форматі JSON. Бібліотека парсить відповідь та трансформує його у об’єкт з об’єктної моделі. Користувач отримує об’єкт відповіді та працює з ним далі відповідно до його планів.

На основі списку об’єктів, якими оперує Deezer, створимо об’єктну модель:

Рисунок 4.2 – Модель Deezer API

У нас буде абстрактний клас ChartMember та декілька похідних від нього класів. Це необхідно, бо сервіс Deezer будує рейтинг кращих пісень, альбомів і т.д., і кожен такий рейтинг має схожу структуру. Спільні поля буде винесено у абстрактний клас. Усі інші класи також є частиною моделі, але не залежать один від одного, тому вони не мають відношень.

Для створення и виконання HTTP запитів нам знадобляться декілька сервісів. Тому ми створимо інтерфейси HttpRequest та HttpClient. Також будуть створені реалізації цих інтерфейсів. Ще нам знадобиться утилітарний

клас URLParamsEncoder, щоб форматувати текстові рядки в URL-захищений формат. Повна схема HTTP сервісів наступна:

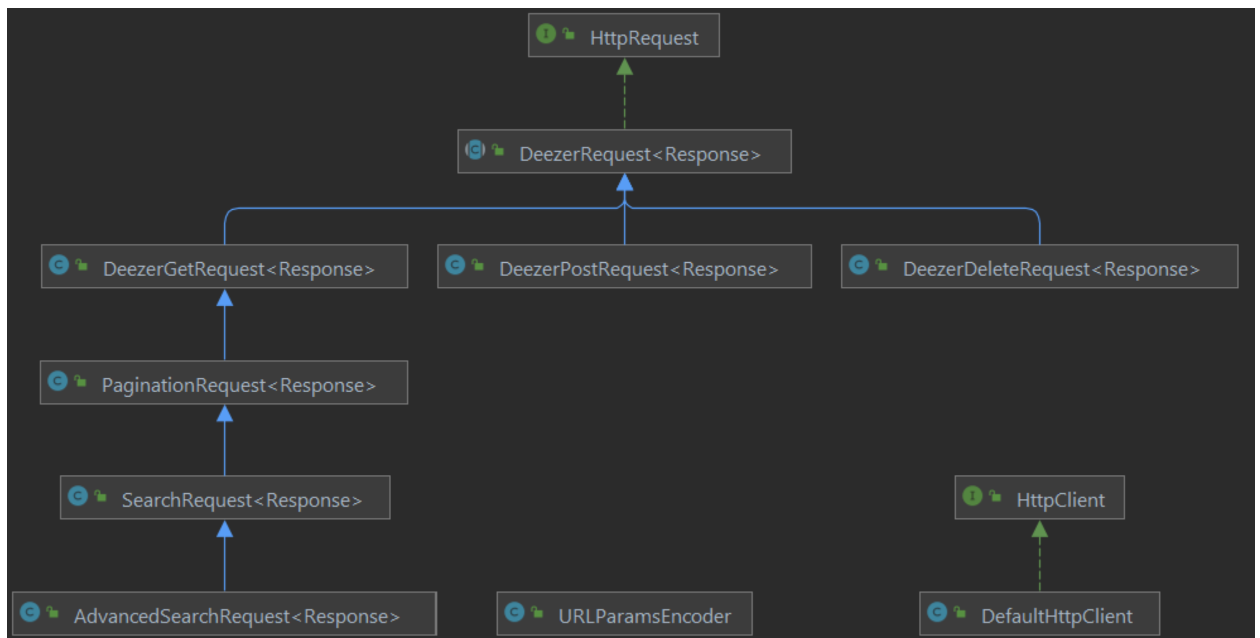


Рисунок 4.3 – HTTP сервіси

Тепер побудуємо модель запитів. У сервісі Deezer запити розподілені по групам відповідно до моделі об’єктів, тому відповідним чином ми розподілимо їх по окремих класах. У нас буде абстрактний клас DeezerRequests, який має поле accessToken. Створимо 13 класів, що відповідають 13 групам запитів Deezer. Усі класи наслідують DeezerRequests та мають свої методи для виконання запитів. Кожен метод повертає відповідний об’єкт моделі. Повна схема класів запитів має наступний вигляд:

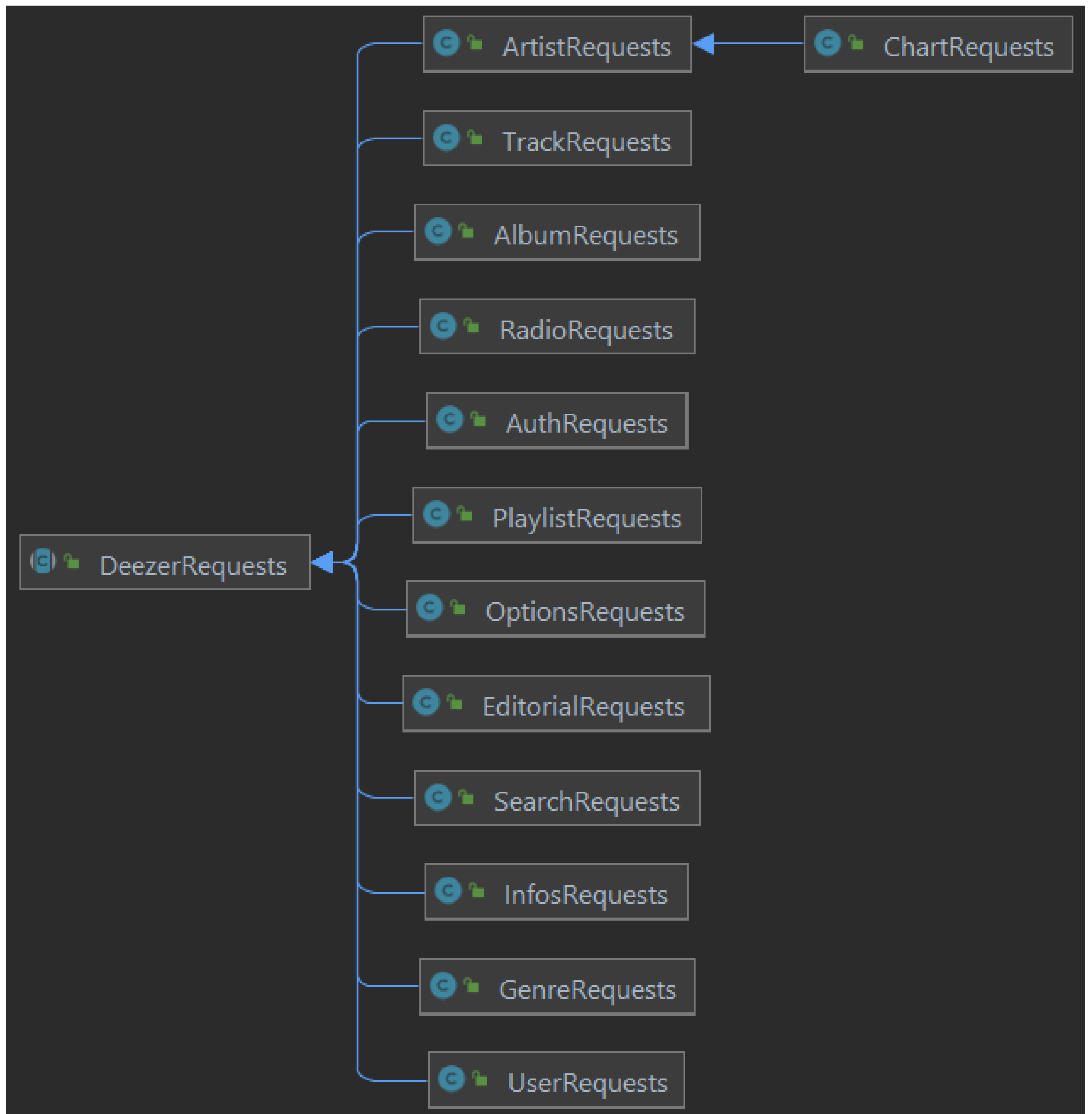


Рисунок 4.4 – Схема запитів Deezer

## 4.2 Програмна реалізація

Оскільки реалізація інформаційної системи буде мовою програмування Java, будемо використовувати середовище розробки Intellij Idea – найпопулярніше та найзручніше середовище для створення Java застосунків.

Створимо новий Maven проект під назвою deezer-api:

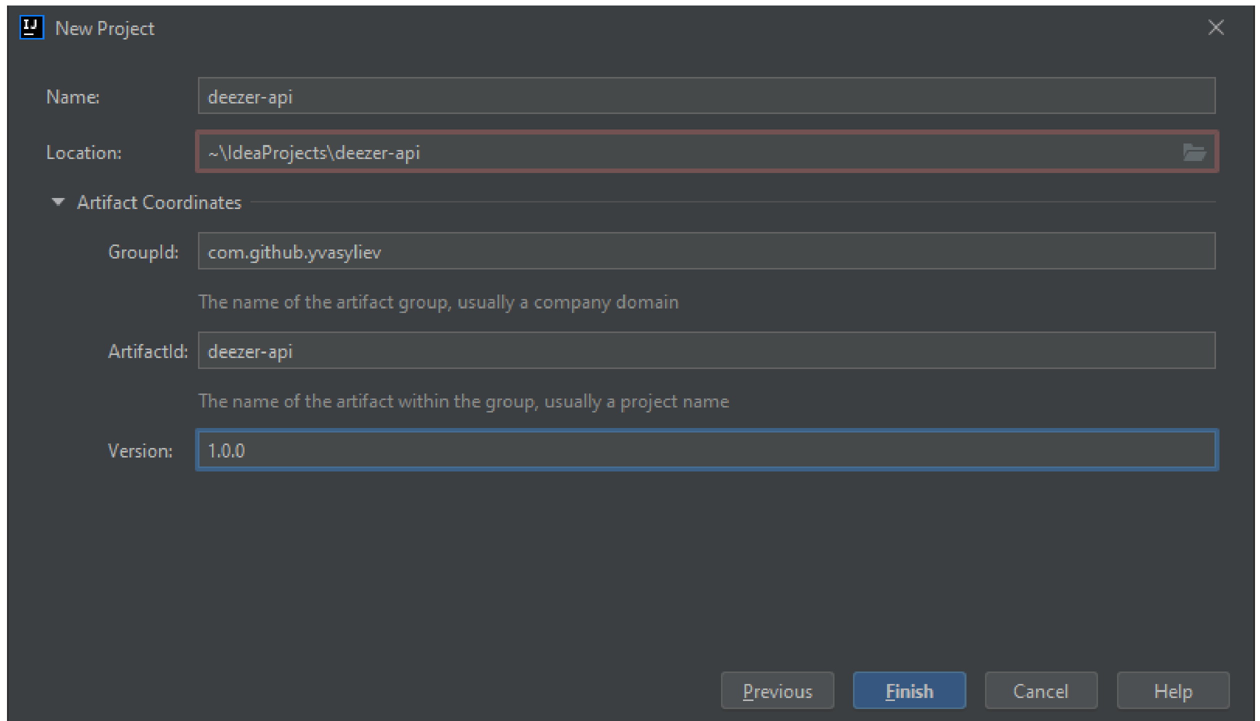


Рисунок 4.5 - Новий проект dezer-api

Відкриємо pom.xml та налаштуємо Maven проект відповідно до інструкцій.[8]

Додаймо базовий опис нашого проекту:

```
<groupId>com.github.yvasyliev</groupId>
<artifactId>deezer-api</artifactId>
<packaging>jar</packaging>
<version>1.0.1</version>
<name>Deezer API Java Library</name>
<description>A Java implementation of Deezer API.</description>
<url>https://github.com/yvasyliev/deezer-api</url>
```

Вкажемо тип ліцензії – MIT – щоб будь який користувач міг модифікувати наш інтерфейс:

```
<licenses>
    <license>
        <name>MIT License</name>
        <url>https://www.opensource.org/licenses/mit-license.php</url>
    </license>
</licenses>
```

Використовувати будемо мову Java версії 8:

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

Для нашої бібліотеки знадобляться деякі сторонні реалізації, а саме:

1. SLF4J – для логування;
2. GSON – для парсингу JSON відповідей Deezer;
3. JUnit – для написання юніт-тестів.

Вкажемо їх у pom.xml:

```
<dependencies>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>1.7.32</version>
    </dependency>
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.8.9</version>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <version>5.8.1</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Також у pom.xml ми додано деякі плагіни, за допомогою яких буде відбувати збірка проекту, запакування в архів, генерація документації та відправка даних у зовнішній репозиторій. Повний вміст pom.xml представлено у додатку.

Далі розглянемо написання коду нашої бібліотеки. Для компактності будемо вказувати лише значущі частини коду. Повний текст програми доступний у додатку або за посиланням https://github.com/yvasyliev/deezer-api.

Перш за все створимо модель нашої бібліотеки. Deezer API передбачає наступні об’єкти:

1. Album – музичний альбом;
2. Artist – виконавець;
3. Chart – чарт, рейтинг;
4. Contributor - учасник альбому;
5. Editorial – вибір редакції;
6. Folder – папка з музикою/альбомами;
7. Genre – музичний жанр;
8. Infos – інформація про даний регіон;
9. Offer – підписка;
10. Options – налаштування користувача;
11. Playlist – список відтворення;
12. Podcast – подкаст;
13. Radio – випадковий список відтворення;
14. Track – музичний запис;
15. User – користувач.

Нам знадобляться деякі допоміжні об’єкти:

16.AccessToken – токен доступу;

17.ChartMember – входить до чарту/рейтингу;

18.Permission – дозволи додатку;

19.SearchOrder – сортування результату пошуку.

Також варто зазначити, що Deezer повертає список об'єктів у вигляді іншого об'єкта з допоміжними полями. Тому у пакеті api.deezer.objects будемо зберігати основні та допоміжні об'єкти, а в пакеті api.deezer.objects.data – об'єкти колекцій.

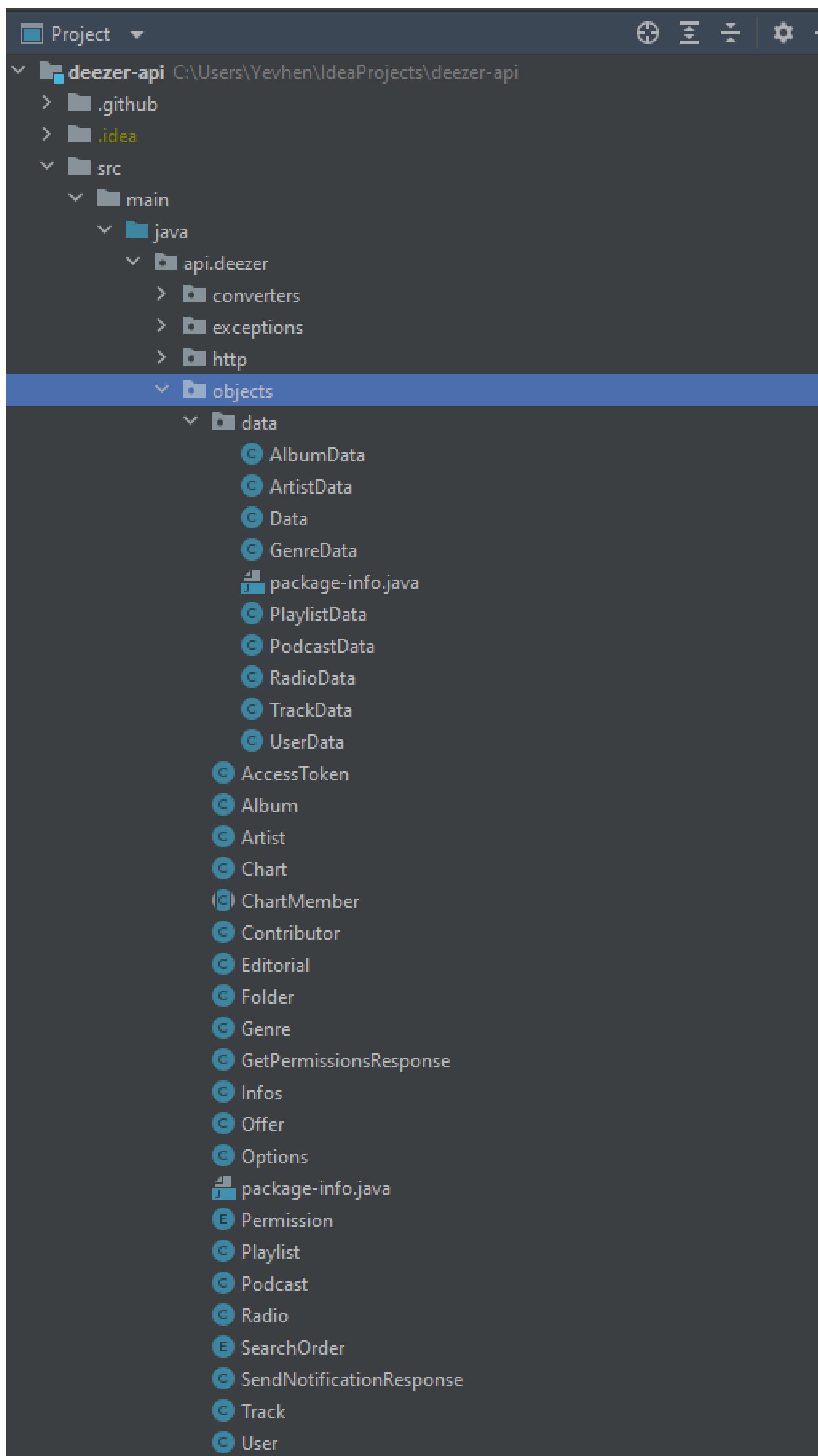


Рисунок 4.6 - Вміст пакету api.deezer.objects

Створимо файл src/main/resources/deezer/api.properties:

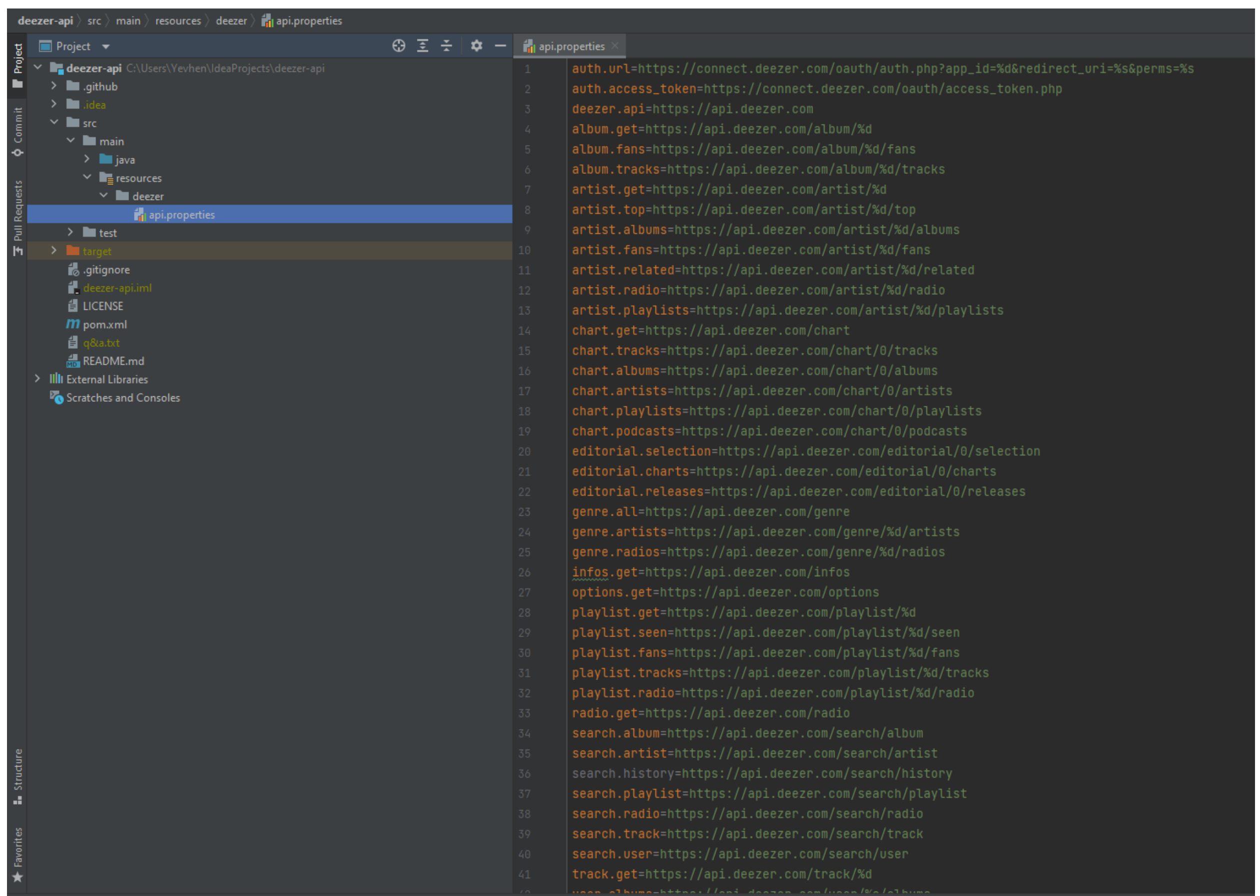


Рисунок 4.7 - Вміст файлу api.properties

У цьому файлі будемо зберігати посилання на запити Deezer API. Пізніше ми використаємо, коли будемо створювати інтерфейси запитів.

В нашій бібліотеці в деяких випадках можуть виникати помилкові ситуації, які супроводжуються генеруванням виключення (Exception). Щоб виділити такі помилки поміж інших (для спрощення аналізу, оскільки користувач бібліотека найбільш ймовірно буде використовувати також інші бібліотеки), створимо клас DeezerException в пакеті api.deezer.exceptions:

```
package api.deezer.exceptions;

/**
 * Exceptions that occur in <b>Deezer API</b> library.
 */
public class DeezerException extends Exception {
    public DeezerException(Throwable cause) {
        super(cause);
    }
```

```
    public DeezerException(String message) {
        super(message);
    }
}
```

Наша бібліотека буде відправляти та отримувати дані за протоколом HTTP у текстовому вигляді. Для цього нам необхідно передбачити деякі конвертери, що будуть трансформувати дані моделей у текст та навпаки.

Створимо інтерфейс Converter у пакеті api.deezer.converters:

```
package api.deezer.converters;

/**
 * Converts object from type of {@link From} to type of {@link To}.
 *
 * @param <From> source object type.
 * @param <To>   destination object type.
 */
public interface Converter<From, To> {
    /**
     * Converts object from type of {@link From} to type of {@link To}.
     *
     * @param from original object.
     * @return converted object.
     */
    To covert(From from);
}
```

Усі конвертери будуть реалізовувати даний інтерфейс.

Створимо конвертер AccessTokenConverter, що конвертує текстове представлення токена доступу у клас AccessToken:

```
package api.deezer.converters;

import api.deezer.objects.AccessToken;

import java.util.Arrays;
import java.util.Map;
import java.util.stream.Collectors;

/**
 * Converts Deezer API response to {@link AccessToken} object.
 */
```

```
public class AccessTokenConverter implements Converter<String, AccessToken> {
    @Override
    public AccessToken covert(String response) {
        Map<String, String> params = toParams(response);

        AccessToken accessToken = new AccessToken();
        accessToken.setAccessToken(params.get("access_token"));
        accessToken.setExpires(Integer.valueOf(params.get("expires")));

        return accessToken;
    }

    private Map<String, String> toParams(String response) {
        return Arrays.stream(response.split("&"))
                .map(param -> param.split("="))
                .collect(Collectors.toMap(param -> param[0], param -> param[1]));
    }
}
```

Створимо конвертер GsonConverter, що конвертує текстове представлення об'єкта у форматі JSON у відповідний Java клас:

```
package api.deezer.converters;

import com.google.gson.Gson;

/**
 * Converts Deezer API response to POJO. Implementation is based on {@link Gson}.
 *
 * @param <Response> POJO type.
 */
public class GsonConverter<Response> implements Converter<String, Response> {
    /**
     * POJO type.
     */
    private final Class<Response> clazz;

    /**
     * {@link Gson} object.
     */
    private Gson gson = new Gson();

    public GsonConverter(Class<Response> clazz) {
        this.clazz = clazz;
    }

    @Override
```

```
    public Response covert(String response) {
        return gson.fromJson(response, clazz);
    }

    public Gson getGson() {
        return gson;
    }

    public void setGson(Gson gson) {
        this.gson = gson;
    }
}
```

Створимо конвертер ListConverter, що конвертує список об'єктів Java класу у текстове представлення:

```
package api.deezer.converters;

import java.util.List;
import java.util.stream.Collectors;

/**
 * Converts list of objects to comma separated values.
 *
 * @param <T> object type.
 */
public class ListConverter<T> implements Converter<List<T>, String> {
    @Override
    public String covert(List<T> list) {
        return list.stream().map(Object::toString).collect(Collectors.joining(","));
    }
}
```

Створимо конвертер PermissionsConverter, що конвертує список об'єктів Permission у текстове представлення:

```
package api.deezer.converters;

import api.deezer.objects.Permission;

import java.util.List;
import java.util.stream.Collectors;

/**
 * Converts list of {@link Permission} into comma separated values.
 */
public class PermissionsConverter extends ListConverter<Permission> {
```

```
    @Override
    public String covert(List<Permission> list) {
        return
list.stream().map(Permission::getValue).collect(Collectors.joining(","));
    }
}
```

Створимо конвертер TracksDataConverter, що конвертує JSON об’єкт в TrackData:

```
package api.deezer.converters;

import api.deezer.objects.data.TrackData;
import com.google.gson.JsonElement;

/**
 * Converts Deezer API response to {@link TrackData} object.
 */
public class TracksDataConverter extends GsonConverter<TrackData> {
    public TracksDataConverter() {
        super(TrackData.class);
    }

    @Override
    public TrackData covert(String response) {
        JsonElement jsonElement = super.getGson().fromJson(response,
JsonElement.class);
        boolean isFalse = jsonElement.isJsonPrimitive()
                && jsonElement.getAsJsonPrimitive().isBoolean()
                && !jsonElement.getAsJsonPrimitive().getAsBoolean();
        return isFalse ? null : super.covert(response);
    }
}
```

Тепер створимо інтерфейс HttpRequest у пакеті api.deezer.http, який буде представляти HTTP запит:

```
package api.deezer.http;

import java.util.Map;

/**
 * Represents HTTP request.
 */
public interface HttpRequest {
    /**
     * Gets request URL.
     *
```

```
     * @return request URL.
     */
    String getUrl();

    /**
     * Gets request method. (E.g. GET or POST)
     *
     * @return request method.
     */
    String getRequestMethod();

    /**
     * Gets request URL params.
     *
     * @return request URL params.
     */
    Map<String, String> getParams();
}
```

Даний клас містить базову інформацію про кожен запит: адреса запиту (URL), HTTP метод та параметри запиту.

Створимо інтерфейс HttpClient, що буде виконувати відповідний запит:

```
package api.deezer.http;

import java.io.IOException;

/**
 * Executes HTTP requests.
 */
public interface HttpClient {
    /**
     * Executes a HTTP request.
     *
     * @param httpRequest {@link HttpRequest} object.
     * @return response body.
     * @throws IOException if errors occur.
     */
    String execute(HttpRequest httpRequest) throws IOException;
}
```

Інтерфейс містить єдиний метод execute, що виконує запит та повертає тіло відповіді на запит.

Створимо утилітарний клас URLParamsEncoder в пакеті api.deezer.http.utils, що буде трасформувати параметри запиту в екрановане для URL представлення:

```
package api.deezer.http.utils;

import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

/**
 * Encodes string values to URL-encoded values.
 */
public class URLParamsEncoder {
    /**
     * Encodes request params into URL-encoded params.
     *
     * @param params request params.
     * @return a string format <i>key1=value1{@literal &}key2=value2...</i>
     * @throws UnsupportedEncodingException if errors occur.
     */
    public static String encode(Map<String, String> params) throws
UnsupportedEncodingException {
        List<String> paramsList = new ArrayList<>(params.size());
        for (Map.Entry<String, String> entry : params.entrySet()) {
            paramsList.add(encode(entry.getKey()) + "=" + encode(entry.getValue()));
        }
        return String.join("&", paramsList);
    }

    /**
     * Encodes a string into URL-encoded string.
     *
     * @param s source string.
     * @return URL-encoded string.
     * @throws UnsupportedEncodingException if errors occur.
     */
    public static String encode(String s) throws UnsupportedEncodingException {
        return URLEncoder.encode(s, "UTF-8");
    }
}
```

Тепер перейдемо до створення реалізацій інтерфейсів.

Створимо наступні класи:

1. DefaultHttpClient – виконує HTTP запит;
2. DeezerRequest – абстрактний клас запиту;
3. DeezerGetRequest – запит HTTP GET;
4. DeezerPostRequest – запит HTTP POST;
5. DeezerDeleteRequest – запит HTTP DELETE;
6. PaginationRequest – запит HTTP GET, що містить пагінацію;
7. SearchRequest – запит HTTP GET, що містить пагінацію і виконує пошук об’єктів Deezer;
8. AdvancedSearchRequest – запит HTTP GET, що містить пагінацію і виконує пошук об’єктів Deezer з додатковою фільтрацією.

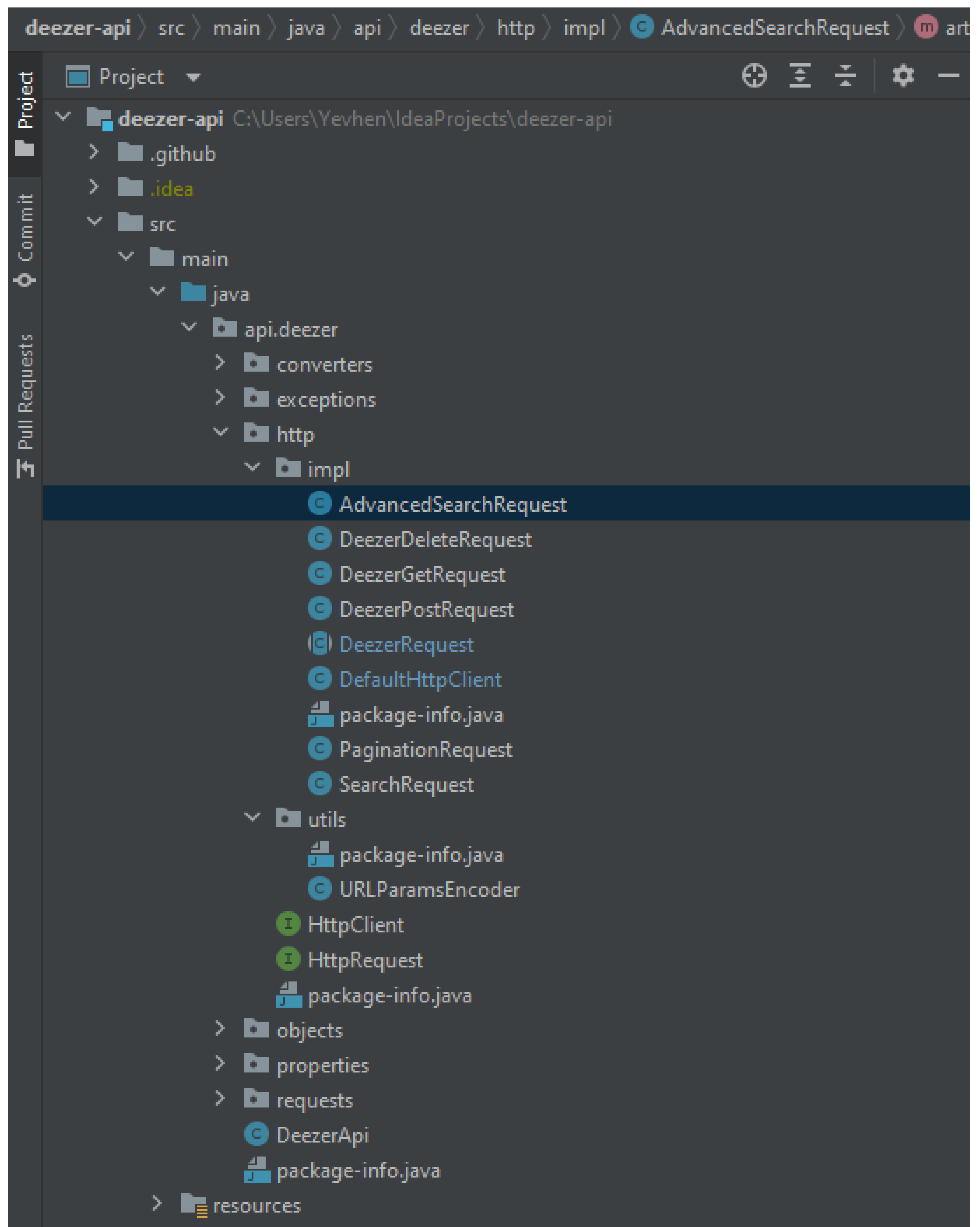


Рисунок 4.8 – Структура пакету api.deezer.http

Створимо клас DeezerProperties у пакеті api.deezer.properties, що буде зчитувати api.properties та дасть змогу звертатись до посилань Deezer API у зручному вигляді:

```
package api.deezer.properties;

import org.slf4j.Logger;
```

```
import org.slf4j.LoggerFactory;

import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;

public class DeezerProperties {
    private static final Logger LOGGER =
LoggerFactory.getLogger(DeezerProperties.class);
    private static final String PROPERTIES_FILE = "/deezer/api.properties";
    private static final Properties PROPERTIES = new Properties();

    private static void loadProperties() {
        try (InputStream inputStream =
DeezerProperties.class.getResourceAsStream(PROPERTIES_FILE)) {
            PROPERTIES.load(inputStream);
        } catch (IOException e) {
            LOGGER.error("failed to load {}", PROPERTIES_FILE);
        }
    }

    public static String getProperty(String key) {
        if (PROPERTIES.isEmpty()) {
            loadProperties();
        }
        return PROPERTIES.getProperty(key);
    }
}
```

Тепер створимо абстрактний клас DeezerRequests у пакеті api.deezer.requests:

```
package api.deezer.requests;

import api.deezer.properties.DeezerProperties;

import java.util.AbstractMap;
import java.util.Arrays;
import java.util.Map;
import java.util.stream.Collectors;

/**
 * Deezer requests.
 */
public abstract class DeezerRequests {
    /**
     * Deezer <i>access_token</i>.
     */
```

```
    private String accessToken;

    public DeezerRequests() {
    }

    public DeezerRequests(String accessToken) {
        this.accessToken = accessToken;
    }

    /**
     * Gets property from {@link DeezerProperties} and formats string with arguments.
     *
     * @param property property key.
     * @param args     format arguments.
     * @return formatted string.
     */
    protected String property(String property, Object... args) {
        return String.format(
                DeezerProperties.getProperty(property),
                args
        );
    }

    /**
     * Creates params map.
     *
     * @param entries params.
     * @return params map.
     */
    @SafeVarargs
    protected final Map<String, String> params(Map.Entry<String, String>... entries)
{
        return Arrays.stream(entries).collect(Collectors.toMap(Map.Entry::getKey,
Map.Entry::getValue));
    }

    /**
     * Creates param.
     *
     * @param key   param key.
     * @param value param value.
     * @return param entry.
     */
    public Map.Entry<String, String> entry(String key, String value) {
        return new AbstractMap.SimpleEntry<>(key, value);
    }

    /**
```

```
     * Gets Deezer <i>access_token</i>.
     *
     * @return Deezer <i>access_token</i>.
     */
    protected String getAccessToken() {
        return accessToken;
    }
}
```

Даний клас містить поле accessToken та допоміжні методи для формування запиту:

1. property(String property, Object... args) – знаходить URL за відповідним ключем та форматує текстовий рядок з параметрами;
2. params(Map.Entry<String, String>... entries) – створює мапу параметрів запиту;
3. entry(String key, String value) – створює параметр запиту.

У цьому ж пакеті створимо клас AlbumRequests:

```
package api.deezer.requests;

import api.deezer.http.impl.DeezerGetRequest;
import api.deezer.http.impl.DeezerRequest;
import api.deezer.http.impl.PaginationRequest;
import api.deezer.objects.Album;
import api.deezer.objects.data.TrackData;
import api.deezer.objects.data.UserData;

/**
 * Requests related to albums.
 */
public class AlbumRequests extends DeezerRequests {
    /**
     * Gets album by ID.
     *
     * @param albumId album ID.
     * @return {@link Album} object.
     */
    public DeezerRequest<Album> getById(int albumId) {
        return new DeezerGetRequest<>(property("album.get", albumId), Album.class);
    }

    /**
```

```
     * Gets album fans.
     *
     * @param albumId album ID.
     * @return album fans.
     */
    public PaginationRequest<UserData> getFans(int albumId) {
        return new PaginationRequest<>(property("album.fans", albumId),
UserData.class);
    }

    /**
     * Gets album tracks.
     *
     * @param albumId album ID.
     * @return album tracks.
     */
    public PaginationRequest<TrackData> getTracks(int albumId) {
        return new PaginationRequest<>(property("album.tracks", albumId),
TrackData.class);
    }
}
```

Даний клас містить методи створення запитів Deezer, що відносяться до альбомів.

Аналогічно створимо решту класів:

1. ArtistRequests – запити, що відносяться до виконавців;
2. AuthRequests – запити, що відносяться до авторизації;
3. ChartRequests– запити, що відносяться до чартів;
4. EditorialRequests– запити, що відносяться до вибору редакції;
5. GenreRequests– запити, що відносяться до жанрів;
6. InfosRequests– запити, що відносяться до інформації про регіон;
7. OptionsRequests– запити, що відносяться до опцій користувача;
8. PlaylistRequests– запити, що відносяться до списків відтворень;
9. RadioRequests– запити, що відносяться до випадкових списків відтворень;
10. SearchRequests – пошукові запити;

11. TrackRequests – запити, що відносяться до музичних записів;

12. UserRequests – запити, що відносяться до користувачів.

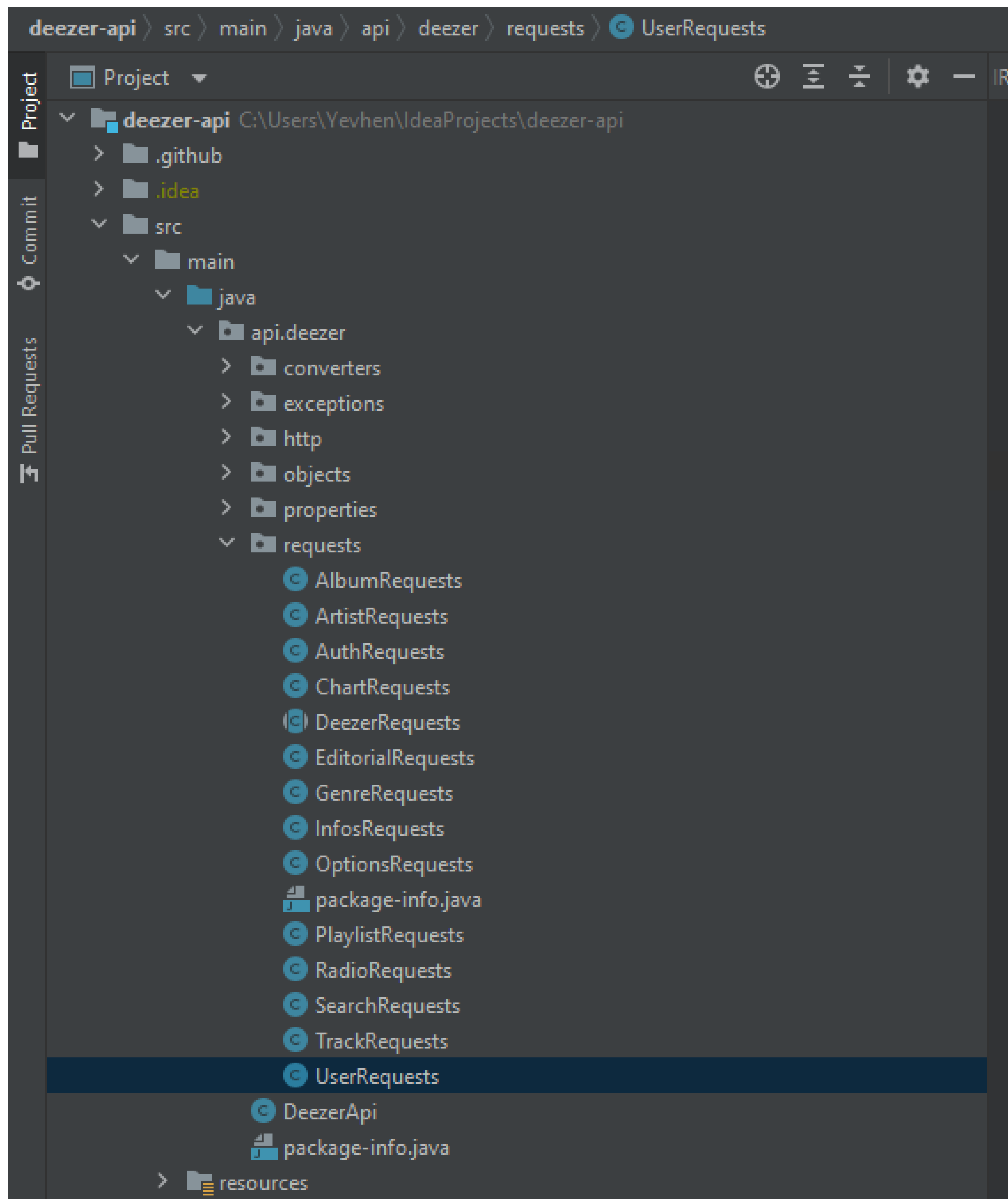


Рисунок 4.9 – Структура пакету api.deezer.requests

Тепер створимо головний клас у пакеті api.deezer – DeezerApi:

```
package api.deezer;

import api.deezer.requests.AlbumRequests;
import api.deezer.requests.ArtistRequests;
```

```
import api.deezer.requests.AuthRequests;
import api.deezer.requests.ChartRequests;
import api.deezer.requests.EditorialRequests;
import api.deezer.requests.GenreRequests;
import api.deezer.requests.InfosRequests;
import api.deezer.requests.OptionsRequests;
import api.deezer.requests.PlaylistRequests;
import api.deezer.requests.RadioRequests;
import api.deezer.requests.SearchRequests;
import api.deezer.requests.TrackRequests;
import api.deezer.requests.UserRequests;

/**
 * Deezer APIs.
 */
public class DeezerApi {
    /**
     * Deezer <i>access_token</i>.
     */
    private String accessToken;

    public DeezerApi() {
    }

    public DeezerApi(String accessToken) {
        this.accessToken = accessToken;
    }

    /**
     * Album requests.
     *
     * @return album requests.
     */
    public AlbumRequests album() {
        return new AlbumRequests();
    }

    /**
     * Artist requests.
     *
     * @return artist requests.
     */
    public ArtistRequests artist() {
        return new ArtistRequests();
    }

    /**
     * Chart requests
```

```
     *
     * @return chart requests.
     */
    public ChartRequests chart() {
        return new ChartRequests();
    }

    /**
     * Editorial requests.
     *
     * @return editorial requests.
     */
    public EditorialRequests editorial() {
        return new EditorialRequests();
    }

    /**
     * Genre requests.
     *
     * @return genre requests.
     */
    public GenreRequests genre() {
        return new GenreRequests();
    }

    /**
     * Infos requests.
     *
     * @return infos requests.
     */
    public InfosRequests infos() {
        return new InfosRequests();
    }

    /**
     * Options requests.
     *
     * @return options requests.
     */
    public OptionsRequests options() {
        return new OptionsRequests();
    }

    /**
     * Playlist requests.
     *
     * @return playlist requests.
     */
```

```
    public PlaylistRequests playlist() {
        return new PlaylistRequests(accessToken);
    }

    /**
     * Radio requests.
     *
     * @return radio requests.
     */
    public RadioRequests radio() {
        return new RadioRequests();
    }

    /**
     * Auth requests.
     *
     * @return auth requests.
     */
    public AuthRequests auth() {
        return new AuthRequests();
    }

    /**
     * Search requests.
     *
     * @return search requests.
     */
    public SearchRequests search() {
        return new SearchRequests();
    }

    /**
     * Track requests.
     *
     * @return track requests.
     */
    public TrackRequests track() {
        return new TrackRequests(accessToken);
    }

    /**
     * User requests.
     *
     * @return user requests.
     */
    public UserRequests user() {
        return new UserRequests(accessToken);
    }
```

```
}
```

Даний клас створює об’єкти класів з пакету api.deezer.requests. Таким чином користувач бібліотеки може звертатись до цього класу, коли він забажає виконати запит до Deezer API.

Створимо юніт тести, щоб перевірити, що наші запити відповідають очікуваному формату Deezer.

```java
package api.deezer.requests;

import ...

class AlbumRequestsTest {
    DeezerApi deezerApi = new DeezerApi("accessToken");

    @Test
    void getById() {
        DeezerRequest<Album> request = deezerApi.album().getById(302127);
        assertEquals("https://api.deezer.com/album/302127", request.getUrl());
        assertEquals("get", request.getParams().get("request_method"));
    }

    @Test
    void getFans() {
        PaginationRequest<UserData> request = deezerApi.album().getFans(302127).limit(5).offset(2);
        assertEquals("https://api.deezer.com/album/302127/fans", request.getUrl());
        assertEquals("get", request.getParams().get("request_method"));
        assertEquals("5", request.getParams().get("limit"));
        assertEquals("2", request.getParams().get("offset"));
    }

    @Test
    void getTracks() {
        PaginationRequest<TrackData> request = deezerApi.album().getTracks(302127).limit(5).offset(2);
        assertEquals("https://api.deezer.com/album/302127/tracks", request.getUrl());
        assertEquals("get", request.getParams().get("request_method"));
        assertEquals("5", request.getParams().get("limit"));
        assertEquals("2", request.getParams().get("offset"));
    }
}
```

Рисунок 4.10 – Юніт-тести

Виконаємо тестування:

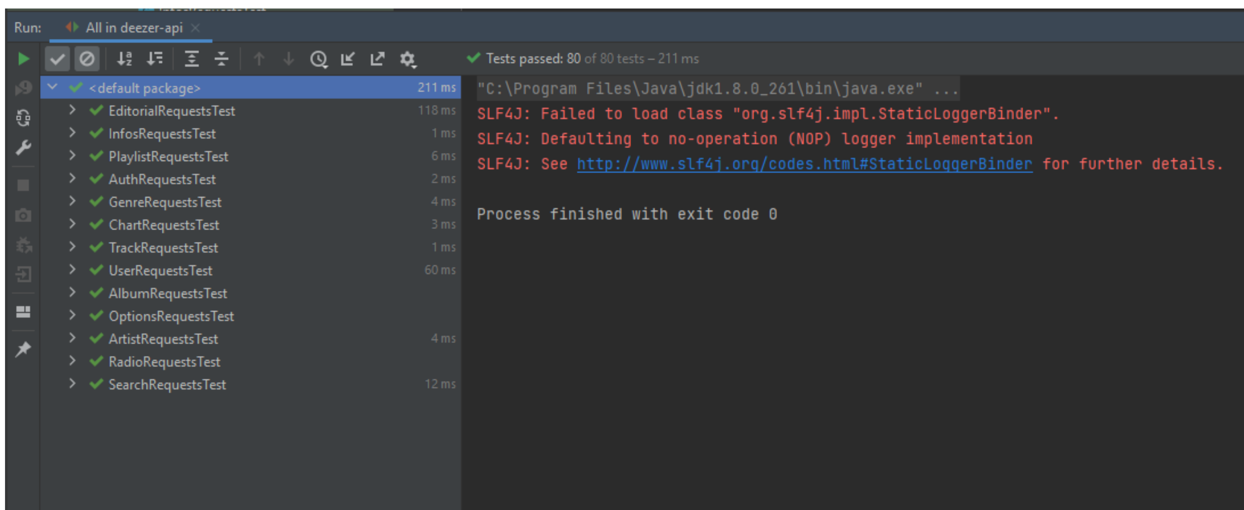


Рисунок 4.11 – Тестування бібліотеки

Опублікуємо наш проект на GitHub. Intellij Idea дозволяє зробити це у два кліки: VCS – Share Project on GitHub:

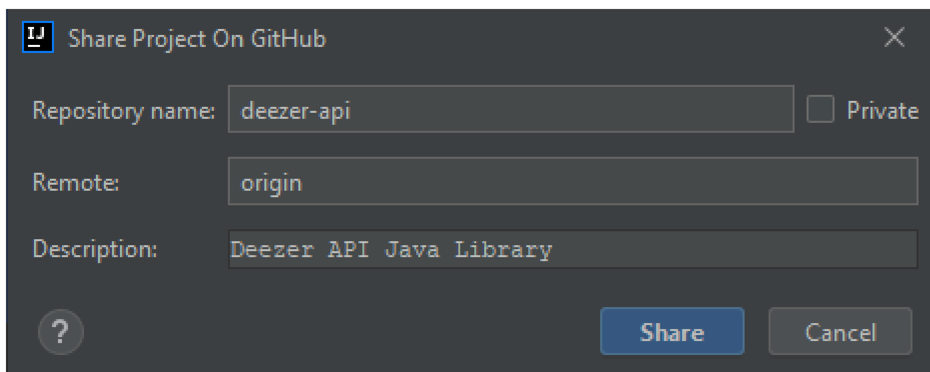


Рисунок 4.12 – Публікація проекту на GitHub

Тепер вихідний код бібліотеки доступний за посиланням: https://github.com/yvasyliev/deezer-api.

Нам залишилось створити проект у репозиторії Maven. Для цього ми створили Jira акаунт на ресурсі https://issues.sonatype.org/ та запит на створення проекту для com.github.yvasyliev:

⌄ Bot Central-OSSRH added a comment - 08/30/20 01:12 PM

com.github.yvasyliev has been prepared, now user(s) Vasyliev can:

- Deploy snapshot artifacts into repository https://oss.sonatype.org/content/repositories/snapshots
- Deploy release artifacts into the staging repository https://oss.sonatype.org/service/local/staging/deploy/maven2
- Release staged artifacts into repository 'Releases'

please comment on this ticket when you promoted your first release, thanks

Рисунок 4.13 – Створення репозиторію для com.github.yvasyliev

Створивши репозиторій ми готові до публікації нашої бібліотеки. Для цього необхідно перейти в кореневу директорії проекту та виконати Maven команду: mvn clean deploy:

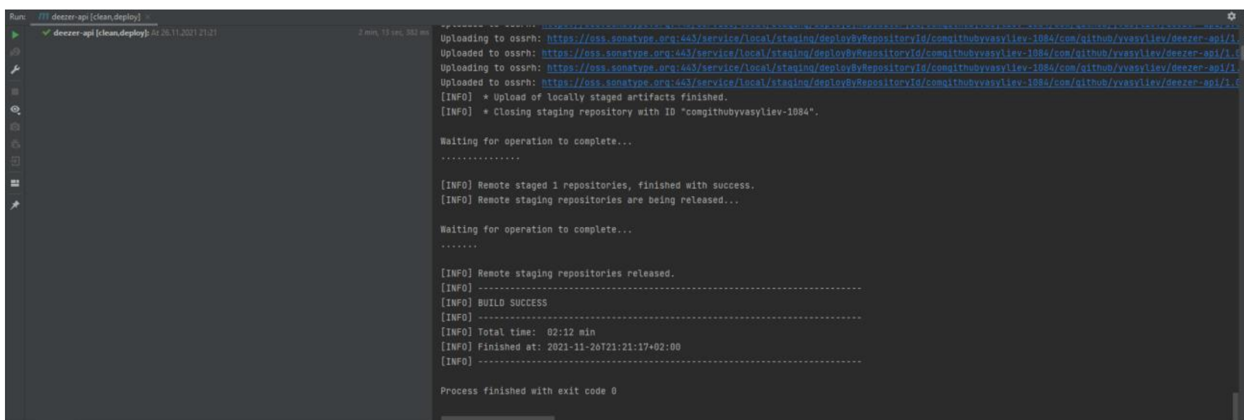


Рисунок 4.14 – Публікація бібліотеки

Тепер нашу бібліотеку можна знайти за посиланням: https://search.maven.org/artifact/com.github.yvasyliev/deezer-api/1.0.1/jar

## 4.3 Приклад застосування

Створимо новий проект Maven:

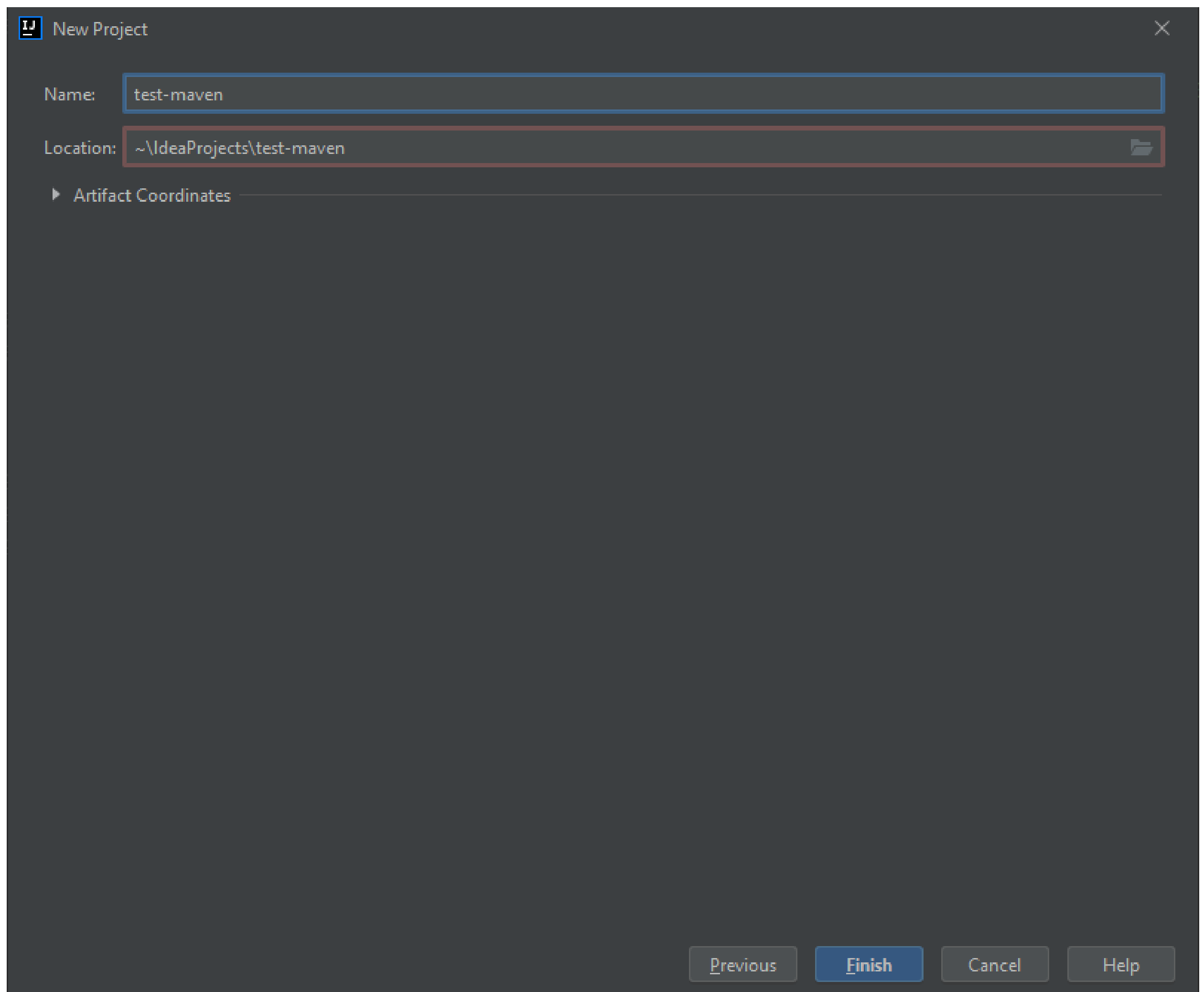


Рисунок 4.15 – Створення нового проекту

Включемо нашу бібліотеку у pom.xml:

```
<?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>

    <groupId>org.example</groupId>
    <artifactId>test-maven</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
```

```
        <dependency>
            <groupId>com.github.yvasyliev</groupId>
            <artifactId>deezer-api</artifactId>
            <version>1.0.2</version>
        </dependency>
    </dependencies>
</project>
```

Створимо клас Main у пакеті app з наступним вмістом:

```
package app;

import api.deezer.DeezerApi;
import api.deezer.exceptions.DeezerException;
import api.deezer.objects.data.AlbumData;
import api.deezer.objects.data.ArtistData;

public class Main {
    public static void main(String[] args) throws DeezerException {
        DeezerApi deezerApi = new DeezerApi();

        ArtistData queen =
deezerApi.search().searchArtist("queen").limit(1).execute();
        System.out.println(queen);

        int queenId = queen.getData().get(0).getId();

        ArtistData relatedArtist =
deezerApi.artist().getRelatedArtists(queenId).limit(1).execute();
        System.out.println(relatedArtist);

        AlbumData queenAlbums = deezerApi.artist().getAlbums(queenId).execute();
        System.out.println(queenAlbums);
    }
}
```

Запустимо додаток та подивимось на результат:

Рисунок 4.16 – Результат виконання

# ВИСНОВКИ

У результаті виконання даної роботи було проаналізовано структуру та приклади застосування архітектурного стилю Representational State Transfer. Він описує методи створення веб-додатків та способів взаємодії між ними. REST використовує лише HTTP протокол передачі даних та базується на використанні HTTP методів для отримання інформації, редагування та видалення. Найчастіше застосовуються наступні HTTP методи: GET, POST, PUT, DELETE. Як правило, дані передаються передаються у форматі HTML, XML або JSON. REST описує декілька обмежень, які повинні дотримуватись веб-додатки: клієнт-серверна архітектура, відсутність стану, кешування, однорідний інтерфейс, шари абстракції, запитування коду. Ті веб-додатки, що дотримуються згадані обмеження, називаються RESTful.

Також було досліджено інформацію, що надає сервіс стрімінга музики Deezer, про роботу з його відкритим REST API. Даний сервіс пропонує ряд методів для знаходження музичних записів, музичних альбомів, авторів, списків відтворення та іншої інформації. За допомогою даних методів є можливість створення власних клієнтів Deezer, а також розширювати можливості сервісу. Частина методів вимагають передачі спецільного access_token для взаємодії з даними користувача. Для отримання цього токену необхідно здійснити три кроки: зареєструвати наш додаток у системі Deezer, щоб отримати ідентифікаційні дані та сформувати правильне посилання; користувач повинен перейти за даним посиланням та ввести дані свого профілю (виконати вхід у систему Deezer); запросити дозвіл у користувача на доступ до даних його профілю, та отримати access_token, який згенерує Deezer.

Було розглянуто існуючу реалізацію подібного інтерфейсу - Spotify Web API Java. Даний проект з відкритим вихідним кодом реалізує інтерфейс звернень до системи стрімінга музики Spotify. Сервіс Spotify містить

аналогічні до Deezer REST API, за допомогою яких можна отримувати інформації про музичні записи та дані користувача сервісу. Процедура аутентифікації користувача та власного додатку досить подібна до аналогічної системи Deezer. Програмна реалізація базується на застосуванні принципів ООП для побудови відповідних HTTP запитів. У представленому прикладі ми побачили базову взаємодію з сервісом Spotify, і ми бачимо що така реалізація є інтуїтивною зрозумілою та зручною для створення власних додатків. Ці знання допомогли нам створити та розповсюдити власну бібліотеку.

На виході даної роботи ми отримали Java бібліотеку, за допомогою якої можна створювати користувацький клієнт Deezer. Бібліотека була створена з використанням Java версії 8 та фреймворком Maven. Було представлено об’єктну модель бібліотеки та сервіси взаємодії з Deezer API. Бібліотека уміє формувати правильні REST запити для отримання та редагування данних. Для того, щоб кожен користувач зміг у будь-який момент звернутись до даної бібліотеки, її опубліковано на віддаленому репозиторії – Maven Central. Цей репозиторій містить безліч користувацьких бібліотек та є найбільш популярним серед Java-розробників. В роботі було оглянуто процедуру завантаження та вимоги до бібліотек, які заплановано до відкритого опублікування. Також було продемонстровано приклад застосування бібліотеки. Було створено окремий додаток, до якого підключили бібліотеку Deezer API, було виконано декілька запитів та продемонстровано результати виконання.

# СПИСОК ЛІТЕРАТУРИ

1. World Wide Web [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/World_Wide_Web.
2. Hypertext Transfer Protocol [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol.
3. Karanam R. Introduction to REST API - RESTful Web Services [Електронний ресурс] / Ranga Karanam. – 2019. – Режим доступу до ресурсу: https://www.springboottutorial.com/introduction-to-rest-api.
4. Representational state transfer [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Representational_state_transfer.
5. REST APIs [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/cloud/learn/rest-apis.
6. Guide to uploading artifacts to the Central Repository [Електронний ресурс] – Режим доступу до ресурсу: https://maven.apache.org/repository/guide-central-repository-upload.html.
7. Deezer API [Електронний ресурс] – Режим доступу до ресурсу: https://developers.deezer.com/api.
8. Deploying to OSSRH with Apache Maven - Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://central.sonatype.org/publish/publish-maven/.
9. Spotify Web API Java [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/spotify-web-api-java/spotify-web-api-java.

# ДОДАТОК – КОД ПРОГРАМИ

pom.xml

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

    <groupId>com.github.yvasyliev</groupId>
    <artifactId>deezer-api</artifactId>
    <packaging>jar</packaging>
    <version>1.0.3</version>
    <name>Deezer API Java Library</name>
    <description>A Java implementation of Deezer API.</description>
    <url>https://github.com/yvasyliev/deezer-api</url>

    <licenses>
        <license>
            <name>MIT License</name>
            <url>https://www.opensource.org/licenses/mit-license.php</url>
        </license>
    </licenses>

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

    <dependencies>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>1.7.32</version>
        </dependency>
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.8.9</version>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
```

```
            <version>5.8.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <distributionManagement>
        <snapshotRepository>
            <id>ossrh</id>
            <url>https://oss.sonatype.org/content/repositories/snapshots</url>
        </snapshotRepository>
        <repository>
            <id>ossrh</id>
            <url>https://oss.sonatype.org/service/local/staging/deploy/maven2/</url>
        </repository>
    </distributionManagement>

    <build>
        <resources>
            <resource>
                <directory>src/main/resources</directory>
            </resource>
        </resources>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-resources-plugin</artifactId>
                <version>3.0.1</version>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
            </plugin>
            <plugin>
                <groupId>org.sonatype.plugins</groupId>
                <artifactId>nexus-staging-maven-plugin</artifactId>
                <version>1.6.7</version>
                <extensions>true</extensions>
                <configuration>
                    <serverId>ossrh</serverId>
                    <nexusUrl>https://oss.sonatype.org/</nexusUrl>
                    <autoReleaseAfterClose>true</autoReleaseAfterClose>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-release-plugin</artifactId>
                <version>3.0.0-M1</version>
```

```
            <configuration>
                <autoVersionSubmodules>true</autoVersionSubmodules>
                <useReleaseProfile>false</useReleaseProfile>
                <releaseProfiles>release</releaseProfiles>
                <goals>deploy</goals>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-source-plugin</artifactId>
            <version>2.2.1</version>
            <executions>
                <execution>
                    <id>attach-sources</id>
                    <goals>
                        <goal>jar-no-fork</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-javadoc-plugin</artifactId>
            <version>2.9.1</version>
            <executions>
                <execution>
                    <id>attach-javadocs</id>
                    <goals>
                        <goal>jar</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-gpg-plugin</artifactId>
            <version>1.5</version>
            <executions>
                <execution>
                    <id>sign-artifacts</id>
                    <phase>verify</phase>
                    <goals>
                        <goal>sign</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <!-- To build jar with dependencies -->
```

```
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-assembly-plugin</artifactId>
                <version>3.3.0</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                        <configuration>
                            <descriptorRefs>
                                <descriptorRef>jar-with-dependencies</descriptorRef>
                            </descriptorRefs>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.19.1</version>
                <dependencies>
                    <dependency>
                        <groupId>org.junit.platform</groupId>
                        <artifactId>junit-platform-surefire-provider</artifactId>
                        <version>1.1.0</version>
                    </dependency>
                </dependencies>
            </plugin>
        </plugins>
    </build>

    <scm>
        <connection>scm:git:ssh://github.com/yvasyliev/deezer-api.git</connection>
        <developerConnection>scm:git:ssh://github.com/yvasyliev/deezer-
api.git</developerConnection>
        <url>https://github.com/yvasyliev/deezer-api</url>
    </scm>

    <developers>
        <developer>
            <name>Yevhen Vasyliev</name>
            <email>ye.vasyliev@gmail.com</email>
            <organization>com.github.yvasyliev</organization>
            <organizationUrl>https://github.com/yvasyliev</organizationUrl>
        </developer>
    </developers>
```

```
</project>
```

ArtistRequests.java

```
package api.deezer.requests;

import api.deezer.http.impl.DeezerGetRequest;
import api.deezer.http.impl.DeezerRequest;
import api.deezer.http.impl.PaginationRequest;
import api.deezer.objects.Artist;
import api.deezer.objects.data.AlbumData;
import api.deezer.objects.data.ArtistData;
import api.deezer.objects.data.PlaylistData;
import api.deezer.objects.data.TrackData;
import api.deezer.objects.data.UserData;

/**
 * Requests related to artists.
 */
public class ArtistRequests extends DeezerRequests {
    /**
     * Get artists by ID.
     *
     * @param artistId artist ID.
     * @return {@link Artist} object.
     */
    public DeezerRequest<Artist> getById(int artistId) {
        return new DeezerGetRequest<>(property("artist.get", artistId),
Artist.class);
    }

    /**
     * Gets artist's top five favourite tracks.
     *
     * @param artistId artist ID.
     * @return artist's top five favourite tracks.
     */
    public PaginationRequest<TrackData> getArtistTopFiveTracks(int artistId) {
        return new PaginationRequest<>(property("artist.top", artistId),
TrackData.class);
    }

    /**
     * Gets artist's albums.
     *
     * @param artistId artist ID.
     * @return artist's albums.
     */
    public PaginationRequest<AlbumData> getAlbums(int artistId) {
```

```
        return new PaginationRequest<>(property("artist.albums", artistId),
AlbumData.class);
    }

    /**
     * Gets artist fans.
     *
     * @param artistId artist ID.
     * @return artist fans.
     */
    public PaginationRequest<UserData> getFans(int artistId) {
        return new PaginationRequest<>(property("artist.fans", artistId),
UserData.class);
    }

    /**
     * Gets related artists.
     *
     * @param artistId artist ID.
     * @return related artists.
     */
    public PaginationRequest<ArtistData> getRelatedArtists(int artistId) {
        return new PaginationRequest<>(property("artist.related", artistId),
ArtistData.class);
    }

    /**
     * Gets artist radio.
     *
     * @param artistId artist ID.
     * @return artist radio.
     */
    public PaginationRequest<TrackData> getRadio(int artistId) {
        return new PaginationRequest<>(property("artist.radio", artistId),
TrackData.class);
    }

    /**
     * Gets artist's playlists.
     *
     * @param artistId artist ID.
     * @return artist's playlists.
     */
    public PaginationRequest<PlaylistData> getPlaylists(int artistId) {
        return new PaginationRequest<>(property("artist.playlists", artistId),
PlaylistData.class);
    }
}
```

