ClubConnect - Complete Project Guide

Everything You Need to Know to Finish This Project

📋 TABLE OF CONTENTS

1. What You're Building
2. Before You Start - Setup
3. Project Structure - Creating Folders
4. Step-by-Step Implementation
5. Testing Your Work
6. Creating the Report
7. Submission Checklist

🎯 WHAT YOU'RE BUILDING

You're building **ClubConnect** - a desktop application (like Microsoft Word or iTunes, but for managing university clubs).

What It Does:

- **Students** can join clubs and RSVP to events
- **Club Leaders** can create events and manage their club
- **Admins** can approve clubs and budgets
- **Everyone** can see clubs and events

Technology:

- **Programming Language**: Java
- **User Interface**: Java Swing (makes windows, buttons, tables)
- **Database**: MySQL (stores all the data)

🔧 BEFORE YOU START - SETUP

STEP 1: Install Software (1 hour)

A. Install Java JDK

1. Go to: https://www.oracle.com/java/technologies/downloads/
2. Download **Java 11** or higher for your OS (Windows/Mac)
3. Run the installer
4. **Test it works:**
   - Open Command Prompt (Windows) or Terminal (Mac)
   - Type: `java -version`
   - You should see: `java version "11.0.x"` or higher

B. Install MySQL

1. Go to: https://dev.mysql.com/downloads/mysql/
2. Download **MySQL Community Server** for your OS
3. During installation:
   - Set root password to: `root` (or remember what you chose)
   - Choose "Use Legacy Authentication"
4. **Test it works:**
   - Open MySQL Workbench (installed with MySQL)
   - Connect to Local instance
   - You should see a connection screen

C. Install an IDE (Code Editor)

Option 1: IntelliJ IDEA (Recommended)

1. Go to: https://www.jetbrains.com/idea/download/
2. Download **Community Edition** (it's free)
3. Install it
4. Open IntelliJ → New Project → Java → Create

Option 2: Eclipse

1. Go to: https://www.eclipse.org/downloads/
2. Download Eclipse IDE for Java Developers
3. Install it

Option 3: VS Code

1. Go to: https://code.visualstudio.com/
2. Download and install
3. Install "Extension Pack for Java" from Extensions

D. Download Required Libraries

Create a folder called `Libraries` on your Desktop. Download these 3 files:

1. **MySQL Connector** (connects Java to MySQL)
   - Go to: https://dev.mysql.com/downloads/connector/j/
   - Download "Platform Independent" ZIP
   - Extract it → Find `mysql-connector-java-8.0.33.jar`
   - Copy to your Libraries folder
2. **iText** (creates PDF files)
   - Go to: https://mvnrepository.com/artifact/com.itextpdf/itextpdf/5.5.13.3
   - Click "jar" link to download
   - Save `itextpdf-5.5.13.3.jar` to Libraries folder
3. **JavaMail** (sends emails)
   - Go to: https://mvnrepository.com/artifact/javax.mail/mail/1.4.7
   - Click "jar" link to download
   - Save `mail-1.4.7.jar` to Libraries folder

You should now have 3 .jar files in your Libraries folder

🗀 PROJECT STRUCTURE

STEP 2: Create Your Project (15 minutes)

A. Create Project in IntelliJ IDEA

1. **Open IntelliJ IDEA**
2. Click **"New Project"**
3. Settings:
   - Name: `ClubConnect`
   - Location: Choose where you want it (like Desktop or Documents)
   - Language: **Java**
   - Build System: **IntelliJ** (not Maven/Gradle)
   - JDK: **11** or higher
4. Click **"Create"**

B. Add the Libraries

1. In IntelliJ, go to: **File → Project Structure** (or press Ctrl+Alt+Shift+S)
2. Click **"Libraries"** on the left
3. Click the **"+"** button → **"Java"**
4. Navigate to your Libraries folder
5. Select ALL 3 .jar files (hold Ctrl/Cmd and click each)
6. Click **"OK"** → **"Apply"** → **"OK"**

C. Create Package Structure

In IntelliJ, find the `src` folder (left side, Project view)

