2.3. Оцінка складності робіт методом UCP

1. Першим етапом є визначення типів акторів. Акторам Неавторизований користувач і Авторизований користувач призначається тип "Складний", так як вони є кінцевими користувачами, з якими буде відбуватися взаємодія через інтерфейс користувача. Акторам YouTube і SoundCloud призначається тип "Середній", так як вони окремо являють собою складний і гнучкий API.

Таблиця 2.3.1 – Типи акторів

Назва	Тип	Вага
Неавторизований користувач	Складний	3
Авторизований користувач	Складний	3
YouTube	Середній	2
SoundCloud	Середній	2

UAW = 2 * 3 + 2 * 2 = 6 + 4 = 10

2. Другим кроком є визначення типів варіантів використання за кількістю виконуваних транзакцій.

Таблиця 2.3.2 – Типи варіантів використання

Назва	Тип	Вага	Кількість транзакцій
Реєстрація	простий	5	2
Авторизація	простий	5	2
Перегляд списку музичних композицій	простий	5	2
Прослуховування музичної композиції	середній	10	4
Зміна стану музичного програвача	простий	5	1
Фільтрування списку музичних композицій	простий	5	2
Пошук музичних композицій	простий	5	2
Перегляд рекомендацій до музичної композиції	простий	5	2

Продовження таблиці 2.3.2 – Типи варіантів використання

Назва	Тип	Вага	Кількість транзакцій
Перегляд історії змін музичної композиції	простий	5	2
Створення списку музичних композицій	простий	5	2
Редагування списку музичних композицій	простий	5	2
Видалення списку музичних композицій	простий	5	2
Перегляд переліку списків музичних композицій	простий	5	1
Створення елемента списку музичних композицій	середній	10	4
Редагування елемента списку музичних композицій	середній	10	6
Видалення елемента списку музичних композицій	простий	5	1
Створення рекомендації до музичної композиції	простий	5	2
Оцінка рекомендації до музичної композиції	простий	5	2
Поскаржитися на рекомендацію до музичної композиції	простий	5	2

UUCW = 16 * 5 + 3 * 10 = 110

UCP = UAW + UUCW = 10 + 110 = 120

3. Третім етапом є визначення впливу технічних факторів на терміни розробки програмного продукту.

Таблиця 2.3.3 – Вплив технічних факторів

Фактор	Опис	Вага	Оцінка
T1	Розподіленість системи	2	1
T2	Час відгуку	1	1
T3	Ефективність кінцевого користувача	1	1
T4	Складність обробки	1	1

Продовження таблиці 2.3.3 – Вплив технічних факторів

Фактор	Опис	Вага	Оцінка
T5	Фокус на повторному використанні коду	1	1
T6	Простота інсталяції	0,5	1
T7	Простота використання	0,5	3
T8	Портативність	2	1
T9	Простота зміни	1	2
T10	Паралельні обчислення	1	1
T11	Засоби захисту	1	2
T12	Доступ до третьої сторони	1	5
T13	Потреба в спеціальному навчанні	1	1

TFactor = 2 + 1 + 1 + 1 + 1 + 0.5 + 1.5 + 2 + 2 + 1 + 2 + 5 + 1 = 21

TCF = 0.6 + (0.01 * TFactor) = 0.6 * 0.21 = 0.126

3. Четвертим етапом є визначення впливу зовнішних факторів на проект.

Таблиця 2.3.4 – Вплив зовнішних факторів

Фактор	Опис	Вага	Оцінка
F1	Знайомство з процесом розробки	1,5	2
F2	Досвід подібних проектів	0,5	2
F3	Досвід об'єктно-орієнтованої розробки	1	2
F4	Досвідченість аналітика	0,5	3
F5	Мотивація	1	4
F6	Стабільність вимог	2	3
F7	Часткова зайнятість працівників	-1	2
F8	Складність мови програмування	-1	2

EFactor = 3 + 1 + 2 + 1.5 + 4 + 6 - 2 - 2 = 13.5

EF = 1.4 + (-0.03 * EFactor) = 1.4 - 0.405 = 0.995

5. Останнім етапом отримання UCP оцінки є підрахунок підсумовуючої оцінки. На один UCP доводиться 28 годин, тому що кількість показників F1-F6 нижче 3 і показників F7-F8 вище 3 дорівнює 3.

AUCP = UCP * TCF * EF = 120 * 28 * 0.126 * 0.995 = 422 (чоловіко-годин).

Кількість учасників команди дорівнює 2, тому очікувана тривалість виконання проекту дорівнює 422 / 2 = 211 годинам.