AuthRequests.java

```
package api.deezer.requests;

import api.deezer.converters.AccessTokenConverter;
import api.deezer.converters.Converter;
import api.deezer.converters.PermissionsConverter;
import api.deezer.exceptions.DeezerException;
import api.deezer.http.impl.DeezerGetRequest;
import api.deezer.http.impl.DeezerRequest;
import api.deezer.http.utils.URLParamsEncoder;
import api.deezer.objects.AccessToken;
import api.deezer.objects.Permission;
import api.deezer.properties.DeezerProperties;

import java.io.UnsupportedEncodingException;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * Authorization requests.
 */
public class AuthRequests extends DeezerRequests {
    /**
     * Converts list of {@link Permission} into comma separated values.
     */
    private final Converter<List<Permission>, String> permissionsConverter = new
PermissionsConverter();

    /**
     * Gets login url for current application.
     *
     * @param appId       application ID.
     * @param redirectUri redirect URI.
     * @param perms       permissions.
     * @return full login URL.
     * @throws DeezerException if errors occur.
     */
    public String getLoginUrl(int appId, String redirectUri, Permission... perms)
throws DeezerException {
        return getLoginUrl(appId, redirectUri, Arrays.asList(perms));
    }

    /**
     * Gets login url for current application.
     *
```

```
     * @param appId       application ID.
     * @param redirectUri redirect URI.
     * @param perms       permissions.
     * @return full login URL.
     * @throws DeezerException if errors occur.
     */
    public String getLoginUrl(int appId, String redirectUri, List<Permission> perms)
throws DeezerException {
        try {
            return property(
                    "auth.url",
                    appId,
                    URLParamsEncoder.encode(redirectUri),
                    URLParamsEncoder.encode(permissionsConverter.covert(perms))
            );
        } catch (UnsupportedEncodingException e) {
            throw new DeezerException(e);
        }
    }

    /**
     * Gets Deezer <i>access_token</i>.
     *
     * @param appId  application ID.
     * @param secret application secret string.
     * @param code   login code.
     * @return Deezer <i>access_token</i>.
     */
    public DeezerRequest<AccessToken> getAccessToken(int appId, String secret, String
code) {
        Map<String, String> params = new HashMap<>();
        params.put("app_id", String.valueOf(appId));
        params.put("secret", secret);
        params.put("code", code);
        params.put("output", "json"); // TODO: 31.10.2021 why does
https://connect.deezer.com/oauth/access_token.php always return a string format
"access_token=${access_token}&expires=${expires}" ignoring "output=json" parameter?

        DeezerGetRequest<AccessToken> request = new DeezerGetRequest<>(
                DeezerProperties.getProperty("auth.access_token"),
                params,
                AccessToken.class
        );
        request.setResponseConverter(new AccessTokenConverter());
        return request;
    }
}
```

ChartRequests.java

```
package api.deezer.requests;

import api.deezer.http.impl.DeezerGetRequest;
import api.deezer.http.impl.DeezerRequest;
import api.deezer.http.impl.PaginationRequest;
import api.deezer.objects.Chart;
import api.deezer.objects.data.AlbumData;
import api.deezer.objects.data.ArtistData;
import api.deezer.objects.data.PlaylistData;
import api.deezer.objects.data.PodcastData;
import api.deezer.objects.data.TrackData;
import api.deezer.properties.DeezerProperties;

/**
 * Requests related to charts.
 */
public class ChartRequests extends ArtistRequests {
    /**
     * Get all charts.
     *
     * @return all charts.
     */
    public DeezerRequest<Chart> getAll() {
        return new DeezerGetRequest<>(DeezerProperties.getProperty("chart.get"),
Chart.class);
    }

    /**
     * Gets top tracks.
     *
     * @return top tracks.
     */
    public PaginationRequest<TrackData> getTopTracks() {
        return new PaginationRequest<>(DeezerProperties.getProperty("chart.tracks"),
TrackData.class);
    }

    /**
     * Gets top albums.
     *
     * @return top albums.
     */
    public PaginationRequest<AlbumData> getTopAlbums() {
        return new PaginationRequest<>(DeezerProperties.getProperty("chart.albums"),
AlbumData.class);
    }

    /**
```

```
     * Gets top artists.
     *
     * @return top artists.
     */
    public PaginationRequest<ArtistData> getTopArtists() {
        return new PaginationRequest<>(DeezerProperties.getProperty("chart.artists"), 
ArtistData.class);
    }

    /**
     * Gets top playlists.
     *
     * @return top playlists.
     */
    public PaginationRequest<PlaylistData> getTopPlaylists() {
        return new 
PaginationRequest<>(DeezerProperties.getProperty("chart.playlists"), 
PlaylistData.class);
    }

    /**
     * Gets top podcasts.
     *
     * @return top podcasts.
     */
    public PaginationRequest<PodcastData> getTopPodcasts() {
        return new 
PaginationRequest<>(DeezerProperties.getProperty("chart.podcasts"), 
PodcastData.class);
    }
}
```

EditorialRequests.java

```
package api.deezer.requests;

import api.deezer.http.impl.DeezerGetRequest;
import api.deezer.http.impl.DeezerRequest;
import api.deezer.http.impl.PaginationRequest;
import api.deezer.objects.Chart;
import api.deezer.objects.data.AlbumData;
import api.deezer.properties.DeezerProperties;

/**
 * Requests related to editorial.
 */
public class EditorialRequests extends DeezerRequests {
    /**
     * Gets selected albums.
```

```
     *
     * @return selected albums.
     */
    public PaginationRequest<AlbumData> getSelectedAlbums() {
        return new
PaginationRequest<>(DeezerProperties.getProperty("editorial.selection"),
AlbumData.class);
    }

    /**
     * Get chart.
     *
     * @return chart.
     */
    public DeezerRequest<Chart> getChart() {
        return new
DeezerGetRequest<>(DeezerProperties.getProperty("editorial.charts"), Chart.class);
    }

    /**
     * Gets new releases.
     *
     * @return new releases.
     */
    public PaginationRequest<AlbumData> getNewReleases() {
        return new
PaginationRequest<>(DeezerProperties.getProperty("editorial.releases"),
AlbumData.class);
    }
}
```

GenreRequests.java

```
package api.deezer.requests;

import api.deezer.http.impl.DeezerGetRequest;
import api.deezer.http.impl.DeezerRequest;
import api.deezer.http.impl.PaginationRequest;
import api.deezer.objects.data.ArtistData;
import api.deezer.objects.data.GenreData;
import api.deezer.objects.data.RadioData;
import api.deezer.properties.DeezerProperties;

/**
 * Requests related to genres.
 */
public class GenreRequests extends DeezerRequests {
    /**
     * Gets all genres.
```

```
     *
     * @return all genres.
     */
    public DeezerRequest<GenreData> getAll() {
        return new DeezerGetRequest<>(DeezerProperties.getProperty("genre.all"),
GenreData.class);
    }

    /**
     * Gets artists by genre.
     *
     * @param genreId genre ID.
     * @return list of artists.
     */
    public PaginationRequest<ArtistData> getArtistsByGenreId(int genreId) {
        return new PaginationRequest<>(property("genre.artists", genreId),
ArtistData.class);
    }

    /**
     * Gets radios by genre.
     *
     * @param genreId genre ID.
     * @return list of radios.
     */
    public PaginationRequest<RadioData> getRadiosByGenreId(int genreId) {
        return new PaginationRequest<>(property("genre.radios", genreId),
RadioData.class);
    }
}
```

InfosRequests.java

```
package api.deezer.requests;

import api.deezer.http.impl.DeezerGetRequest;
import api.deezer.http.impl.DeezerRequest;
import api.deezer.objects.Infos;
import api.deezer.properties.DeezerProperties;

/**
 * Requests related to infos.
 */
public class InfosRequests extends DeezerRequests {
    /**
     * Gets infos.
     *
     * @return infos.
     */
```

```
    public DeezerRequest<Infos> get() {
        return new DeezerGetRequest<>(DeezerProperties.getProperty("infos.get"),
Infos.class);
    }
}
```

OptionsRequests.java

```
package api.deezer.requests;

import api.deezer.http.impl.DeezerGetRequest;
import api.deezer.http.impl.DeezerRequest;
import api.deezer.objects.Options;
import api.deezer.properties.DeezerProperties;

/**
 * Requests related to options.
 */
public class OptionsRequests extends DeezerRequests {
    /**
     * Gets options.
     *
     * @return options.
     */
    public DeezerRequest<Options> get() {
        return new DeezerGetRequest<>(DeezerProperties.getProperty("options.get"),
Options.class);
    }
}
```

PlaylistRequests.java

```
package api.deezer.requests;

import api.deezer.converters.Converter;
import api.deezer.converters.ListConverter;
import api.deezer.converters.TracksDataConverter;
import api.deezer.http.impl.*;
import api.deezer.objects.Playlist;
import api.deezer.objects.data.TrackData;
import api.deezer.objects.data.UserData;

import java.util.Arrays;
import java.util.List;
import java.util.Map;

/**
 * Requests related to playlists.
 */
```

```
public class PlaylistRequests extends DeezerRequests {
    /**
     * Converts list of integers into comma separated values.
     */
    private final Converter<List<Integer>, String> listConverter = new
ListConverter<>();

    public PlaylistRequests(String accessToken) {
        super(accessToken);
    }

    /**
     * Gets playlist.
     *
     * @param playlistId playlist ID.
     * @return {@link Playlist} object.
     */
    public DeezerRequest<Playlist> getById(long playlistId) {
        return new DeezerGetRequest<>(property("playlist.get", playlistId),
Playlist.class);
    }

    /**
     * Marks playlist as seen.
     *
     * @param playlistId playlist ID.
     * @return <i>true</i> if was successful.
     */
    public DeezerRequest<Boolean> markAsSeen(long playlistId) {
        return new DeezerPostRequest<>(property("playlist.seen", playlistId),
accessTokenParam(), Boolean.class);
    }

    /**
     * Gets playlist fans.
     *
     * @param playlistId playlist ID.
     * @return playlist fans.
     */
    public PaginationRequest<UserData> getFans(long playlistId) {
        return new PaginationRequest<>(property("playlist.fans", playlistId),
UserData.class);
    }

    /**
     * Gets playlist tracks.
     *
     * @param playlistId playlist ID.
```

```
     * @return playlist tracks.
     */
    public PaginationRequest<TrackData> getTracks(long playlistId) {
        return new PaginationRequest<>(property("playlist.tracks", playlistId),
TrackData.class);
    }

    /**
     * Gets playlist radio.
     *
     * @param playlistId playlist ID.
     * @return playlist radio.
     */
    public PaginationRequest<TrackData> getRadio(long playlistId) {
        PaginationRequest<TrackData> paginationRequest = new PaginationRequest<>(
                property("playlist.radio", playlistId),
                TrackData.class
        );
        paginationRequest.setResponseConverter(new TracksDataConverter());
        return paginationRequest;
    }

    /**
     * Adds tracks to playlist
     *
     * @param playlistId playlist ID.
     * @param trackIds   tracks IDs.
     * @return <i>true</i> if successful.
     */
    public DeezerRequest<Boolean> addTracks(int playlistId, Integer... trackIds) {
        return addTracks(playlistId, Arrays.asList(trackIds));
    }

    /**
     * Adds tracks to playlist.
     *
     * @param playlistId playlist ID.
     * @param trackIds   tracks IDs.
     * @return <i>true</i> if successful.
     */
    public DeezerRequest<Boolean> addTracks(int playlistId, List<Integer> trackIds) {
        Map<String, String> params = accessTokenParam();
        params.put("songs", listConverter.covert(trackIds));
        return new DeezerPostRequest<>(
                property("playlist.tracks", playlistId),
                params,
                Boolean.class
        );
```

```
    }

    /**
     * Orders tracks in playlist.
     *
     * @param playlistId playlist ID.
     * @param trackIds   tracks IDs.
     * @return <i>true</i> if successful.
     */
    public DeezerRequest<Boolean> orderTracks(int playlistId, Integer... trackIds) {
        return orderTracks(playlistId, Arrays.asList(trackIds));
    }

    /**
     * Orders tracks in playlist.
     *
     * @param playlistId playlist ID.
     * @param trackIds   tracks IDs.
     * @return <i>true</i> if successful.
     */
    public DeezerRequest<Boolean> orderTracks(int playlistId, List<Integer> trackIds)
{
        Map<String, String> params = accessTokenParam();
        params.put("order", listConverter.covert(trackIds));
        return new DeezerPostRequest<>(
                property("playlist.tracks", playlistId),
                params,
                Boolean.class
        );
    }

    /**
     * Deletes playlist.
     *
     * @param playlistId playlist ID.
     * @return <i>true</i> if successful.
     */
    public DeezerRequest<Boolean> delete(long playlistId) {
        return new DeezerDeleteRequest<>(
                property("playlist.get", playlistId),
                accessTokenParam(),
                Boolean.class
        );
    }

    /**
     * Removes tracks from playlist.
     *
```

```
     * @param playlistId playlist ID.
     * @param trackIds   track IDs.
     * @return <i>true</i> if successful.
     */
    public DeezerRequest<Boolean> removeTracks(long playlistId, Integer... trackIds)
{
        return removeTracks(playlistId, Arrays.asList(trackIds));
    }

    /**
     * Removes tracks from playlist.
     *
     * @param playlistId playlist ID.
     * @param trackIds   track IDs.
     * @return <i>true</i> if successful.
     */
    public DeezerRequest<Boolean> removeTracks(long playlistId, List<Integer>
trackIds) {
        Map<String, String> params = accessTokenParam();
        params.put("songs", listConverter.covert(trackIds));
        return new DeezerDeleteRequest<>(
                property("playlist.get", playlistId),
                params,
                Boolean.class
        );
    }

    /**
     * Creates <i>access_token</i> param.
     *
     * @return <i>access_token</i> param.
     */
    private Map<String, String> accessTokenParam() {
        return params(entry("access_token", getAccessToken()));
    }
}
```

RadioRequests.java

```
package api.deezer.requests;

import api.deezer.http.impl.DeezerGetRequest;
import api.deezer.http.impl.DeezerRequest;
import api.deezer.objects.data.RadioData;
import api.deezer.properties.DeezerProperties;

/**
 * Requests related to radio.
 */
```

```
public class RadioRequests extends DeezerRequests {
    /**
     * Gets radio.
     *
     * @return radio.
     */
    public DeezerRequest<RadioData> get() {
        return new DeezerGetRequest<>(DeezerProperties.getProperty("radio.get"),
RadioData.class);
    }
}
```