Right-click on `src` → New → Package

Create these packages (one by one):

```
com.clubconnect
com.clubconnect.database
com.clubconnect.models
com.clubconnect.dao
com.clubconnect.services
com.clubconnect.ui
com.clubconnect.ui.panels
com.clubconnect.ui.dialogs
com.clubconnect.utils
```

Your structure should look like:

```
ClubConnect/
└── src/
    └── com/
        └── clubconnect/
            ├── (empty for now)
            ├── database/
            ├── models/
            ├── dao/
            ├── services/
            ├── ui/
            │   ├── panels/
            │   └── dialogs/
            └── utils/
```

💻 STEP-BY-STEP IMPLEMENTATION

PHASE 1: Database Setup (30 minutes)

File 1: DatabaseManager.java

1. **Right-click** on `com.clubconnect.database` → New → Java Class
2. Name it: `DatabaseManager`
3. **Copy and paste this code:**

```
java
```

```java
package com.clubconnect.database;

import java.sql.*;

public class DatabaseManager {
    private static final String DB_URL = "jdbc:mysql://localhost:3306/";
    private static final String DB_NAME = "clubconnect_db";
    private static final String DB_USER = "root";
    private static final String DB_PASSWORD = "root"; // CHANGE THIS to your MySQL password

    private static Connection connection;

    public static Connection getConnection() throws SQLException {
        if (connection == null || connection.isClosed()) {
            connection = DriverManager.getConnection(DB_URL + DB_NAME, DB_USER, DB_PASSWORD);
        }
        return connection;
    }

    public static void initializeDatabase() {
        try {
            // Create database
            Connection conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
            Statement stmt = conn.createStatement();
            stmt.executeUpdate("CREATE DATABASE IF NOT EXISTS " + DB_NAME);
            conn.close();

            // Connect to database
            connection = getConnection();
            createTables();

            System.out.println("✓ Database initialized successfully!");
        } catch (SQLException e) {
            System.err.println("✗ Database error: " + e.getMessage());
            e.printStackTrace();
        }
    }

    private static void createTables() throws SQLException {
        Statement stmt = connection.createStatement();

        // Users table
        stmt.executeUpdate(
```

```
    "CREATE TABLE IF NOT EXISTS users (" +
    "user_id INT AUTO_INCREMENT PRIMARY KEY," +
    "student_id VARCHAR(50) UNIQUE NOT NULL," +
    "username VARCHAR(100) NOT NULL," +
    "email VARCHAR(100) UNIQUE NOT NULL," +
    "password VARCHAR(255) NOT NULL," +
    "role ENUM('ADMIN', 'LEADER', 'MEMBER', 'GUEST') DEFAULT 'MEMBER'," +
    "contact_number VARCHAR(20)," +
    "created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP," +
    "status ENUM('ACTIVE', 'INACTIVE') DEFAULT 'ACTIVE'" +
    ")"
);

// Clubs table
stmt.executeUpdate(
    "CREATE TABLE IF NOT EXISTS clubs (" +
    "club_id INT AUTO_INCREMENT PRIMARY KEY," +
    "name VARCHAR(200) NOT NULL," +
    "category VARCHAR(100)," +
    "mission_statement TEXT," +
    "description TEXT," +
    "created_by INT," +
    "initial_budget DECIMAL(10,2) DEFAULT 0," +
    "current_budget DECIMAL(10,2) DEFAULT 0," +
    "status ENUM('PENDING', 'ACTIVE', 'INACTIVE', 'ARCHIVED') DEFAULT 'PENDING'," +
    "created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP," +
    "FOREIGN KEY (created_by) REFERENCES users(user_id)" +
    ")"
);

// Club Members table
stmt.executeUpdate(
    "CREATE TABLE IF NOT EXISTS club_members (" +
    "membership_id INT AUTO_INCREMENT PRIMARY KEY," +
    "club_id INT," +
    "user_id INT," +
    "role ENUM('LEADER', 'MEMBER') DEFAULT 'MEMBER'," +
    "status ENUM('PENDING', 'ACTIVE', 'ALUMNI') DEFAULT 'PENDING'," +
    "joined_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP," +
    "FOREIGN KEY (club_id) REFERENCES clubs(club_id) ON DELETE CASCADE," +
    "FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE," +
    "UNIQUE KEY unique_membership (club_id, user_id)" +
    ")"
);
```

```
// Events table
stmt.executeUpdate(
    "CREATE TABLE IF NOT EXISTS events (" +
    "event_id INT AUTO_INCREMENT PRIMARY KEY," +
    "club_id INT," +
    "title VARCHAR(200) NOT NULL," +
    "description TEXT," +
    "event_date DATE NOT NULL," +
    "start_time TIME NOT NULL," +
    "end_time TIME NOT NULL," +
    "venue VARCHAR(200)," +
    "capacity INT," +
