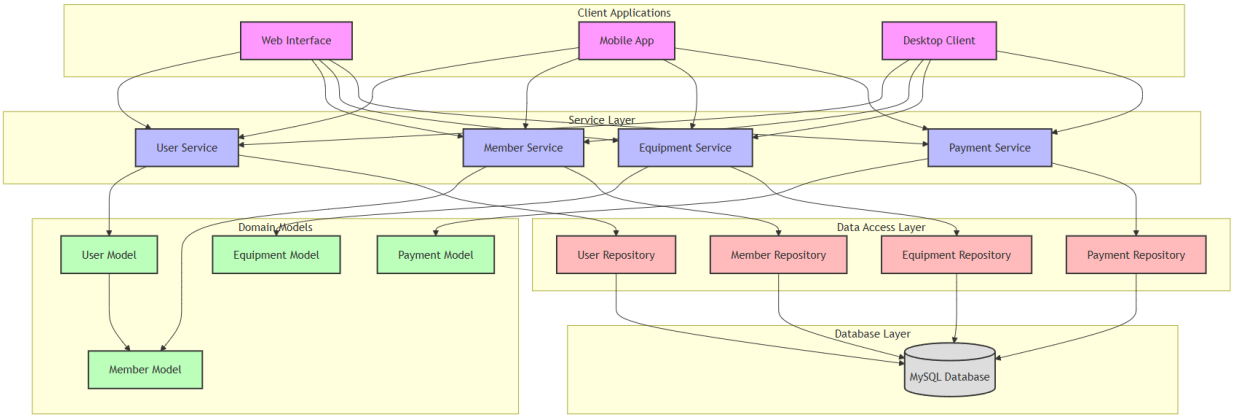
GDMS Visual Architecture



Gym Database Management System (GDMS)

A Comprehensive Java-Based Solution

Project Documentation

March 19, 2025

Contents

Introduction

1.1 Project Overview

The Gym Database Management System (GDMS) is a comprehensive software solution designed to manage gym operations efficiently. This system handles member management, equipment tracking, payment processing, and various other aspects of gym administration.

1.2 Objectives

- Streamline gym membership management
- Automate payment processing and tracking
- Monitor gym equipment maintenance
- Provide secure authentication and authorization
- Generate reports and analytics

System Architecture

2.1 Overview

The system follows a layered architecture pattern with clear separation of concerns:

- Client Layer
- Security Layer
- Business Logic Layer
- Domain Layer
- Data Access Layer
- Database Layer

2.2 Technology Stack

- Java 17
- Spring Security
- MySQL Database
- Hibernate ORM
- JUnit 5
- Maven

Implementation

3.1 Domain Models

3.1.1 Member Class

Listing 3.1: Member.java

```java
package com.gdms.model;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder
    ;
import java.time.LocalDate;

public class Member extends User {
    private String membershipType;
    private LocalDate membershipStartDate;
    private LocalDate membershipEndDate;
    private static final BCryptPasswordEncoder passwordEncoder =
        new BCryptPasswordEncoder();

    public Member(String name, String email, String password,
                  String membershipType) {
        super(name, email, passwordEncoder.encode(password));
        this.membershipType = membershipType;
        this.membershipStartDate = LocalDate.now();
        this.membershipEndDate = membershipStartDate.plusMonths(1);
    }

    public boolean isMembershipActive() {
        return LocalDate.now().isBefore(membershipEndDate);
    }

    public void renewMembership(int months) {
        if (membershipEndDate.isBefore(LocalDate.now())) {
            membershipStartDate = LocalDate.now();
        } else {
            membershipStartDate = membershipEndDate;
        }
        membershipEndDate = membershipStartDate.plusMonths(months);
    }
```

```
33
34      @Override
35      public String getRole() {
36          return "MEMBER";
37      }
38
39      // Getters and setters
40      public String getMembershipType() {
41          return membershipType;
42      }
43
44      public void setMembershipType(String membershipType) {
45          this.membershipType = membershipType;
46      }
47
48      public LocalDate getMembershipStartDate() {
49          return membershipStartDate;
50      }
51
52      public LocalDate getMembershipEndDate() {
53          return membershipEndDate;
54      }
55  }
```

