Desarrollo web de aplicaciones entorno servidor

Spring REST Web Application Project
Digital Game Store Documentation

Vincenzo Vitulli

DAW II Bilingüe

Contents

1. Project Overview

1.1 Introduction

The Digital Game Store is a web-based application that simulates an online game store website to purchase or lease digital games.

It consists of two main components:

- A REST API backend service (digitalgamestore)
- A web-based client application (digitalgamestoreclientapplication)

1.2 System Architecture

The project uses a client-server architecture with:

- Backend: Spring Boot REST API with MariaDB database
- Frontend: Spring Boot web application with Thymeleaf templates and session-based authentication
- Database: MariaDB running in Docker container
- Session Management: HTTP Session-based authentication without Spring Security.

1.3 Technology Stack

Backend (digitalgamestore):

- Spring Boot 3.4.2
- Spring Data JPA
- Spring Web
- Docker Compose
- MariaDB
- PHPMyAdmin
- Lombok – Actuator

Frontend (digitalgamestoreclientapplication):

- Spring Boot 3.4.2
- Thymeleaf
- Bootstrap 5.3.2
- RestTemplate
- Lombok
- Spring DevTools
- Session-based authentication

1.4 Project Structure

```
Web Services Spring REST Digital Game Store/
├── DigitalGameStore.sql            # Database schema creation script
├── InsertIntoGame.sql              # Sample game data for initial setup
│
├── digitalgamestore/               # Backend REST API
│   ├── build.gradle                # Gradle build configuration
│   ├── settings.gradle             # Gradle settings
│   ├── docker-compose.yml          # Docker configuration
│   ├── src/main/java/dws/
│   │   ├── controllers/            # REST endpoints
│   │   ├── entities/               # Database entities
│   │   └── repositories/           # Data access layer
│   └── src/main/resources/
│       └── application.properties
│
├── digitalgamestoreclientapplication/  # Frontend Web App
│   ├── build.gradle                    # Gradle build configuration
│   ├── settings.gradle                 # Gradle settings
│   ├── src/main/java/com/dws/
│   │   ├── config/                 # RestTemplate configuration
│   │   ├── controllers/            # Web controllers
│   │   ├── entities/               # Data models
│   │   └── services/               # Business logic and API communication
│   └── src/main/resources/
│       ├── templates/              # Thymeleaf templates
│       └── application.properties
```

2. Database Design

2.1 Database Schema

The application uses a MariaDB database with three main tables: - User - Game - Transaction

2.2 Table Structures

User Table

```
CREATE TABLE User (
    userId INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    accountBalance DECIMAL(10,2) NOT NULL DEFAULT 0.00
);
```

Game Table

```
CREATE TABLE Game (
    gameId INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    genre SET('Action','Adventure','Arcade',...) NOT NULL,
    developer VARCHAR(255) NOT NULL,
    releaseDate DATE NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    leasePrice DECIMAL(10,2) NOT NULL,
    description TEXT NOT NULL
);
```

Transaction Table

```
CREATE TABLE Transaction (
    transactionId INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    game_id INT NOT NULL,
    transactionType ENUM('Purchase', 'Lease') NOT NULL,
    transactionDate DATE NOT NULL,
    expiryDate DATE,
    amount DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (user_id) REFERENCES User(userId),
    FOREIGN KEY (game_id) REFERENCES Game(gameId)
);
```

2.3 Key Features

- Auto-incrementing Primary Keys: All tables use auto-incrementing integer primary keys
- Foreign Key Relationships: Transaction table links to both User and Game tables
- Data Types:
  - DECIMAL(10,2) for monetary values
  - DATE for temporal data
  - ENUM/SET for constrained choices
  - TEXT for long descriptions