3. Проектування програмного продукту

3.1. Концептуальне проектування програмного продукту

Для проектованої програмної системи була розроблена діаграма концептуальних класів, що зображає види об'єктів і суб'єктів веб-застосування. Діаграму можна спостерігати на рисунку 3.1.1.

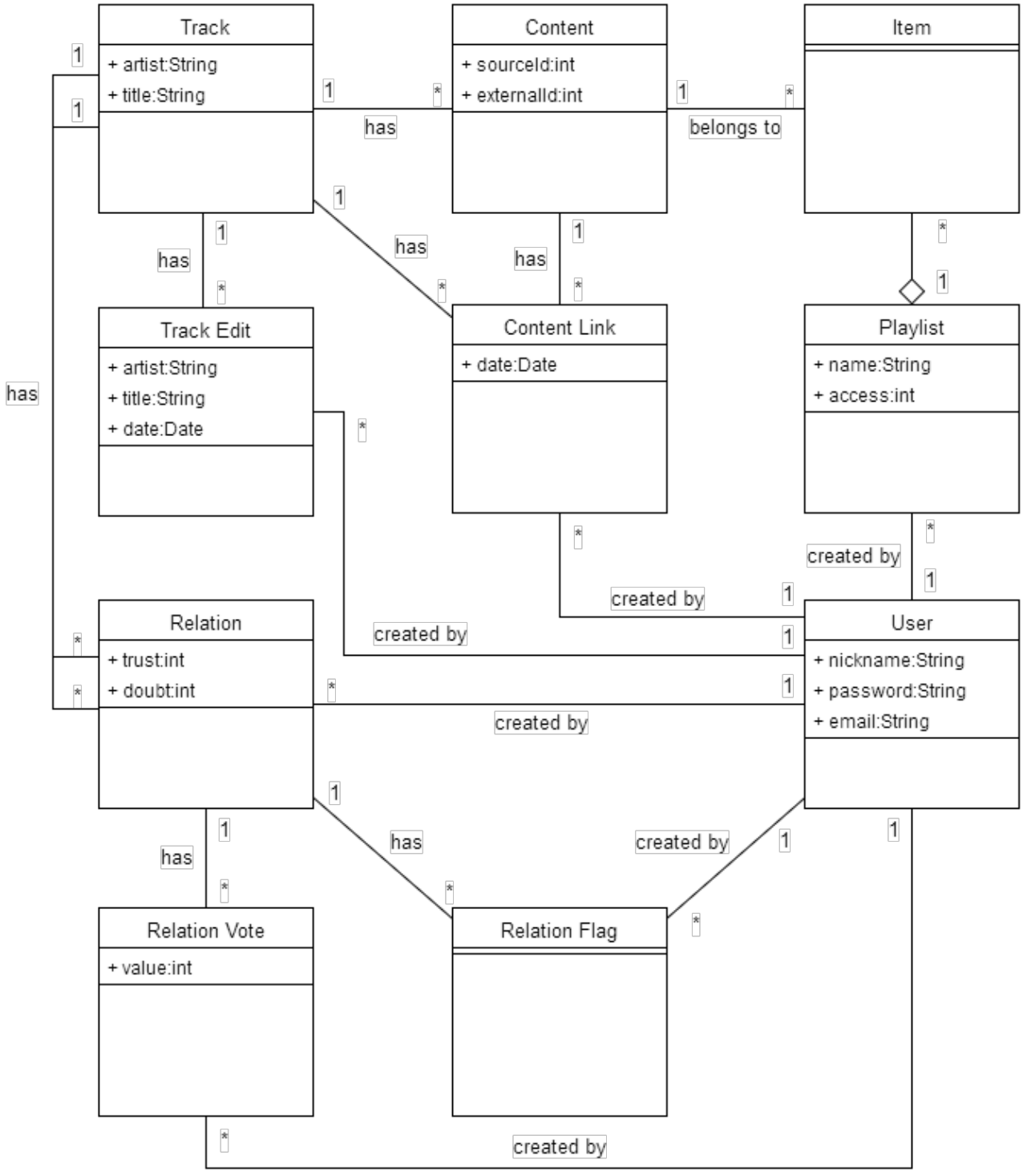

Рисунок 3.1.1 – Діаграма концептуальних класів

3.2. Розробка діаграми програмних класів

Для серверної сторони веб-застосування була розроблена діаграма програмних класів, що зображає компоненти системи та їх функціонал. Результат можна спостерігати на рисунку 3.2.1.