SearchRequests.java

```
package api.deezer.requests;

import api.deezer.http.impl.AdvancedSearchRequest;
import api.deezer.http.impl.SearchRequest;
import api.deezer.objects.data.AlbumData;
import api.deezer.objects.data.ArtistData;
import api.deezer.objects.data.PlaylistData;
import api.deezer.objects.data.RadioData;
import api.deezer.objects.data.TrackData;
import api.deezer.objects.data.UserData;
import api.deezer.properties.DeezerProperties;

/**
 * Requests related to search.
 */
public class SearchRequests extends DeezerRequests {
    /**
     * Searches albums.
     *
     * @param q album name.
     * @return list of albums.
     */
    public SearchRequest<AlbumData> searchAlbum(String q) {
        return new SearchRequest<>(DeezerProperties.getProperty("search.album"), q,
AlbumData.class);
    }

    /**
     * Searches albums.
     *
     * @return list of albums.
     */
    public AdvancedSearchRequest<AlbumData> searchAlbum() {
```

```
        return new
AdvancedSearchRequest<>(DeezerProperties.getProperty("search.album"),
AlbumData.class);
    }

    /**
     * Searches artists.
     *
     * @param q artist name.
     * @return list of artists.
     */
    public SearchRequest<ArtistData> searchArtist(String q) {
        return new SearchRequest<>(DeezerProperties.getProperty("search.artist"), q,
ArtistData.class);
    }

    /**
     * Searches artists.
     *
     * @return list of artists.
     */
    public AdvancedSearchRequest<ArtistData> searchArtist() {
        return new
AdvancedSearchRequest<>(DeezerProperties.getProperty("search.artist"),
ArtistData.class);
    }

    /**
     * Searches playlists.
     *
     * @param q playlist name.
     * @return list of playlists.
     */
    public SearchRequest<PlaylistData> searchPlaylist(String q) {
        return new SearchRequest<>(DeezerProperties.getProperty("search.playlist"),
q, PlaylistData.class);
    }

    /**
     * Searches playlists.
     *
     * @return list of playlists.
     */
    public AdvancedSearchRequest<PlaylistData> searchPlaylist() {
        return new
AdvancedSearchRequest<>(DeezerProperties.getProperty("search.playlist"),
PlaylistData.class);
    }
```

```
    /**
     * Searches radio.
     *
     * @param q radio name.
     * @return list of radios.
     */
    public SearchRequest<RadioData> searchRadio(String q) {
        return new SearchRequest<>(DeezerProperties.getProperty("search.radio"), q,
RadioData.class);
    }

    /**
     * Searches radio.
     *
     * @return list of radios.
     */
    public AdvancedSearchRequest<RadioData> searchRadio() {
        return new
AdvancedSearchRequest<>(DeezerProperties.getProperty("search.radio"),
RadioData.class);
    }

    /**
     * Searches tracks.
     *
     * @param q track name.
     * @return list of tracks.
     */
    public SearchRequest<TrackData> searchTrack(String q) {
        return new SearchRequest<>(DeezerProperties.getProperty("search.track"), q,
TrackData.class);
    }

    /**
     * Searches tracks.
     *
     * @return list of tracks.
     */
    public AdvancedSearchRequest<TrackData> searchTrack() {
        return new
AdvancedSearchRequest<>(DeezerProperties.getProperty("search.track"),
TrackData.class);
    }

    /**
     * Searches users.
     *
     * @param q username.
     * @return list of users.
```

```
     */
    public SearchRequest<UserData> searchUser(String q) {
        return new SearchRequest<>(DeezerProperties.getProperty("search.user"), q,
UserData.class);
    }

    /**
     * Searches users.
     *
     * @return list of users.
     */
    public AdvancedSearchRequest<UserData> searchUser() {
        return new
AdvancedSearchRequest<>(DeezerProperties.getProperty("search.user"), UserData.class);
    }
    // TODO: 01.11.2021 search.history
}
```

TrackRequests.java

```
package api.deezer.requests;

import api.deezer.http.impl.DeezerDeleteRequest;
import api.deezer.http.impl.DeezerGetRequest;
import api.deezer.http.impl.DeezerRequest;
import api.deezer.objects.Track;

/**
 * Requests related to tracks.
 */
public class TrackRequests extends DeezerRequests {
    public TrackRequests(String accessToken) {
        super(accessToken);
    }

    /**
     * Gets track by ID.
     *
     * @param id track ID.
     * @return track.
     */
    public DeezerRequest<Track> getById(int id) {
        return new DeezerGetRequest<>(property("track.get", id), Track.class);
    }

    /**
     * Deletes user personal track.
     *
     * @param trackId track ID.
```

```
     * @return <i>true</i> if successful.
     */
    public DeezerRequest<Boolean> delete(int trackId) {
        return new DeezerDeleteRequest<>(
                property("track.get", trackId),
                params(entry("access_token", getAccessToken())),
                Boolean.class
        );
    }
}
```

UserRequests.java

```
package api.deezer.requests;

import api.deezer.converters.Converter;
import api.deezer.converters.ListConverter;
import api.deezer.http.impl.*;
import api.deezer.objects.GetPermissionsResponse;
import api.deezer.objects.Options;
import api.deezer.objects.SendNotificationResponse;
import api.deezer.objects.User;
import api.deezer.objects.data.*;

import java.util.Arrays;
import java.util.List;
import java.util.Map;

// TODO: 01.11.2021 user / charts - what is the implementation?

/**
 * Requests related to user.
 */
public class UserRequests extends DeezerRequests {
    /**
     * Converts list of integers to comma separated values.
     */
    private final Converter<List<Integer>, String> listConverter = new
ListConverter<Integer>();

    public UserRequests(String accessToken) {
        super(accessToken);
    }

    /**
     * Gets user by ID.
     *
     * @param userId user ID.
```

```
     * @return user.
     */
    public DeezerRequest<User> getById(long userId) {
        return new DeezerGetRequest<>(
                property("user.get", userId),
                User.class
        );
    }

    /**
     * Gets current user.
     *
     * @return current user.
     */
    public DeezerRequest<User> getMe() {
        return new DeezerGetRequest<>(
                property("user.get", "me"),
                accessTokenParam(),
                User.class
        );
    }

    /**
     * Gets user's favourite albums.
     *
     * @param userId user ID.
     * @return user's favourite albums.
     */
    public PaginationRequest<AlbumData> getFavouriteAlbums(long userId) {
        return new PaginationRequest<>(
                property("user.albums", userId),
                AlbumData.class
        );
    }

    /**
     * Gets user's favourite albums.
     *
     * @param userId user ID.
     * @return user's favourite albums.
     */
    public PaginationRequest<ArtistData> getFavouriteArtists(long userId) {
        return new PaginationRequest<>(
                property("user.artists", userId),
                ArtistData.class
        );
    }
```

```
    /**
     * Gets user's flow.
     *
     * @return user's flow.
     */
    public PaginationRequest<TrackData> getFlow() {
        return new PaginationRequest<>(
                property("user.flow", "me"),
                accessTokenParam(),
                TrackData.class
        );
    }

    /**
     * Gets user's followings.
     *
     * @param userId user ID.
     * @return user's followings.
     */
    public PaginationRequest<UserData> getFollowings(long userId) {
        return new PaginationRequest<>(
                property("user.followings", userId),
                UserData.class
        );
    }

    /**
     * Gets user's followers.
     *
     * @param userId user ID.
     * @return user's followers.
     */
    public PaginationRequest<UserData> getFollowers(long userId) {
        return new PaginationRequest<>(
                property("user.followers", userId),
                UserData.class
        );
    }

    /**
     * Gets current user's history.
     *
     * @return current user's history.
     */
    public PaginationRequest<TrackData> getMyHistory() {
        return new PaginationRequest<>(
                property("user.history", "me"),
                accessTokenParam(),
```

```
                TrackData.class
        );
    }

    // TODO: 03.11.2021 is it working?

    /**
     * Sends notification to user.
     *
     * @param message notification message.
     * @return request status.
     */
    public DeezerRequest<SendNotificationResponse> sendNotification(String message) {
        return new DeezerPostRequest<>(
                property("user.notifications"),
                accessTokenParam(),
                SendNotificationResponse.class
        );
    }

    /**
     * Gets user's permissions.
     *
     * @return user's permissions.
     */
    public DeezerRequest<GetPermissionsResponse> getPermissions() {
        return new DeezerGetRequest<>(
                property("user.permissions"),
                accessTokenParam(),
                GetPermissionsResponse.class
        );
    }

    /**
     * Gets user's options.
     *
     * @return user's options.
     */
    public DeezerRequest<Options> getOptions() {
        return new DeezerGetRequest<>(
                property("user.options"),
                accessTokenParam(),
                Options.class
        );
    }

    /**
     * Gets user's songs.
```

```
     *
     * @return user's songs.
     */
    public PaginationRequest<TrackData> getPersonalSongs() {
        return new PaginationRequest<>(
                property("user.personal"),
                accessTokenParam(),
                TrackData.class
        );
    }

    /**
     * Gets user's playlists.
     *
     * @param userId user ID.
     * @return user's playlists.
     */
    public PaginationRequest<PlaylistData> getPlaylists(long userId) {
        return new PaginationRequest<>(
                property("user.playlists", userId),
                PlaylistData.class
        );
    }

    /**
     * Gets user's favourite radios.
     *
     * @param userId user ID.
     * @return user's favourite radios.
     */
    public PaginationRequest<RadioData> getFavouriteRadios(long userId) {
        return new PaginationRequest<>(
                property("user.radios", userId),
                RadioData.class
        );
    }

    /**
     * Gets user's recommended albums.
     *
     * @return user's recommended albums.
     */
    public PaginationRequest<AlbumData> getRecommendedAlbums() {
        return new PaginationRequest<>(
                property("recommendations.albums"),
                accessTokenParam(),
                AlbumData.class
        );
```

```
    }

    /**
     * Gets user's recommended releases.
     *
     * @return user's recommended releases.
     */
    public PaginationRequest<AlbumData> getRecommendedReleases() {
        return new PaginationRequest<>(
                property("recommendations.releases"),
                accessTokenParam(),
                AlbumData.class
        );
    }

    /**
     * Gets user's recommended releases.
     *
     * @return user's recommended releases.
     */
    public PaginationRequest<ArtistData> getRecommendedArtists() {
        return new PaginationRequest<>(
                property("recommendations.artists"),
                accessTokenParam(),
                ArtistData.class
        );
    }

    /**
     * Gets user's recommended playlists.
     *
     * @return user's recommended playlists.
     */
    public PaginationRequest<PlaylistData> getRecommendedPlaylists() {
        return new PaginationRequest<>(
                property("recommendations.playlists"),
                accessTokenParam(),
                PlaylistData.class
        );
    }

    /**
     * Gets user's recommended tracks.
     *
     * @return user's recommended tracks.
     */
    public PaginationRequest<TrackData> getRecommendedTracks() {
        return new PaginationRequest<>(
```

```
                property("recommendations.tracks"),
                accessTokenParam(),
                TrackData.class
        );
    }

    /**
     * Gets user's recommended radios.
     *
     * @return user's recommended radios.
     */
    public PaginationRequest<RadioData> getRecommendedRadios() {
        return new PaginationRequest<>(
                property("recommendations.radios"),
                accessTokenParam(),
                RadioData.class
        );
    }

    /**
     * Gets user's favourite tracks.
     *
     * @param userId user ID.
     * @return user's favourite tracks.
     */
    public PaginationRequest<TrackData> getFavouriteTracks(long userId) {
        return new PaginationRequest<>(
                property("user.tracks", userId),
                TrackData.class
        );
    }

    /**
     * Adds album to user library.
     *
     * @param albumId album ID.
     * @return <i>true</i> if was successful.
     */
    public DeezerRequest<Boolean> addAlbumToLibrary(int albumId) {
        Map<String, String> params = accessTokenParam();
        params.put("album_id", String.valueOf(albumId));
        return new DeezerPostRequest<>(
                property("user.albums", "me"),
                params,
                Boolean.class
        );
    }
```

```
/**
 * Adds artist to user favourites.
 *
 * @param artistId artist ID.
 * @return <i>true</i> if was successful.
 */
public DeezerRequest<Boolean> addArtistToFavourites(int artistId) {
    Map<String, String> params = accessTokenParam();
    params.put("artist_id", String.valueOf(artistId));
    return new DeezerPostRequest<>(
            property("user.artists", "me"),
            params,
            Boolean.class
    );
}

/**
 * Follows a user.
 *
 * @param userId user ID to follow.
 * @return <i>true</i> if was successful.
 */
public DeezerRequest<Boolean> follow(int userId) {
    Map<String, String> params = accessTokenParam();
    params.put("user_id", String.valueOf(userId));
    return new DeezerPostRequest<>(
            property("user.followings", "me"),
            params,
            Boolean.class
    );
}

/**
 * Creates a playlist.
 *
 * @param playlistTitle playlist title.
 * @return <i>true</i> if was successful.
 */
public DeezerRequest<Boolean> createPlaylist(String playlistTitle) {
    Map<String, String> params = accessTokenParam();
    params.put("title", String.valueOf(playlistTitle));
    return new DeezerPostRequest<>(
            property("user.playlists", "me"),
            params,
            Boolean.class
    );
}
```

```
    /**
     * Adds playlist to user favourites.
     *
     * @param playlistId playlist ID.
     * @return <i>true</i> if was successful.
     */
    public DeezerRequest<Boolean> addPlaylistToFavourites(long playlistId) {
        Map<String, String> params = accessTokenParam();
        params.put("playlist_id", String.valueOf(playlistId));
        return new DeezerPostRequest<>(
                property("user.playlists", "me"),
                params,
                Boolean.class
        );
    }

    /**
     * Adds podcast to user favourites.
     *
     * @param podcastId podcast ID.
     * @return <i>true</i> if was successful.
     */
    public DeezerRequest<Boolean> addPodcastToFavourites(int podcastId) {
        Map<String, String> params = accessTokenParam();
        params.put("podcast_id", String.valueOf(podcastId));
        return new DeezerPostRequest<>(
                property("user.podcasts", "me"),
                params,
                Boolean.class
        );
    }

    /**
     * Adds radio to user favourites.
     *
     * @param radioId radio ID.
     * @return <i>true</i> if was successful.
     */
    public DeezerRequest<Boolean> addRadioToFavourites(int radioId) {
        Map<String, String> params = accessTokenParam();
        params.put("radio_id", String.valueOf(radioId));
        return new DeezerPostRequest<>(
                property("user.radios", "me"),
                params,
                Boolean.class
        );
    }
```

```
    /**
     * Adds tracks to user favourites.
     *
     * @param trackIds tracks IDs.
     * @return <i>true</i> if was successful.
     */
    public DeezerRequest<Boolean> addTracksToFavourites(Integer... trackIds) {
        return addTracksToFavourites(Arrays.asList(trackIds));
    }

    /**
     * Adds tracks to user favourites.
     *
     * @param trackIds tracks IDs.
     * @return <i>true</i> if was successful.
     */
    public DeezerRequest<Boolean> addTracksToFavourites(List<Integer> trackIds) { //
TODO: 21.11.2021 check if it's working
        Map<String, String> params = accessTokenParam();
        params.put("songs", String.valueOf(listConverter.covert(trackIds)));
        return new DeezerPostRequest<>(
                property("user.tracks", "me"),
                params,
                Boolean.class
        );
    }

    /**
     * Removes an album from the user's library.
     *
     * @param albumId album ID.
     * @return <i>true</i> if was successful.
     */
    public DeezerRequest<Boolean> removeAlbum(int albumId) {
        Map<String, String> params = accessTokenParam();
        params.put("album_id", String.valueOf(albumId));
        return new DeezerDeleteRequest<>(
                property("user.albums", "me"),
                params,
                Boolean.class
        );
    }

    /**
     * Removes an artist from the user's favorites.
     *
     * @param artistId artist ID.
     * @return <i>true</i> if was successful.
```

```
     */
    public DeezerRequest<Boolean> removeArtist(int artistId) {
        Map<String, String> params = accessTokenParam();
        params.put("artist_id", String.valueOf(artistId));
        return new DeezerDeleteRequest<>(
                property("user.artists", "me"),
                params,
                Boolean.class
        );
    }

    /**
     * Unfollows user.
     *
     * @param followingId following ID.
     * @return <i>true</i> if was successful.
     */
    public DeezerRequest<Boolean> unfollow(long followingId) {
        Map<String, String> params = accessTokenParam();
        params.put("user_id", String.valueOf(followingId));
        return new DeezerDeleteRequest<>(
                property("user.followings", "me"),
                params,
                Boolean.class
        );
    }

    /**
     * Removes a playlist from the user's favorites.
     *
     * @param playlistId playlist ID.
     * @return <i>true</i> if was successful.
     */
    public DeezerRequest<Boolean> removePlaylist(long playlistId) {
        Map<String, String> params = accessTokenParam();
        params.put("playlist_id", String.valueOf(playlistId));
        return new DeezerDeleteRequest<>(
                property("user.playlists", "me"),
                params,
                Boolean.class
        );
    }

    /**
     * Removes a podcast from the user's favorites.
     *
     * @param podcastId podcast ID.
     * @return <i>true</i> if was successful.
```

```
     */
    public DeezerRequest<Boolean> removePodcast(int podcastId) {
        Map<String, String> params = accessTokenParam();
        params.put("podcast_id", String.valueOf(podcastId));
        return new DeezerDeleteRequest<>(
                property("user.podcasts", "me"),
                params,
                Boolean.class
        );
    }

    /**
     * Removes a radio from the user's favorites.
     *
     * @param radio radio ID.
     * @return <i>true</i> if was successful.
     */
    public DeezerRequest<Boolean> removeRadio(int radio) {
        Map<String, String> params = accessTokenParam();
        params.put("radio_id", String.valueOf(radio));
        return new DeezerDeleteRequest<>(
                property("user.radios", "me"),
                params,
                Boolean.class
        );
    }

    /**
     * Removes a track from the user's favorites.
     *
     * @param trackId track ID.
     * @return <i>true</i> if was successful.
     */
    public DeezerRequest<Boolean> removeTrack(int trackId) {
        Map<String, String> params = accessTokenParam();
        params.put("track_id", String.valueOf(trackId));
        return new DeezerDeleteRequest<>(
                property("user.tracks", "me"),
                params,
                Boolean.class
        );
    }

    /**
     * Creates <i>access_token</i> param.
     *
     * @return <i>access_token</i> param.
     */
```

```
    private Map<String, String> accessTokenParam() {
        return params(entry("access_token", getAccessToken()));
    }
}
```