3.1.2 Equipment Class

Listing 3.2: Equipment.java

```
1   package com.gdms.model;
2
3   import java.time.LocalDate;
4
5   public class Equipment {
6       private int equipmentId;
7       private String name;
8       private String status;
9       private String location;
10      private String description;
11      private LocalDate lastMaintenanceDate;
12      private LocalDate nextMaintenanceDate;
13
14      public Equipment(int equipmentId, String name, String location,
15                       String description) {
16          this.equipmentId = equipmentId;
17          this.name = name;
18          this.status = "Available";
19          this.location = location;
20          this.description = description;
21          this.lastMaintenanceDate = LocalDate.now();
22          this.nextMaintenanceDate =
23              LocalDate.now().plusMonths(1);
24      }
25
```

```
    public void performMaintenance() {
        this.lastMaintenanceDate = LocalDate.now();
        this.nextMaintenanceDate =
            LocalDate.now().plusMonths(1);
        this.status = "Available";
    }

    public boolean isMaintenanceDue() {
        return LocalDate.now().isAfter(nextMaintenanceDate);
    }

    public void reportIssue(String issue) {
        this.status = "Maintenance Required: " + issue;
    }

    // Getters and setters
    public int getEquipmentId() {
        return equipmentId;
    }

    public String getName() {
        return name;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public String getLocation() {
        return location;
    }

    public void setLocation(String location) {
        this.location = location;
    }

    public String getDescription() {
        return description;
    }

    public LocalDate getLastMaintenanceDate() {
        return lastMaintenanceDate;
    }

    public LocalDate getNextMaintenanceDate() {
        return nextMaintenanceDate;
    }
}
```

3.1.3 Payment Class

Listing 3.3: Payment.java

```
package com.gdms.model;

import java.time.LocalDateTime;

public class Payment {
    private int paymentId;
    private int memberId;
    private double amount;
    private String status;
    private String paymentMethod;
    private String transactionId;
    private LocalDateTime paymentDate;

    public Payment(int paymentId, int memberId, double amount,
                   String paymentMethod) {
        this.paymentId = paymentId;
        this.memberId = memberId;
        this.amount = amount;
        this.paymentMethod = paymentMethod;
        this.status = "Pending";
        this.paymentDate = LocalDateTime.now();
    }

    public boolean processPayment(String transactionId) {
        this.transactionId = transactionId;
        this.status = "Completed";
        return true;
    }

    public boolean refundPayment(String reason) {
        if ("Completed".equals(this.status)) {
            this.status = "Refunded: " + reason;
            return true;
        }
        return false;
    }

    public String generateReceipt() {
        return String.format("""
            Receipt
            --------
            Payment ID: %d
            Member ID: %d
            Amount: $%.2f
            Status: %s
            Payment Method: %s
            Transaction ID: %s
            Date: %s
            """,
            paymentId, memberId, amount, status,
```

```
            paymentMethod, transactionId, paymentDate);
    }

    // Getters and setters
    public int getPaymentId() {
        return paymentId;
    }

    public int getMemberId() {
        return memberId;
    }

    public double getAmount() {
        return amount;
    }

    public String getStatus() {
        return status;
    }

    public String getPaymentMethod() {
        return paymentMethod;
    }

    public String getTransactionId() {
        return transactionId;
    }

    public LocalDateTime getPaymentDate() {
        return paymentDate;
    }
}
```

Testing

4.1 Unit Tests

4.1.1 Member Tests

Listing 4.1: MemberTest.java

```java
package com.gdms.model;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import java.time.LocalDate;
import static org.junit.jupiter.api.Assertions.*;

public class MemberTest {
    private Member member;

    @BeforeEach
    void setUp() {
        member = new Member("John Doe", "john@example.com",
                            "password123", "Premium");
    }

    @Test
    void testMemberCreation() {
        assertNotNull(member);
        assertEquals("John Doe", member.getName());
        assertEquals("john@example.com", member.getEmail());
        assertEquals("Premium", member.getMembershipType());
        assertTrue(member.isMembershipActive());
    }

    @Test
    void testAuthentication() {
        assertTrue(member.authenticate("password123"));
        assertFalse(member.authenticate("wrongpassword"));
    }

    @Test
    void testMembershipActive() {
```

```
        assertTrue(member.isMembershipActive());
        // Set end date to yesterday
        member.setMembershipEndDate(LocalDate.now().minusDays(1));
        assertFalse(member.isMembershipActive());
    }

    @Test
    void testRenewMembership() {
        LocalDate originalEndDate = member.getMembershipEndDate();
        member.renewMembership(3);
        assertTrue(member.getMembershipEndDate().isAfter(
           originalEndDate));
        assertEquals(3, member.getMembershipEndDate().getMonthValue() -
                        originalEndDate.getMonthValue());
    }

    @Test
    void testGetRole() {
        assertEquals("MEMBER", member.getRole());
    }
}
```