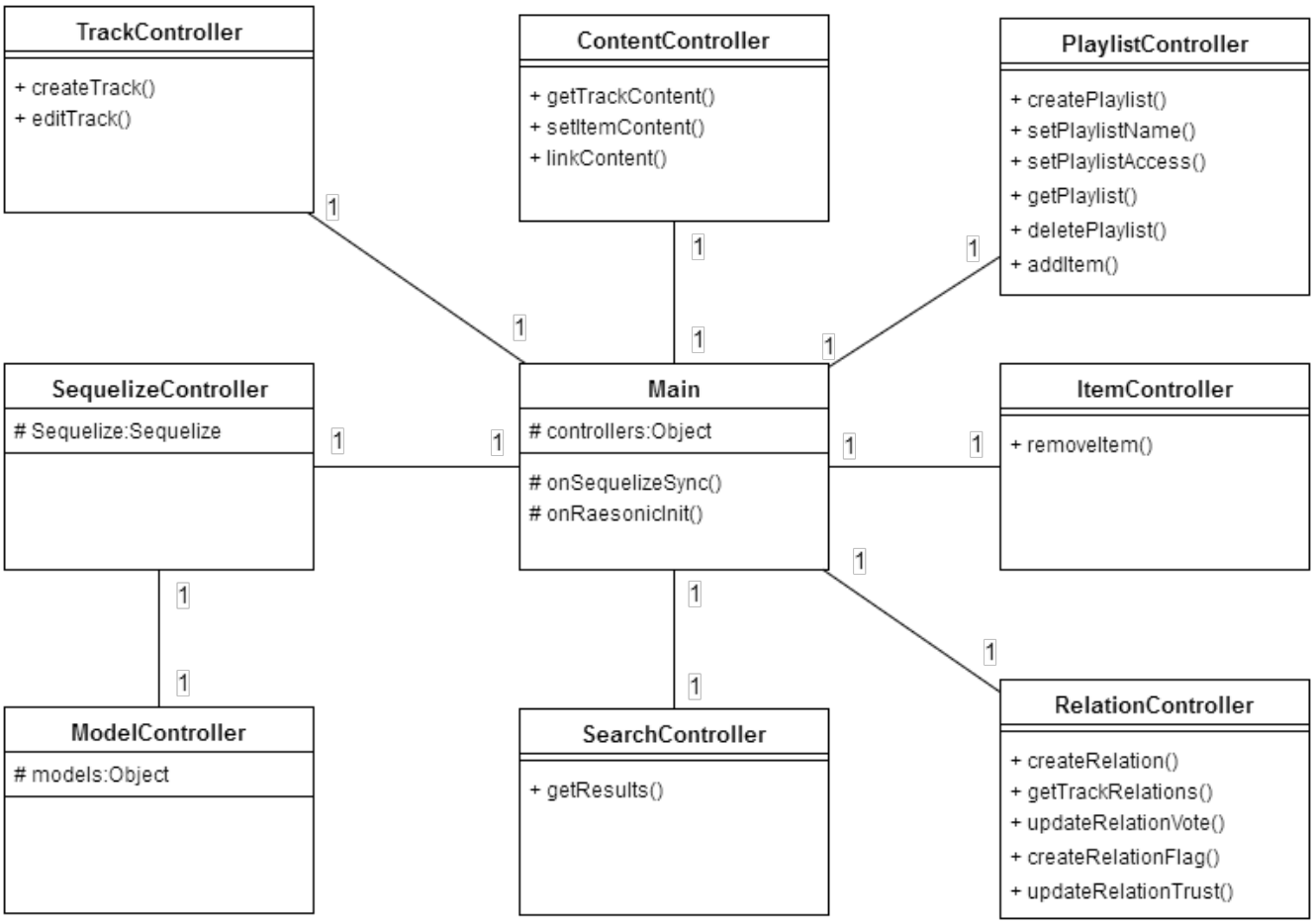

Рисунок 3.2.1 – Діаграма програмних класів

Основним серверним класом є Main, що знаходиться в файлі raesonic.js. SequelizeController відповідає за роботу з middleware, що взаємодіє з базами даних. ModelController містить всі моделі бази даних, тобто класи для всіх таблиць бази даних та інформацію про зв'язки між ними. Решта контролерів відповідають за роботу над даними конкретної області. Структури даних, які дозволяють ознайомиться з призначеннями кожного з них будуть розглянуті далі.

3.3. Опис структур даних

Дані про рекомендації, списки музичних композицій, аудіо/відео контент, історію змін музичних композицій, користувачів веб-застосування, та базова інформація про музичні композиції зберігаються в Базі Даних виду MySQL або SQLite (задається у файлі конфігурації, первісно SQLite).