    "event_type ENUM('MEETING', 'SEATING', 'EVENT') DEFAULT 'EVENT'," +
    "budget_requested BOOLEAN DEFAULT FALSE," +
    "budget_amount DECIMAL(10,2)," +
    "budget_status ENUM('PENDING', 'APPROVED', 'REJECTED') DEFAULT 'PENDING'," +
    "created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP," +
    "FOREIGN KEY (club_id) REFERENCES clubs(club_id) ON DELETE CASCADE" +
    ")"
);

// RSVPs table
stmt.executeUpdate(
    "CREATE TABLE IF NOT EXISTS rsvps (" +
    "rsvp_id INT AUTO_INCREMENT PRIMARY KEY," +
    "event_id INT," +
    "user_id INT," +
    "status ENUM('ATTENDING', 'NOT_ATTENDING', 'WAITLIST') DEFAULT 'ATTENDING'," +
    "attended BOOLEAN DEFAULT FALSE," +
    "rsvp_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP," +
    "FOREIGN KEY (event_id) REFERENCES events(event_id) ON DELETE CASCADE," +
    "FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE," +
    "UNIQUE KEY unique_rsvp (event_id, user_id)" +
    ")"
);

// Resources table
stmt.executeUpdate(
    "CREATE TABLE IF NOT EXISTS resources (" +
    "resource_id INT AUTO_INCREMENT PRIMARY KEY," +
    "resource_name VARCHAR(200) NOT NULL," +
    "resource_type VARCHAR(100)," +
    "capacity INT," +
```

```
        "status ENUM('AVAILABLE', 'UNAVAILABLE') DEFAULT 'AVAILABLE'" +
        ")"
    );

    // Resource Bookings table
    stmt.executeUpdate(
        "CREATE TABLE IF NOT EXISTS resource_bookings (" +
        "booking_id INT AUTO_INCREMENT PRIMARY KEY," +
        "event_id INT," +
        "resource_id INT," +
        "booking_date DATE," +
        "start_time TIME," +
        "end_time TIME," +
        "status ENUM('CONFIRMED', 'CANCELLED') DEFAULT 'CONFIRMED'," +
        "FOREIGN KEY (event_id) REFERENCES events(event_id) ON DELETE CASCADE," +
        "FOREIGN KEY (resource_id) REFERENCES resources(resource_id)" +
        ")"
    );

    // Budget Transactions table
    stmt.executeUpdate(
        "CREATE TABLE IF NOT EXISTS budget_transactions (" +
        "transaction_id INT AUTO_INCREMENT PRIMARY KEY," +
        "club_id INT," +
        "event_id INT NULL," +
        "transaction_type ENUM('INCOME', 'EXPENSE') NOT NULL," +
        "amount DECIMAL(10,2) NOT NULL," +
        "category VARCHAR(100)," +
        "description TEXT," +
        "transaction_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP," +
        "approved_by INT," +
        "FOREIGN KEY (club_id) REFERENCES clubs(club_id) ON DELETE CASCADE," +
        "FOREIGN KEY (event_id) REFERENCES events(event_id) ON DELETE SET NULL," +
        "FOREIGN KEY (approved_by) REFERENCES users(user_id)" +
        ")"
    );

    // Announcements table
    stmt.executeUpdate(
        "CREATE TABLE IF NOT EXISTS announcements (" +
        "announcement_id INT AUTO_INCREMENT PRIMARY KEY," +
        "club_id INT," +
        "title VARCHAR(200) NOT NULL," +
        "content TEXT," +
```

```
            "created_by INT," +
            "created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP," +
            "FOREIGN KEY (club_id) REFERENCES clubs(club_id) ON DELETE CASCADE," +
            "FOREIGN KEY (created_by) REFERENCES users(user_id)" +
            ")"
        );

        // Notifications table
        stmt.executeUpdate(
            "CREATE TABLE IF NOT EXISTS notifications (" +
            "notification_id INT AUTO_INCREMENT PRIMARY KEY," +
            "user_id INT," +
            "title VARCHAR(200)," +
            "message TEXT," +
            "notification_type VARCHAR(50)," +
            "is_read BOOLEAN DEFAULT FALSE," +
            "created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP," +
            "FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE" +
            ")"
        );

        stmt.close();
        System.out.println("✓ All tables created!");
    }

    public static void exportDatabase() {
        // For now, just print message
        System.out.println("✓ Database export complete!");
    }

    public static void closeConnection() {
        try {
            if (connection != null && !connection.isClosed()) {
                connection.close();
                System.out.println("✓ Database connection closed!");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

4. **IMPORTANT**: Change line 7 - Replace `"root"` with YOUR MySQL password

PHASE 2: Create Model Classes (30 minutes)

Models are simple classes that represent data (like User, Club, Event).

File 2: User.java

Right-click `com.clubconnect.models` → New → Java Class → Name: `User`

java

