Desarrollo web entorno servidor

Documentación de la aplicación

Spring Boot ‘Digital Game Store’

Web Services

Vincenzo Vitulli, Lincoln Guaman, Bruno Machaca

DAW II

Tabla de contenidos

Temática de la aplicación

El proyecto desarrollado consiste en una plataforma en línea, llamada 'Digital Game Store' diseñada para la compra y alquiler de videojuegos digitales.

Los usuarios pueden explorar un catálogo de juegos, realizar transacciones de compra o alquiler, y gestionar su cuenta y saldo.

Todo se hace gestionando la base de datos llamada DatabaseGameStore.sql.

Esta aplicación ha sido desarrollada mediante Spring Boot (Spring Framework) y su funcionalidad está hecha enteramente para recibir mediante URL peticiones HTTP de tipo GET, utilizando la anotación @GetMapping de JPA.

Estructura de la Base de Datos DigitalGameStore.sql:

Tabla user

- **Descripción:** Almacena información de los usuarios registrados.
- **Campos:**
    - user_id: Identificador único del usuario (PK).
    - username: Nombre de usuario.
    - email: Correo electrónico del usuario.
    - password: Contraseña de la cuenta.
    - account_balance: Balance disponible en la cuenta del usuario.

Tabla game

- **Descripción:** Almacena información de los juegos disponibles.
- **Campos:**
    - game_id: Identificador único del juego (PK).
    - title: Título del juego.
    - genre: Géneros del juego (SET con 64 valores posibles, como Action, Adventure, etc.).
    - developer: Desarrollador del juego.
    - release_date: Fecha de lanzamiento del juego.
    - price: Precio de compra.
    - lease_price: Precio de alquiler.
    - description: Breve descripción del juego.

Tabla transaction

- **Descripción:** Almacena información sobre compras o alquileres realizados por los usuarios.
- **Campos:**
    - transaction_id: Identificador único de la transacción (PK).
    - user_id: Relación con la tabla user (FK).
    - game_id: Relación con la tabla game (FK).
    - transaction_type: Tipo de transacción enumerado (Purchase o Lease).
    - transaction_date: Fecha de la transacción.
    - expiry_date: Fecha de expiración (solo para alquileres).
    - amount: Monto de la transacción.

Funcionalidades desarrolladas

Gestión de Usuarios:

1. **Obtención de Información de Usuarios:**
    - Los usuarios pueden obtener la información de un usuario específico utilizando su ID.
    - URL Ejemplo: localhost:8080/users/1
2. **Obtención de Todos los Usuarios:**
    - Los usuarios pueden obtener una lista de todos los usuarios registrados en la plataforma.
    - URL Ejemplo: localhost:8080/users/all
3. **Registro de Nuevos Usuarios:**
    - Los usuarios pueden registrarse proporcionando su nombre de usuario, correo electrónico y contraseña.
    - URL Ejemplo: localhost:8080/users/add?username=john&email=john@example.com&password=1234
4. **Eliminación de Usuarios:**
    - Los administradores pueden eliminar cuentas de usuario utilizando su ID.
    - URL Ejemplo: localhost:8080/users/delete/1
5. **Actualización de Saldo de Cuenta:**
    - Los usuarios pueden actualizar su saldo de cuenta, lo cual es esencial para realizar compras y alquileres en la tienda.
    - URL Ejemplo: localhost:8080/users/1/balance?newBalance=500.0

Gestión de Juegos:

1. **Obtención de Información de Juegos:**
    - Los usuarios pueden obtener la información de un juego específico utilizando su ID.
    - URL Ejemplo: localhost:8080/games/1
2. **Obtención de Todos los Juegos:**
    - Los usuarios pueden obtener una lista de todos los juegos disponibles en la tienda.
    - URL Ejemplo: localhost:8080/games/all
3. **Registro de Nuevos Juegos:**
    - Los administradores pueden agregar nuevos títulos a la tienda, especificando detalles como el título, género, desarrollador, fecha de lanzamiento, precio de compra y precio de alquiler.
    - URL Ejemplo: localhost:8080/games/add?title=gameTitle&genre=Action&developer=developerName&releaseDate=2023-11-22&price=69.99&leasePrice=19.99&description=GameDescription
4. **Eliminación de Juegos:**
    - Los administradores pueden eliminar juegos del catálogo utilizando su ID.
    - URL Ejemplo: localhost:8080/games/delete/1
5. **Actualización de Precio de Juegos:**
    - Los administradores pueden actualizar el precio de un juego específico.
    - URL Ejemplo: localhost:8080/games/1/price?newPrice=49.99

Gestión de Transacciones:

1. **Obtención de Información de Transacciones:**
    - Los usuarios pueden obtener la información de una transacción específica utilizando su ID.
    - URL Ejemplo: localhost:8080/transactions/1
2. **Obtención de Todas las Transacciones:**
    - Los usuarios pueden obtener una lista de todas las transacciones registradas en la plataforma.
    - URL Ejemplo: localhost:8080/transactions/all
3. **Registro de Transacciones de Compra y Alquiler:**
    - Los usuarios pueden realizar transacciones para comprar o alquilar juegos. Cada transacción se registra en la base de datos, incluyendo detalles como el usuario, el juego, el tipo de transacción, la fecha y el monto.
    - URL Ejemplo: localhost:8080/transactions/add?userId=1&gameId=2&transactionType=Purchase
4. **Eliminación de Transacciones:**
    - Los administradores pueden eliminar transacciones utilizando su ID.
    - URL Ejemplo: localhost:8080/transactions/delete/1
5. **Consulta de Transacciones por Usuario:**
    - Los usuarios pueden revisar su historial de transacciones, permitiéndoles ver todas las compras y alquileres realizados en la plataforma.
    - URL Ejemplo: localhost:8080/transactions/user/1

Descripción de los servicios web para cada entidad

Servicios Web y su efecto sobre la Base de Datos – Tabla 'user'

Entity User

El archivo User.java define la entidad User que representa a los usuarios en la base de datos. A continuación, se muestra el código relevante:

```java
@Entity
@Table(name = "User")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int userId;

    @Column(nullable = false)
    private String username;

    @Column(nullable = false)
    private String email;

    @Column(nullable = false)
    private String password;

    @Column(nullable = false, columnDefinition = "DECIMAL(10,2)")
    private double accountBalance;

    // Constructors
    public User() {
    }

    public User(String username, String email, String password) {

        this.username = username;
        this.email = email;
        this.password = password;
        this.accountBalance = 0.00;
    }

    // Getters and Setters
}
```

Repositorio User

El archivo UserRepository.java define el repositorio de datos para la entidad User, que permite realizar operaciones CRUD en la base de datos sin necesidad de escribir SQL manualmente. A continuación, se muestra el código relevante:

```java
@Repository // No need to add this annotation since Spring Boot automatically
detects repository interfaces if they extend JpaRepository
public interface UserRepository extends JpaRepository<User, Integer> {
}
```

Controlador User

El archivo UserController.java define los servicios web para la entidad User. A continuación, se describe cada servicio y su efecto sobre la base de datos:

GET /users/{userId}

- Descripción: Devuelve la información del usuario especificado.
- Efecto en la Base de Datos: Recupera un registro de la tabla User por su ID.
- Código:

```
@GetMapping("/{userId}")
    public User getUser(@PathVariable("userId") int userId) {
        try {
            Optional<User> optionalUser = userRepository.findById(userId);
            return optionalUser.orElse(new User());
        } catch (Exception e) {
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
                                              "Error retrieving user", e);
        }
    }
```

GET /users/all

- Descripción: Devuelve una lista de todos los usuarios.
- Efecto en la Base de Datos: Recupera todos los registros de la tabla User.
- Código:

```
@GetMapping("/all")
    public List<User> getAllUsers() {
        try {
            return userRepository.findAll();
        } catch (Exception e) {
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
                                              "Error retrieving users", e);
        }
    }
```