Конфігураційний файл має найменування config.js, та знаходиться в корені папки веб-застосування, що недоступна з мережі Інтернет. Він має такий вигляд:

```
module.exports =
{
	server:
	{
		port: 3000
	},
	database:
	{
		dialect: "sqlite",
		host: "localhost",
		name: "raesonic",
		user: "user",
		password: "password"
	}
}
```

Таким чином, після підключення даного файлу в змінну config, можна отримати необхідний порт, на якому має працювати веб-застосування, використавши ідентифікатор config.server.port, а діалект бази даних, використавши ідентифікатор config.database.dialect.

Розглянемо структури даних таблиць, розташованих в Базі Даних. Моделі класів знаходяться в директорії /server/models/ та мають найменування файлів виду <Назва класу>Model.js.

Музична композиція (Track) зберігає найменування виконавця та самої композиції (до 50 символів).

```
var Track = sequelize.define ("Track",
{
	trackId: {type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true},
	artist: Sequelize.STRING(50),
	title: Sequelize.STRING(50)
});
```

Контент (Content) зберігає інформацію про вид джерела (sourceId) та зовнішний ідентифікатор контенту (до 20 символів).

```
var Content = sequelize.define ("Content",
{
	contentId: {type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true},
	sourceId: Sequelize.INTEGER (1),
	externalId: Sequelize.STRING (20)
});
```

Контент прив'язаний до однієї музичної композиції.

```
Track.hasMany (Content, {foreignKey: "trackId"});
Content.belongsTo (Track, {foreignKey: "trackId"});
```

Список композицій (Playlist) зберігає найменування списку та тип доступу.

```
var Playlist = sequelize.define ("Playlist",
{
	playlistId: {type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true},
	name: Sequelize.STRING (50),
	access: {type: Sequelize.INTEGER (1), defaultValue: 1}
});
```

Список композиції має одного власника.

```
User.hasMany (Playlist, {foreignKey: "userId"});
Playlist.belongsTo (User, {foreignKey: "userId"});
```

Елемент списку композицій (Item) є зв'язком між контентом та списком композицій.

```
var Item = sequelize.define ("Item",
{
	itemId: {type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true}
});
```

До одного та того ж контенту може бути прив'язане кілька елементів списку композицій, а список композицій може складатися з декількох елементів.

```
Content.hasMany (Item, {foreignKey: "contentId"});
Item.belongsTo (Content, {foreignKey: "contentId"});
Playlist.hasMany (Item, {foreignKey: "playlistId"});
Item.belongsTo (Playlist, {foreignKey: "playlistId"});
```

Зв'язок між композиціями (Relation) є користувальницької рекомендацією, точність якої може відрізнятися, а оцінка ґрунтується на результатах голосування користувачами.

```
var Relation = sequelize.define ("Relation",
{
	relationId: {type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true},
	trust: {type: Sequelize.INTEGER, defaultValue: 1},
	doubt: {type: Sequelize.INTEGER, defaultValue: 0}
});
```

Зв'язок проходить через дві композиції, track та linked.

```
Track.hasMany (Relation, {foreignKey: "trackId"});
Relation.belongsTo (Track, {foreignKey: "trackId", as: "Track"});
Track.hasMany (Relation, {foreignKey: "linkedId"});
Relation.belongsTo (Track, {foreignKey: "linkedId", as: "Linked"});
```

Зміна інформації про композицію (TrackEdit) дозволяє отримати інформацію про те, який користувач змінив автора/найменування композиції, та в який час були проведені зміни.

```
var TrackEdit = sequelize.define ("TrackEdit",
{
	editId: {type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true},
	artist: Sequelize.STRING (50),
	title: Sequelize.STRING (50),
	userId: Sequelize.INTEGER,
	date: {type: Sequelize.DATE, defaultValue: Sequelize.NOW}
});
```

Інформація про з'єднання контенту з композицією (ContentLink) також дозволяє відстежити зміни, але вже пов'язані з прив'язкою контенту до певної композиції.

```
var ContentLink = sequelize.define ("ContentLink",
{
	linkId: {type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true},
	userId: Sequelize.INTEGER,
	date: {type: Sequelize.DATE, defaultValue: Sequelize.NOW}
});
```

Оцінки рекомендацій до музичних композицій авторизованими користувачами зберігаються в таблиці голосів (RelationVote).

```
var RelationVote = sequelize.define ("RelationVote",
{
	voteId: {type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true},
	value: Sequelize.INTEGER (1),
	date: {type: Sequelize.DATE, defaultValue: Sequelize.NOW}
});
```

Скарги авторизованих користувачів на невідповідність рекомендацій до музичних композицій зберігаються в таблиці скарг (RelationFlag).

```
var RelationFlag = sequelize.define ("RelationFlag",
{
    flagId: {type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true},
    date: {type: Sequelize.DATE, defaultValue: Sequelize.NOW}
});
```

Діаграму класів для моделей Бази Даних можна спостерігати на рисунку 3.3.1.