```java
package com.clubconnect.models;

import java.sql.Timestamp;

public class User {
    private int userId;
    private String studentId;
    private String username;
    private String email;
    private String password;
    private UserRole role;
    private String contactNumber;
    private Timestamp createdAt;
    private UserStatus status;

    public enum UserRole {
        ADMIN, LEADER, MEMBER, GUEST
    }

    public enum UserStatus {
        ACTIVE, INACTIVE
    }

    public User() {
        this.role = UserRole.MEMBER;
        this.status = UserStatus.ACTIVE;
    }

    public User(String studentId, String username, String email, String password) {
        this.studentId = studentId;
        this.username = username;
        this.email = email;
        this.password = password;
        this.role = UserRole.MEMBER;
        this.status = UserStatus.ACTIVE;
    }

    // Getters and Setters - IntelliJ can generate these!
    // Right-click in class → Generate → Getters and Setters → Select All

    public int getUserId() { return userId; }
    public void setUserId(int userId) { this.userId = userId; }
```

```
    public String getStudentId() { return studentId; }
    public void setStudentId(String studentId) { this.studentId = studentId; }

    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }

    public String getPassword() { return password; }
    public void setPassword(String password) { this.password = password; }

    public UserRole getRole() { return role; }
    public void setRole(UserRole role) { this.role = role; }

    public String getContactNumber() { return contactNumber; }
    public void setContactNumber(String contactNumber) { this.contactNumber = contactNumber; }

    public Timestamp getCreatedAt() { return createdAt; }
    public void setCreatedAt(Timestamp createdAt) { this.createdAt = createdAt; }

    public UserStatus getStatus() { return status; }
    public void setStatus(UserStatus status) { this.status = status; }

    @Override
    public String toString() {
        return username + " (" + studentId + ")";
    }
}
```

Repeat for These Models:

File 3: Club.java - Copy this pattern:

java

```java
package com.clubconnect.models;

import java.math.BigDecimal;
import java.sql.Timestamp;

public class Club {
    private int clubId;
    private String name;
    private String category;
    private String missionStatement;
    private String description;
    private int createdBy;
    private BigDecimal initialBudget;
    private BigDecimal currentBudget;
    private ClubStatus status;
    private Timestamp createdAt;

    public enum ClubStatus {
        PENDING, ACTIVE, INACTIVE, ARCHIVED
    }

    public Club() {
        this.status = ClubStatus.PENDING;
        this.initialBudget = BigDecimal.ZERO;
        this.currentBudget = BigDecimal.ZERO;
    }

    // Generate getters/setters: Right-click → Generate → Getters and Setters

    public int getClubId() { return clubId; }
    public void setClubId(int clubId) { this.clubId = clubId; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getCategory() { return category; }
    public void setCategory(String category) { this.category = category; }

    public String getMissionStatement() { return missionStatement; }
    public void setMissionStatement(String missionStatement) { this.missionStatement = missionStatement; }

    public String getDescription() { return description; }
    public void setDescription(String description) { this.description = description; }
```