GET /users/add

- Descripción: Agrega un nuevo usuario con los parámetros especificados.
- Efecto en la Base de Datos: Inserta un nuevo registro en la tabla User.
- Código:

```
@GetMapping("/add")
    public User addUser(
            @RequestParam("username") String username,
            @RequestParam("email") String email,
            @RequestParam("password") String password) {
        try {
            User user = new User(username, email, password);
            return userRepository.save(user);
        } catch (Exception e) {
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST,
                                              "Invalid user parameters", e);
        }
    }
```

GET /users/delete/{userId}

- Descripción: Elimina el usuario especificado.
- Efecto en la Base de Datos: Elimina un registro de la tabla User por su ID.
- Código:

```
@GetMapping("/delete/{userId}")
    public User deleteUser(@PathVariable("userId") int userId) {
        try {
            Optional<User> optionalUser = userRepository.findById(userId);
            if (optionalUser.isPresent()) {
                User user = optionalUser.get();
                userRepository.delete(user);
                return user;
            } else {
                return new User();
            }
        } catch (Exception e) {
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
                                              "Error deleting user", e);
        }
    }
```

GET /users/{userId}/balance

- Descripción: Actualiza el saldo del usuario especificado.
- Efecto en la Base de Datos: Actualiza el campo accountBalance de un registro en la tabla User.
- Código:

```
@GetMapping("/{userId}/balance")
    public User updateUserBalance(@PathVariable int userId, @RequestParam double
newBalance) {
        try {
            Optional<User> optionalUser = userRepository.findById(userId);
            if (optionalUser.isPresent()) {
                User user = optionalUser.get();
                user.setAccountBalance(newBalance);
                userRepository.save(user);
                return user;
            } else {
                return new User();
            }
        } catch (Exception e) {
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
                                              "Error updating user balance", e);
        }
    }
```

Información para Acceso a los Servicios

- **Base URL**: /users
- **Servicios**:
    - **GET /{userId}**: Obtiene la información de un usuario por su ID.

        Ejemplo: /users/1
    - **GET /all**: Obtiene la lista de todos los usuarios.
        - Ejemplo: /users/all
    - **GET /add**: Agrega un nuevo usuario.
        - Ejemplo: /users/add?username=john&email=john@example.com&password=1234
    - **GET /delete/{userId}**: Elimina un usuario por su ID.
        - Ejemplo: /users/delete/1
    - **GET /{userId}/balance**: Actualiza el saldo de un usuario por su ID.
        - Ejemplo: /users/1/balance?newBalance=500.0

Servicios Web y su efecto sobre la Base de Datos – Tabla ‘game’

Entity Game

El archivo Game.java define la entidad Game que representa a los juegos en la base de datos. A continuación, se muestra el código relevante:

```
@Entity
@Table(name = "Game")
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int gameId;

    @Column(nullable = false)
    private String title;

    @Column(nullable = false, columnDefinition =
"SET('Action','Adventure','Arcade','Battle Royale','Beat ''em
Up','Builder','Card','Casual','City Builder','Comedy','Cyberpunk','Dating
Sim','Detective','Dungeon Crawler','Educational','Endless
Runner','Exploration','Fantasy','Fighting','First-Person Shooter (FPS)','Hack and
Slash','Historical','Horror','Idle','Interactive Story','JRPG','Life
Simulation','Management','Massively Multiplayer Online
(MMO)','Metroidvania','Military','Minigames','Music','Mystery','Narrative','Open
World','Party','Pinball','Platformer','Post-Apocalyptic','Puzzle','Racing','Real-
Time Strategy (RTS)','Retro','Rhythm','Roguelike','Roguelite','Role-Playing Game
(RPG)','Sandbox','Sci-
Fi','Shooter','Social','Sports','Stealth','Strategy','Survival','Survival
Horror','Tactical','Third-Person Shooter','Tower Defense','Trading Card','Turn-
Based Strategy (TBS)','Tycoon','Visual Novel')")
    private String genre;

    @Column(nullable = false)
    private String developer;

    @Column(nullable = false, columnDefinition = "DATE")
    private String releaseDate;

    @Column(nullable = false, columnDefinition = "DECIMAL(10,2)")
    private double price;

    @Column(nullable = false, columnDefinition = "DECIMAL(10,2)")
    private double leasePrice;
```

```java
    @Column(nullable = false, columnDefinition = "TEXT")
    private String description;

    // Constructors
    public Game() {
    }

    public Game(String title, String genre, String developer, String releaseDate,
                double price, double leasePrice, String description) {

        this.title = title;
        this.genre = genre;
        this.developer = developer;
        this.releaseDate = releaseDate;
        this.price = price;
        this.leasePrice = leasePrice;
        this.description = description;
    }

    // Getters and Setters
}
```