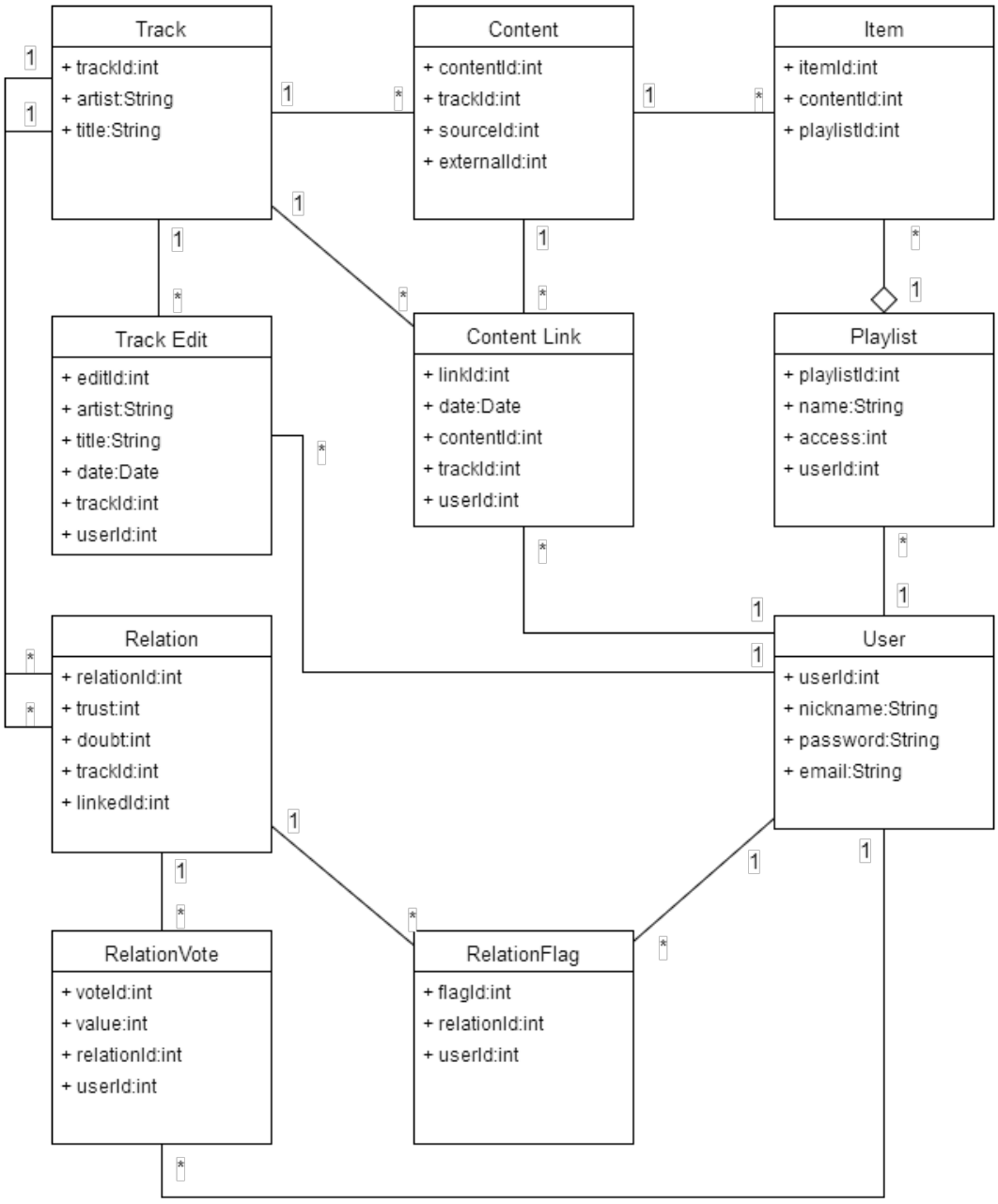

Рисунок 3.3.1 – Діаграма класів для таблиць БД