```
    public int getCreatedBy() { return createdBy; }
    public void setCreatedBy(int createdBy) { this.createdBy = createdBy; }

    public BigDecimal getInitialBudget() { return initialBudget; }
    public void setInitialBudget(BigDecimal initialBudget) { this.initialBudget = initialBudget; }

    public BigDecimal getCurrentBudget() { return currentBudget; }
    public void setCurrentBudget(BigDecimal currentBudget) { this.currentBudget = currentBudget; }

    public ClubStatus getStatus() { return status; }
    public void setStatus(ClubStatus status) { this.status = status; }

    public Timestamp getCreatedAt() { return createdAt; }
    public void setCreatedAt(Timestamp createdAt) { this.createdAt = createdAt; }

    @Override
    public String toString() {
        return name;
    }
}
```

File 4: Event.java:

```
java
```

```java
package com.clubconnect.models;

import java.math.BigDecimal;
import java.sql.Date;
import java.sql.Time;
import java.sql.Timestamp;

public class Event {
    private int eventId;
    private int clubId;
    private String title;
    private String description;
    private Date eventDate;
    private Time startTime;
    private Time endTime;
    private String venue;
    private int capacity;
    private EventType eventType;
    private boolean budgetRequested;
    private BigDecimal budgetAmount;
    private BudgetStatus budgetStatus;
    private Timestamp createdAt;

    public enum EventType {
        MEETING, SEATING, EVENT
    }

    public enum BudgetStatus {
        PENDING, APPROVED, REJECTED
    }

    public Event() {
        this.eventType = EventType.EVENT;
        this.budgetRequested = false;
        this.budgetStatus = BudgetStatus.PENDING;
    }

    // Generate ALL getters and setters here
    public int getEventId() { return eventId; }
    public void setEventId(int eventId) { this.eventId = eventId; }

    public int getClubId() { return clubId; }
    public void setClubId(int clubId) { this.clubId = clubId; }
```

```
    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }

    public String getDescription() { return description; }
    public void setDescription(String description) { this.description = description; }

    public Date getEventDate() { return eventDate; }
    public void setEventDate(Date eventDate) { this.eventDate = eventDate; }

    public Time getStartTime() { return startTime; }
    public void setStartTime(Time startTime) { this.startTime = startTime; }

    public Time getEndTime() { return endTime; }
    public void setEndTime(Time endTime) { this.endTime = endTime; }

    public String getVenue() { return venue; }
    public void setVenue(String venue) { this.venue = venue; }

    public int getCapacity() { return capacity; }
    public void setCapacity(int capacity) { this.capacity = capacity; }

    public EventType getEventType() { return eventType; }
    public void setEventType(EventType eventType) { this.eventType = eventType; }

    public boolean isBudgetRequested() { return budgetRequested; }
    public void setBudgetRequested(boolean budgetRequested) { this.budgetRequested = budgetRequested; }

    public BigDecimal getBudgetAmount() { return budgetAmount; }
    public void setBudgetAmount(BigDecimal budgetAmount) { this.budgetAmount = budgetAmount; }

    public BudgetStatus getBudgetStatus() { return budgetStatus; }
    public void setBudgetStatus(BudgetStatus budgetStatus) { this.budgetStatus = budgetStatus; }

    public Timestamp getCreatedAt() { return createdAt; }
    public void setCreatedAt(Timestamp createdAt) { this.createdAt = createdAt; }

    @Override
    public String toString() {
        return title + " - " + eventDate;
    }
}
```

PHASE 3: Utility Classes (20 minutes)

These are helper classes used throughout the project.

File 5: SessionManager.java

java

```java
package com.clubconnect.utils;

import com.clubconnect.models.User;

public class SessionManager {
    private static User currentUser;

    public static void setCurrentUser(User user) {
        currentUser = user;
    }

    public static User getCurrentUser() {
        return currentUser;
    }

    public static boolean isLoggedIn() {
        return currentUser != null;
    }

    public static void logout() {
        currentUser = null;
    }

    public static boolean isAdmin() {
        return currentUser != null && currentUser.getRole() == User.UserRole.ADMIN;
    }

    public static boolean isLeader() {
        return currentUser != null && currentUser.getRole() == User.UserRole.LEADER;
    }

    public static boolean isMember() {
        return currentUser != null && currentUser.getRole() == User.UserRole.MEMBER;
    }
}
```

File 6: PasswordHasher.java

java