Repositorio Game

El archivo GameRepository.java define el repositorio de datos para la entidad Game, que permite realizar operaciones CRUD en la base de datos sin necesidad de escribir SQL manualmente. A continuación, se muestra el código relevante:

```java
@Repository // No need to add this annotation since Spring Boot automatically
detects repository interfaces if they extend JpaRepository
public interface GameRepository extends JpaRepository<Game, Integer> {
}
```

Controlador Game

El archivo GameController.java define los servicios web para la entidad Game. A continuación, se describe cada servicio y su efecto sobre la base de datos:

GET /games/{gameId}

- Descripción: Devuelve la información del juego especificado.
- Efecto en la Base de Datos: Recupera un registro de la tabla Game por su ID.
- Código:

```
@GetMapping("/{gameId}")
    public Game getGame(@PathVariable("gameId") int gameId) {
        try {
            Optional<Game> optionalGame = gameRepository.findById(gameId);
            return optionalGame.orElse(new Game());
        } catch (Exception e) {
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
                                              "Error retrieving game", e);
        }
    }
```

GET /games/all

- Descripción: Devuelve una lista de todos los juegos.
- Efecto en la Base de Datos: Recupera todos los registros de la tabla Game.
- Código:

```
@GetMapping("/all")
    public List<Game> getAllGames() {
        try {
            return gameRepository.findAll();
        } catch (Exception e) {
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
                                              "Error retrieving games", e);
        }
    }
```

GET /games/add

- Descripción: Agrega un nuevo juego con los parámetros especificados.
- Efecto en la Base de Datos: Inserta un nuevo registro en la tabla Game.
- Código:

Asdas

```
@GetMapping("/add")
    public Game addGame(
            @RequestParam("title") String title,
            @RequestParam("genre") String genre,
            @RequestParam("developer") String developer,
            @RequestParam("releaseDate") String releaseDate,
            @RequestParam("price") double price,
            @RequestParam("leasePrice") double leasePrice,
            @RequestParam("description") String description) {
        try {
            Game game = new Game(title, genre, developer, releaseDate, price,
                                 leasePrice, description);
            return gameRepository.save(game);
        } catch (Exception e) {
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST,
                                              "Invalid game parameters", e);
        }
    }
```

GET /games/delete/{gameId}

- Descripción: Elimina el juego especificado.
- Efecto en la Base de Datos: Elimina un registro de la tabla Game por su ID.
- Código:

```
@GetMapping("/delete/{gameId}")
    public Game deleteGame(@PathVariable("gameId") int gameId) {
        try {
            Optional<Game> optionalGame = gameRepository.findById(gameId);
            if (optionalGame.isPresent()) {
                Game game = optionalGame.get();
                gameRepository.delete(game);
                return game;
            } else {
                return new Game();
            }
        } catch (Exception e) {
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
"Error deleting game", e);
        }
    }
```