2.4 Docker Configuration

The database runs in a Docker container, configured through docker-compose.yml:

```yaml
services:
  mariadb:
    image: mariadb:latest
    environment:
      MYSQL_ROOT_PASSWORD: pass
      MYSQL_DATABASE: DigitalGameStore
    volumes:
      - mariadb_data:/var/lib/mysql
    ports:
      - "3306:3306"

  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    environment:
      PMA_HOST: mariadb
      PMA_USER: root
      PMA_PASSWORD: pass
    ports:
      - "8090:80"
    depends_on:
      - mariadb

volumes:
  mariadb_data:
```

3. Backend API Design (digitalgamestore)

3.1 REST API Overview

The backend provides a RESTful API with three main endpoints: - `/api/users` - User management - `/api/games` - Game catalogue management - `/api/transactions` - Transaction handling

3.2 API Endpoints

User Endpoints

```
POST   /api/users/login    # Login user with username/password
GET    /api/users          # Get all users (Not utilized by client
                             application, as this web app was not intended to
                             be for admin roles)
GET    /api/users/{id}     # Get user by ID
POST   /api/users          # Create new user
PUT    /api/users/{id}     # Update user
DELETE /api/users/{id}     # Delete user
```

Response Examples:

```
POST /api/users/login
Success (200):
{
    "userId": 1,
    "username": "user1",
    "email": "user@example.com",
    "accountBalance": 100.00
}
Error (401):
{
    "message": "Invalid credentials"
}
```

Game Endpoints

```
GET    /api/games          # Get all games
GET    /api/games/{id}     # Get game by ID
POST   /api/games          # Create new game (Not utilized by client application)
PUT    /api/games/{id}     # Update game (Not utilized by client application)
DELETE /api/games/{id}     # Delete game (Not utilized by client application)
PATCH  /api/games/{id}/price # Update game price (Not utilized by client application)
```

Transaction Endpoints

```
GET    /api/transactions           # Get all transactions (Not utilized by client
application)
GET    /api/transactions/{id}      # Get transaction by ID
GET    /api/transactions/user/{id} # Get user's transactions
POST   /api/transactions          # Create new transaction
DELETE /api/transactions/{id}     # Delete transaction
```

3.3 Entity Classes

User Entity

Key fields:

- userId (PK, auto-increment) - int
- username - String
- email - String
- password - String
- accountBalance - double

Game Entity

Key fields:

- gameId (PK, auto-increment) -int
- title - String
- genre - String
- developer - String
- releaseDate - String
- price - double
- leasePrice - double
- description - String

Transaction Entity

Key fields:

- transactionId (PK, auto-increment) - int
- userId (FK to User) -int
- gameId (FK to Game) - int
- transactionType - String
- transactionDate - String
- expiryDate - String
- amount - double

3.4 Repository Layer

The application uses Spring Data JPA repositories:

- UserRepository
- GameRepository
- TransactionRepository

Key features:

- Extends JpaRepository for CRUD operations
- Custom queries for specific operations
- Automatic query generation from method names

3.5 Error Handling

All controllers implement consistent error handling:

- 404 Not Found for missing resources
- 400 Bad Request for invalid input
- 401 Unauthorized for invalid login
- 500 Internal Server Error for server issues

Example error response:

```
{
    "timestamp": "2024-02-14T10:15:30",
    "status": 404,
    "error": "Not Found",
    "message": "User with ID 123 not found",
    "path": "/api/users/123"
}
```

Key features:

- Consistent error response format
- Detailed error messages
- Appropriate HTTP status codes
- Exception logging for debugging

4. Frontend Application Design (digitalgamestoreclientapplication)

4.1 Application Flow

1. Initial Access
    - Login page (default landing page)
    - Registration option for new users
    - Authentication required for all other pages

2. Core Features
    - User authentication and registration
    - Game catalogue browsing
    - Game purchase and leasing
    - User profile management
    - Transaction history viewing
    - Account balance management

4.2 Security Configuration

The application uses session-based authentication:

- HTTP Session management
- Plain text password handling
- URL protection through session checks
- Form-based authentication
- No Spring Security implementation

4.3 Service Layer

GameService

Handles game-related operations:

- Fetching game catalogue
- Individual game details
- Game purchases/leases

UserService

Manages user operations:

- User registration
- Profile updates
- Balance management
- Session-based authentication

TransactionService

Handles transaction operations:

- Creating purchases/leases
- Viewing transaction history
- Managing lease expirations

4.4 View Templates

Layout and Navigation

- Common navbar fragment
- Bootstrap-based responsive design, no added CSS or JavaScript
- Consistent styling across pages

Authentication Views

1. Login Page (`login.html`)
    - Username/password form
    - Registration link
    - Error messaging
2. Registration Page (`users/register.html`)
    - New user registration form
    - Validation messages
    - Login link

Game Views

1. Game Catalogue (`games/list.html`)
    - Grid/list of all games
    - Basic game information
    - Links to detail pages

2. Game Details (`games/details.html`)
    - Complete game information
    - Purchase/lease options
    - Price information

User Views

1. Profile Page (`users/profile.html`)
    - User information
    - Balance display
    - Transaction history
    - Account management options
2. Profile Edit (`users/edit-profile.html`)
    - Email update
    - Password change
    - Form validation

4.5 Controllers

WebController

Handles main navigation and view routing:

- Home page
- Error pages
- Navigation between sections

UserController

Manages user-related views and actions:

- Registration
- Profile management
- Balance operations

GameController

Handles game-related views:

- Catalogue display
- Game details
- Purchase/lease actions

5. Deployment and Configuration

5.1 Prerequisites

- Java Development Kit (JDK) 17 or higher
- Docker Desktop
- Gradle
- MariaDB (automatically handled by Spring Boot)

5.2 Backend Configuration

Application Properties

```
# Server Configuration
server.port=8080

# Database Configuration
spring.datasource.url=jdbc:mariadb://localhost:3306/DigitalGameStore
spring.datasource.username=root
spring.datasource.password=pass
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver

# JPA/Hibernate Configuration
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDialect
```

5.3 Frontend Configuration

Application Properties

```
# Server Configuration
server.port=8081

# Backend API Configuration
api.base-url=http://localhost:8080

# Thymeleaf Configuration
spring.thymeleaf.cache=false
```

5.4 Deployment Steps

1. Backend Deployment:

```
# Navigate to backend directory
cd digitalgamestore

# Run the application (Docker will start automatically)
./gradlew bootRun
```

Note: The Docker containers for MariaDB and PHPMyAdmin will be automatically started by Spring Boot thanks to the spring-boot-docker-compose dependency.

2. Frontend Deployment:

```
# Navigate to frontend directory
cd digitalgamestoreclientapplication

# Run the application
./gradlew bootRun
```

Both applications can also be run through your IDE by running their respective Application.java files.

5.5 Access Points

- Backend API: http://localhost:8080
- Frontend Application: http://localhost:8081
- PHPMyAdmin: http://localhost:8090
    - Username: root
    - Password: pass

6. Maintenance and Troubleshooting

6.1 Common Issues and Solutions

Database Connection Issues

1. Docker Container Not Running

```
# Check container status
docker ps
# Restart containers
docker-compose down
docker-compose up -d
```

2. Wrong Database Credentials
    - Verify credentials in application.properties
    - Check MariaDB root password in docker-compose.yml
    - Confirm PHPMyAdmin access
3. Port Conflicts
    - Ensure ports 8080, 8081, 8090, and 3306 are available
    - Check for other running applications using these ports
    - Modify ports if needed in configuration files

Application Issues

1. Backend Service Problems
    - Check application logs
    - Verify REST API endpoints using Postman/curl
    - Check Docker container status
    - Ensure correct port configuration

```
# Test backend API
curl http://localhost:8080/api/games
```

2. Frontend Service Problems
    - Check browser console for JavaScript errors
    - Verify backend API URL configuration
    - Clear browser cache and cookies
3. Authentication Issues
    - Clear session cookies
    - Reset password through profile page or phpMyAdmin
    - Check session validation in controllers