```java
package com.clubconnect.utils;

import java.security.MessageDigest;

public class PasswordHasher {

    public static String hashPassword(String password) {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            byte[] hash = md.digest(password.getBytes());

            StringBuilder hexString = new StringBuilder();
            for (byte b : hash) {
                String hex = Integer.toHexString(0xff & b);
                if (hex.length() == 1) hexString.append('0');
                hexString.append(hex);
            }
            return hexString.toString();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

File 7: Validator.java

java

```java
package com.clubconnect.utils;

import java.util.regex.Pattern;

public class Validator {

    private static final Pattern EMAIL_PATTERN =
        Pattern.compile("^[A-Za-z0-9+_.-]+@(.+)$");

    public static boolean isValidEmail(String email) {
        return email != null && EMAIL_PATTERN.matcher(email.trim()).matches();
    }

    public static boolean isValidStudentId(String studentId) {
        return studentId != null && studentId.trim().length() >= 5;
    }

    public static boolean isValidPassword(String password) {
        return password != null && password.length() >= 6;
    }

    public static boolean isNotEmpty(String value) {
        return value != null && !value.trim().isEmpty();
    }
}
```

PHASE 4: Data Access (DAO) Layer (1 hour)

These classes talk to the database. You need 3 main ones to start.

File 8: UserDAO.java

This is LONG but just copy-paste it into `com.clubconnect.dao`:

```java
```

```java
package com.clubconnect.dao;

import com.clubconnect.database.DatabaseManager;
import com.clubconnect.models.User;
import com.clubconnect.models.User.UserRole;
import com.clubconnect.models.User.UserStatus;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class UserDAO {

    public boolean createUser(User user) {
        String sql = "INSERT INTO users (student_id, username, email, password, role, contact_number, status) VALUES (?,

        try (Connection conn = DatabaseManager.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {

            pstmt.setString(1, user.getStudentId());
            pstmt.setString(2, user.getUsername());
            pstmt.setString(3, user.getEmail());
            pstmt.setString(4, user.getPassword());
            pstmt.setString(5, user.getRole().name());
            pstmt.setString(6, user.getContactNumber());
            pstmt.setString(7, user.getStatus().name());

            int affectedRows = pstmt.executeUpdate();

            if (affectedRows > 0) {
                try (ResultSet rs = pstmt.getGeneratedKeys()) {
                    if (rs.next()) {
                        user.setUserId(rs.getInt(1));
                    }
                }
                return true;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false;
    }
```

```
public User authenticate(String studentId, String password) {
    String sql = "SELECT * FROM users WHERE student_id = ? AND password = ? AND status = 'ACTIVE'";

    try (Connection conn = DatabaseManager.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setString(1, studentId);
        pstmt.setString(2, password);

        try (ResultSet rs = pstmt.executeQuery()) {
            if (rs.next()) {
                return extractUserFromResultSet(rs);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
}

public User getUserById(int userId) {
    String sql = "SELECT * FROM users WHERE user_id = ?";

    try (Connection conn = DatabaseManager.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setInt(1, userId);

        try (ResultSet rs = pstmt.executeQuery()) {
            if (rs.next()) {
                return extractUserFromResultSet(rs);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
}

public boolean studentIdExists(String studentId) {
    String sql = "SELECT COUNT(*) FROM users WHERE student_id = ?";

    try (Connection conn = DatabaseManager.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
```

```
            pstmt.setString(1, studentId);

            try (ResultSet rs = pstmt.executeQuery()) {
                if (rs.next()) {
                    return rs.getInt(1) > 0;
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false;
    }

    public boolean emailExists(String email) {
        String sql = "SELECT COUNT(*) FROM users WHERE email = ?";

        try (Connection conn = DatabaseManager.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setString(1, email);

            try (ResultSet rs = pstmt.executeQuery()) {
                if (rs.next()) {
                    return rs.getInt(1) > 0;
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false;
    }

    private User extractUserFromResultSet(ResultSet rs) throws SQLException {
        User user = new User();
        user.setUserId(rs.getInt("user_id"));
        user.setStudentId(rs.getString("student_id"));
```