4.1.2 Equipment Tests

Listing 4.2: EquipmentTest.java

```
package com.gdms.model;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import java.time.LocalDate;
import static org.junit.jupiter.api.Assertions.*;

public class EquipmentTest {
    private Equipment equipment;

    @BeforeEach
    void setUp() {
        equipment = new Equipment(1, "Treadmill", "Cardio Area",
                                  "Commercial Grade Treadmill");
    }

    @Test
    void testEquipmentCreation() {
        assertNotNull(equipment);
        assertEquals(1, equipment.getEquipmentId());
        assertEquals("Treadmill", equipment.getName());
        assertEquals("Available", equipment.getStatus());
        assertEquals("Cardio Area", equipment.getLocation());
    }

    @Test
    void testPerformMaintenance() {
```

```
28          LocalDate oldMaintenanceDate = equipment.getLastMaintenanceDate
                ();
29          equipment.performMaintenance();
30          assertTrue(equipment.getLastMaintenanceDate()
31                              .isAfter(oldMaintenanceDate));
32          assertEquals("Available", equipment.getStatus());
33      }
34
35      @Test
36      void testIsMaintenanceDue() {
37          assertFalse(equipment.isMaintenanceDue());
38          // Set next maintenance date to yesterday
39          equipment.setNextMaintenanceDate(LocalDate.now().minusDays(1));
40          assertTrue(equipment.isMaintenanceDue());
41      }
42
43      @Test
44      void testReportIssue() {
45          equipment.reportIssue("Motor problem");
46          assertEquals("Maintenance Required: Motor problem",
47                       equipment.getStatus());
48      }
49  }
```

4.1.3 Payment Tests

Listing 4.3: PaymentTest.java

```
1   package com.gdms.model;
2
3   import org.junit.jupiter.api.BeforeEach;
4   import org.junit.jupiter.api.Test;
5   import static org.junit.jupiter.api.Assertions.*;
6
7   public class PaymentTest {
8       private Payment payment;
9
10      @BeforeEach
11      void setUp() {
12          payment = new Payment(1, 1, 99.99, "Credit Card");
13      }
14
15      @Test
16      void testPaymentCreation() {
17          assertNotNull(payment);
18          assertEquals(1, payment.getPaymentId());
19          assertEquals(1, payment.getMemberId());
20          assertEquals(99.99, payment.getAmount());
21          assertEquals("Pending", payment.getStatus());
22          assertEquals("Credit Card", payment.getPaymentMethod());
23      }
24
25      @Test
```

```
    void testProcessPayment() {
        assertTrue(payment.processPayment("TXN123"));
        assertEquals("Completed", payment.getStatus());
        assertEquals("TXN123", payment.getTransactionId());
    }

    @Test
    void testRefundPayment() {
        payment.processPayment("TXN123");
        assertTrue(payment.refundPayment("Customer request"));
        assertTrue(payment.getStatus()
                          .startsWith("Refunded: Customer request"));
    }

    @Test
    void testGenerateReceipt() {
        payment.processPayment("TXN123");
        String receipt = payment.generateReceipt();
        assertTrue(receipt.contains("Payment ID: 1"));
        assertTrue(receipt.contains("Amount: $99.99"));
        assertTrue(receipt.contains("Transaction ID: TXN123"));
    }
}
```

Project Configuration

5.1 Maven Configuration

Listing 5.1: 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>com.gdms</groupId>
    <artifactId>gym-management-system</artifactId>
    <version>1.0-SNAPSHOT</version>

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

    <dependencies>
        <!-- JUnit Jupiter for testing -->
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
            <version>5.9.2</version>
            <scope>test</scope>
        </dependency>

        <!-- MySQL Connector -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.33</version>
        </dependency>

```

```
        <!-- Hibernate ORM -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>6.2.7.Final</version>
        </dependency>

        <!-- Spring Security -->
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-crypto</artifactId>
            <version>6.1.2</version>
        </dependency>

        <!-- SLF4J API -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>2.0.7</version>
        </dependency>

        <!-- Logback Classic -->
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>1.4.8</version>
        </dependency>

        <!-- Apache Commons Logging -->
        <dependency>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
            <version>1.2</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.11.0</version>
                <configuration>
                    <source>${maven.compiler.source}</source>
                    <target>${maven.compiler.target}</target>
                </configuration>
            </plugin>

            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>3.0.0</version>
            </plugin>
```

```
87          </plugins>
88      </build>
89  </project>
```

Future Enhancements

6.1 Planned Features

- Integration with fitness tracking devices
- Mobile application development
- Advanced analytics and reporting
- Class scheduling system
- Trainer management module
- Nutrition tracking system

6.2 Technical Improvements

- Implement caching with Redis
- Add message queuing with RabbitMQ
- Enhance security features
- Implement real-time notifications
- Add backup and disaster recovery

Conclusion

The Gym Database Management System provides a robust solution for gym management needs. The system's modular architecture ensures scalability and maintainability, while comprehensive testing ensures reliability. Future enhancements will further improve the system's capabilities and user experience.