AdvancedSearchRequest.java

```
package api.deezer.http.impl;

/**
 * Executes Deezer API advanced search request.
 *
 * @param <Response> response POJO type.
 */
public class AdvancedSearchRequest<Response> extends SearchRequest<Response> {
    public AdvancedSearchRequest(String url, Class<Response> responseClass) {
        super(url, responseClass);
    }

    @Override
    public AdvancedSearchRequest<Response> limit(int limit) {
        return (AdvancedSearchRequest<Response>) super.limit(limit);
    }

    @Override
    public AdvancedSearchRequest<Response> offset(int offset) {
        return (AdvancedSearchRequest<Response>) super.offset(offset);
    }

    /**
     * Adds <b>artist</b> parameter.
     *
     * @param artist artist name.
     * @return current instance.
     */
    public AdvancedSearchRequest<Response> artist(String artist) {
        return mergeQ("artist", str(artist));
    }

    /**
     * Adds <b>album</b> parameter.
     *
     * @param album album name.
     * @return current instance.
     */
    public AdvancedSearchRequest<Response> album(String album) {
        return mergeQ("album", str(album));
    }
```

```
/**
 * Adds <b>track</b> parameter.
 *
 * @param track track name.
 * @return current instance.
 */
public AdvancedSearchRequest<Response> track(String track) {
    return mergeQ("track", str(track));
}

/**
 * Adds <b>label</b> parameter.
 *
 * @param label label.
 * @return current instance.
 */
public AdvancedSearchRequest<Response> label(String label) {
    return mergeQ("label", str(label));
}

/**
 * Adds <b>dur_min</b> parameter.
 *
 * @param durMin minimum duration.
 * @return current instance.
 */
public AdvancedSearchRequest<Response> durMin(int durMin) {
    return mergeQ("dur_min", String.valueOf(durMin));
}

/**
 * Adds <b>dur_max</b> parameter.
 *
 * @param durMax maximum duration.
 * @return current instance.
 */
public AdvancedSearchRequest<Response> durMax(int durMax) {
    return mergeQ("dur_max", String.valueOf(durMax));
}

/**
 * Adds <b>bpm_min</b> parameter.
 *
 * @param bpmMin minimum BPM.
 * @return current instance.
 */
public AdvancedSearchRequest<Response> bpmMin(int bpmMin) {
```

```
        return mergeQ("bpm_min", String.valueOf(bpmMin));
    }

    /**
     * Adds <b>bpm_max</b> parameter.
     *
     * @param bpmMax maximum BPM.
     * @return current instance.
     */
    public AdvancedSearchRequest<Response> bpmMax(int bpmMax) {
        return mergeQ("bpm_max", String.valueOf(bpmMax));
    }

    /**
     * Adds double quotes to string.
     *
     * @param string original string.
     * @return double-quoted string.
     */
    private String str(String string) {
        return "\"" + string + "\"";
    }

    /**
     * Merges Q parameter.
     *
     * @param key q key.
     * @param val q value.
     * @return current instance.
     */
    private AdvancedSearchRequest<Response> mergeQ(String key, String val) {
        this.getParams().merge("q", key + ":" + val, (existingQ, newQ) -> existingQ +
" " + newQ);
        return this;
    }
}
```

DeezerDeleteRequest.java

```
package api.deezer.http.impl;

import java.util.HashMap;
import java.util.Map;

/**
 * Executes Deezer API DELETE request.
 *
 * @param <Response> response POJO type.
```

```
 */
public class DeezerDeleteRequest<Response> extends DeezerRequest<Response> {
    public DeezerDeleteRequest(String url, Class<Response> responseClass) {
        this(url, new HashMap<>(), responseClass);
    }

    public DeezerDeleteRequest(String url, Map<String, String> params,
Class<Response> responseClass) {
        super(url, params, responseClass);
        params.put("request_method", "delete");
    }
}
```

DeezerGetRequest.java

```
package api.deezer.http.impl;

import java.util.HashMap;
import java.util.Map;

/**
 * Executes Deezer API GET request.
 *
 * @param <Response> response POJO type.
 */
public class DeezerGetRequest<Response> extends DeezerRequest<Response> {
    public DeezerGetRequest(String url, Class<Response> responseClass) {
        this(url, new HashMap<>(), responseClass);
    }

    public DeezerGetRequest(String url, Map<String, String> params, Class<Response>
responseClass) {
        super(url, params, responseClass);
        params.put("request_method", "get");
    }
}
```

DeezerPostRequest.java

```
package api.deezer.http.impl;

import java.util.HashMap;
import java.util.Map;

/**
 * Executes Deezer API POST request.
 *
 * @param <Response> response POJO type.
 */
```

```
public class DeezerPostRequest<Response> extends DeezerRequest<Response> {
    public DeezerPostRequest(String url, Class<Response> responseClass) {
        this(url, new HashMap<>(), responseClass);
    }

    public DeezerPostRequest(String url, Map<String, String> params, Class<Response>
responseClass) {
        super(url, params, responseClass);
        params.put("request_method", "post");
    }
}
```

DefaultHttpClient.java

```
package api.deezer.http.impl;

import api.deezer.http.HttpClient;
import api.deezer.http.HttpRequest;
import api.deezer.http.utils.URLParamsEncoder;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Map;
import java.util.stream.Collectors;

/**
 * Default implementation of {@link HttpClient}. Based on {@link HttpURLConnection}.
 */
public class DefaultHttpClient implements HttpClient {
    /**
     * {@link Logger} object.
     */
    private static final Logger LOGGER =
LoggerFactory.getLogger(DefaultHttpClient.class);

    @Override
    public String execute(HttpRequest httpRequest) throws IOException {
        LOGGER.debug(
                "Executing request: request_method={}; url={}; params={}",
                httpRequest.getRequestMethod(),
                httpRequest.getUrl(),
                httpRequest.getParams()
        );
```

```
        HttpURLConnection connection = null;
        try {
            connection = (HttpURLConnection) new
URL(httpRequest.getUrl()).openConnection();
            connection.setRequestMethod(httpRequest.getRequestMethod());
            connection.setDoOutput(true);

            Map<String, String> params = httpRequest.getParams();
            if (params != null && !params.isEmpty()) {
                try (DataOutputStream output = new
DataOutputStream(connection.getOutputStream())) {
                    output.writeBytes(URLParamsEncoder.encode(params));
                    output.flush();
                }
            }

            try (BufferedReader reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()))) {
                String response = reader.lines().collect(Collectors.joining());
                LOGGER.debug("Response: {}", response);
                return response;
            }
        } finally {
            if (connection != null) {
                connection.disconnect();
            }
        }
    }
}
```

PaginationRequest.java

```
package api.deezer.http.impl;

import java.util.Map;

/**
 * Executes Deezer API request with pagination.
 *
 * @param <Response> response POJO type.
 */
public class PaginationRequest<Response> extends DeezerGetRequest<Response> {
    public PaginationRequest(String url, Class<Response> responseClass) {
        super(url, responseClass);
    }

    public PaginationRequest(String url, Map<String, String> params, Class<Response>
responseClass) {
        super(url, params, responseClass);
```

```
    }

    /**
     * Adds <b>limit</b> parameter to the request.
     *
     * @param limit limit value.
     * @return current instance.
     */
    public PaginationRequest<Response> limit(int limit) {
        getParams().put("limit", String.valueOf(limit));
        return this;
    }

    /**
     * Adds <b>offset</b> parameter to the request.
     *
     * @param offset offset value.
     * @return current instance.
     */
    public PaginationRequest<Response> offset(int offset) {
        getParams().put("offset", String.valueOf(offset));
        return this;
    }
}
```

SearchRequest.java

```
package api.deezer.http.impl;

import api.deezer.objects.SearchOrder;

/**
 * Executes Deezer API search request.
 *
 * @param <Response> response POJO type.
 */
public class SearchRequest<Response> extends PaginationRequest<Response> {
    public SearchRequest(String url, String q, Class<Response> responseClass) {
        super(url, responseClass);
        getParams().put("q", q);
    }

    public SearchRequest(String url, Class<Response> responseClass) {
        super(url, responseClass);
    }

    @Override
    public SearchRequest<Response> limit(int limit) {
```

```
        return (SearchRequest<Response>) super.limit(limit);
    }

    @Override
    public SearchRequest<Response> offset(int offset) {
        return (SearchRequest<Response>) super.offset(offset);
    }

    /**
     * Adds <b>strict</b> parameter.
     *
     * @return current instance.
     */
    public SearchRequest<Response> strict() {
        getParams().put("strict", "on");
        return this;
    }

    /**
     * Adds <b>order</b> parameter.
     *
     * @param searchOrder search order.
     * @return current instance.
     */
    public SearchRequest<Response> order(SearchOrder searchOrder) {
        getParams().put("order", searchOrder.name());
        return this;
    }
}
```