GET /games/{gameId}/price

- Descripción: Actualiza el precio del juego especificado.
- Efecto en la Base de Datos: Actualiza el campo price de un registro en la tabla Game.
- Código:

```
@GetMapping("/{gameId}/price")
    public Game updateGamePrice(@PathVariable("gameId") int gameId,
@RequestParam("newPrice") double newPrice) {
        try {
            Optional<Game> optionalGame = gameRepository.findById(gameId);
            if (optionalGame.isPresent()) {
                Game game = optionalGame.get();
                game.setPrice(newPrice);
                gameRepository.save(game);
                return game;
            } else {
                return new Game();
            }
        } catch (Exception e) {
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
                                              "Error updating game price", e);
        }
    }
```

Información para Acceso a los Servicios

- **Base URL**: /games
- **Servicios**:
    - **GET /{gameId}**: Obtiene la información de un juego por su ID.
        - Ejemplo: /games/1
    - **GET /all**: Obtiene la lista de todos los juegos.
        - Ejemplo: /games/all
    - **GET /add**: Agrega un nuevo juego.
        - Ejemplo: /games/add?title=gameTitle&genre=Action&developer=developerName&releaseDate=2023-11-22&price=69.99&leasePrice=19.99&description=GameDescription
    - **GET /delete/{gameId}**: Elimina un juego por su ID.
        - Ejemplo: /games/delete/1
    - **GET /{gameId}/price**: Actualiza el precio de un juego por su ID.
        - Ejemplo: /games/1/price?newPrice=49.99

Servicios Web y su efecto sobre la Base de Datos – Tabla ‘transaction

Entity Transaction

El archivo Transaction.java define la entidad Transaction que representa las transacciones en la base de datos. A continuación, se muestra el código relevante:

```java
@Entity
@Table(name = "Transaction")
public class Transaction {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int transactionId;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "game_id", nullable = false)
    private Game game;

    @Column(nullable = false, columnDefinition = "ENUM('Purchase', 'Lease')")
    private String transactionType;

    @Column(nullable = false, columnDefinition = "DATE")
    private String transactionDate;

    @Column(columnDefinition = "DATE")
    private String expiryDate;

    @Column(nullable = false, columnDefinition = "DECIMAL(10,2)")
    private double amount;









    // Constructors
```

```
    public Transaction() {
    }

    public Transaction(User user, Game game, String transactionType,
            String transactionDate, String expiryDate, double amount) {
        this.user = user;
        this.game = game;
        this.transactionType = transactionType;
        this.transactionDate = transactionDate;
        this.expiryDate = expiryDate;
        this.amount = amount;
    }

    // Getters and Setters
}
```

Repositorio Transaction

El archivo TransactionRepository.java define el repositorio de datos para la entidad Transaction, que permite realizar operaciones CRUD en la base de datos sin necesidad de escribir SQL manualmente. A continuación, se muestra el código relevante:

```java
@Repository // No need to add this annotation since Spring Boot automatically
detects repository interfaces if they extend JpaRepository
public interface TransactionRepository
                extends JpaRepository<Transaction, Integer> {
    @Query("SELECT t FROM Transaction t WHERE t.user.userId = :userId")
    List<Transaction> findAllByUserId(@Param("userId") int userId);
}
```

Controlador Transaction

El archivo TransactionController.java define los servicios web para la entidad Transaction. A continuación, se describe cada servicio y su efecto sobre la base de datos:

GET /transaction/{transactionId}

- Descripción: Devuelve la información de la transacción especificada.
- Efecto en la Base de Datos: Recupera un registro de la tabla Transaction por su ID.
- Código:

```java
@GetMapping("/{transactionId}")
    public Transaction getTransaction(@PathVariable("transactionId") int
                                                                transactionId) {
        try {
            Optional<Transaction> optionalTransaction =
            transactionRepository.findById(transactionId);
            return optionalTransaction.orElse(new Transaction());
        } catch (Exception e) {
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
                                            "Error retrieving transaction", e);
        }
    }
```

GET /transactions/all

- Descripción: Devuelve una lista de todas las transacciones.
- Efecto en la Base de Datos: Recupera todos los registros de la tabla Transaction.
- Código:

```
@GetMapping("/all")
    public List<Transaction> getAllTransactions() {
        try {
            return transactionRepository.findAll();
        } catch (Exception e) {
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
                                              "Error retrieving transactions", e);
        }
    }
```

GET /transactions/add

- Descripción: Agrega una nueva transacción con los parámetros especificados.
- Efecto en la Base de Datos: Inserta un nuevo registro en la tabla Transaction.
- Código:

```
@GetMapping("/add")
    public Transaction addTransaction(
                    @RequestParam("userId") int userId,
                    @RequestParam("gameId") int gameId,
                    @RequestParam("transactionType") String transactionType) {
        try {
            Optional<User> optionalUser = userRepository.findById(userId);
            Optional<Game> optionalGame = gameRepository.findById(gameId);

            if (optionalUser.isEmpty()) {
                throw new ResponseStatusException(HttpStatus.NOT_FOUND,
                                                  "User not found");
            }

            if (optionalGame.isEmpty()) {
                throw new ResponseStatusException(HttpStatus.NOT_FOUND,
                                                 "Game not found");
            }

            User user = optionalUser.get();
            Game game = optionalGame.get();

            LocalDate currentDate = LocalDate.now();

            String expiryDate = transactionType.equals("Lease") ?
            currentDate.plusDays(30).toString() : null;

            double amount = game.getPrice();

            Transaction transaction = new Transaction(user, game,
                                        transactionType, currentDate.toString(),
                                        expiryDate, amount);
            return transactionRepository.save(transaction);
        } catch (Exception e) {
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST,
                                           "Invalid transaction parameters", e);
        }
```

GET /transactions/delete/{transactionId}

- Descripción: Elimina la transacción especificada.
- Efecto en la Base de Datos: Elimina un registro de la tabla Transaction por su ID.
- Código:

```
@GetMapping("/delete/{transactionId}")
    public Transaction deleteTransaction(@PathVariable("transactionId") int
                                                                    transactionId) {
        try {
            Optional<Transaction> optionalTransaction =
            transactionRepository.findById(transactionId);
            if (optionalTransaction.isPresent()) {
                Transaction transaction = optionalTransaction.get();
                transactionRepository.delete(transaction);
                return transaction;
            } else {
                return new Transaction();
            }
        } catch (Exception e) {
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
                                              "Error deleting transaction", e);
        }
    }
```

GET /transactions/user/{userId}

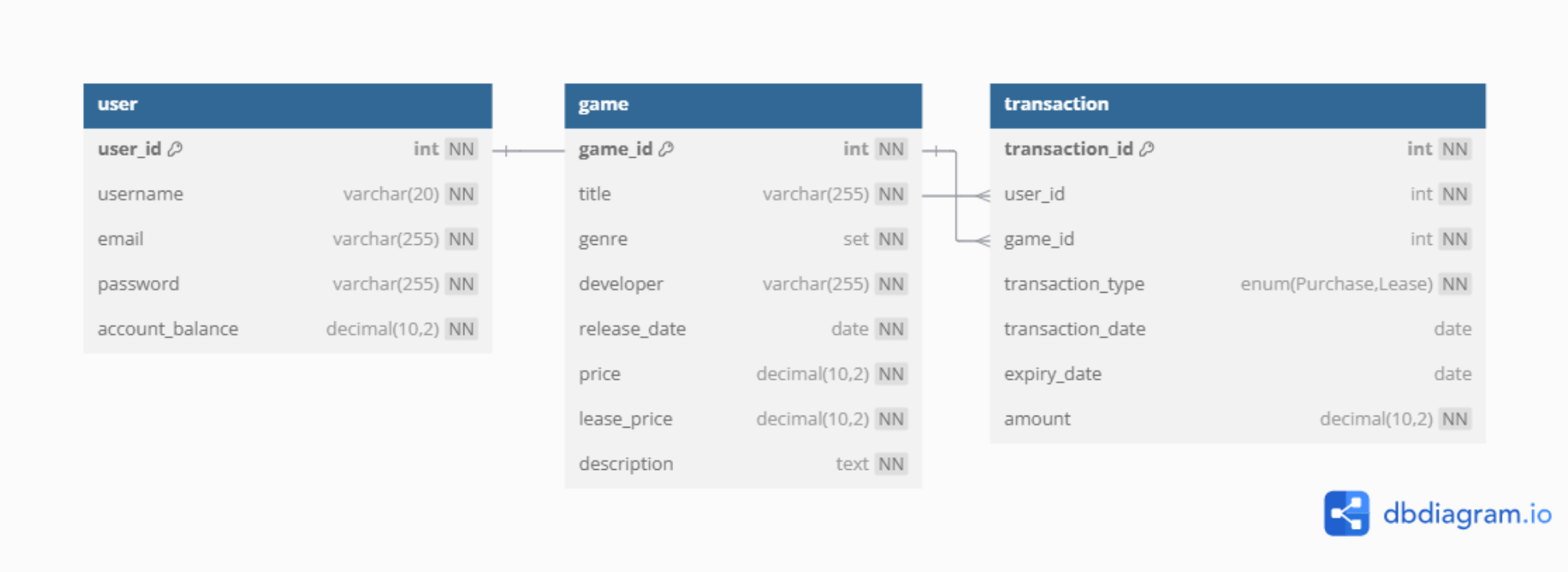
- Descripción: Devuelve todas las transacciones del usuario especificado.
- Efecto en la Base de Datos: Recupera todos los registros de la tabla Transaction asociados con el ID de usuario especificado.
- Código

```
@GetMapping("/user/{userId}")
    public List<Transaction> getTransactionsByUserId(@PathVariable int userId) {
        try {
            return transactionRepository.findAllByUserId(userId);
        } catch (Exception e) {
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
                                    "Error retrieving transactions by user", e);
        }
    }
```

Información para Acceso a los Servicios

- **Base URL**: /transactions
- **Servicios**:
    - **GET /{transactionId}**: Obtiene la información de una transacción por su ID.
        - Ejemplo: /transactions/1
    - **GET /all**: Obtiene la lista de todas las transacciones.
        - Ejemplo: /transactions/all
    - **GET /add**: Agrega una nueva transacción.
        - Ejemplo: /transactions/add?userId=1&gameId=2&transactionType=Purchase
    - **GET /delete/{transactionId}**: Elimina una transacción por su ID.
        - Ejemplo: /transactions/delete/1
    - **GET /user/{userId}**: Obtiene todas las transacciones de un usuario por su ID.
        - Ejemplo: /transactions/user/1

Diagrama Relacional de Tablas



Note: The 'genre' column is a SET type with the following possible values:

'Action', 'Adventure', 'Arcade', 'Battle Royale', 'Beat ''em Up', 'Builder', 'Card', 'Casual', 'City Builder', 'Comedy', 'Cyberpunk', 'Dating Sim', 'Detective', 'Dungeon Crawler', 'Educational', 'Endless Runner', 'Exploration', 'Fantasy', 'Fighting', 'First-Person Shooter (FPS)', 'Hack and Slash', 'Historical', 'Horror', 'Idle', 'Interactive Story', 'JRPG', 'Life Simulation', 'Management', 'Massively Multiplayer Online (MMO)', 'Metroidvania', 'Military', 'Minigames', 'Music', 'Mystery', 'Narrative', 'Open World', 'Party', 'Pinball', 'Platformer', 'Post-Apocalyptic', 'Puzzle', 'Racing', 'Real-Time Strategy (RTS)', 'Retro', 'Rhythm', 'Roguelike', 'Roguelite', 'Role-Playing Game (RPG)', 'Sandbox', 'Sci-Fi', 'Shooter', 'Social', 'Sports', 'Stealth', 'Strategy', 'Survival', 'Survival Horror', 'Tactical', 'Third-Person Shooter', 'Tower Defense', 'Trading Card', 'Turn-Based Strategy (TBS)', 'Tycoon', 'Visual Novel'

Diagrama ERD