Database Setup Guide for the Onboarding Management Platform

In this project, we will use **MySQL** (or any similar SQL database system) as the database to store all relevant data related to companies, teams, workers, onboarding processes, and users. Below is a detailed explanation of how to set up the database, including server setup, table creation, and collaboration guidelines.

---

1. **Install MySQL Database Server**

Before working with MySQL, make sure MySQL is installed on your local machine (or you can use a shared development environment if preferred).

On Local Machine (for each developer):

- **Windows**: Download and install MySQL from the official website (https://dev.mysql.com/downloads/installer/).
- **macOS**: Install MySQL via Homebrew (https://brew.sh/) by running:

```
brew install mysql
```

- **Linux**: Install MySQL through package managers:

```
sudo apt-get install mysql-server   # For Ubuntu/Debian-based systems
sudo yum install mysql-server       # For CentOS/Fedora-based systems
```

After installation, start the MySQL service:

```
sudo service mysql start
```

2. **Start MySQL Server**

Once installed, start the MySQL service:

```
# On macOS/Linux
sudo service mysql start

# On Windows, start it from the Services app or use the MySQL Workbench
```

You can verify it's running by logging into the MySQL shell:

```
mysql -u root -p
```

After entering your password, you'll enter the MySQL shell where you can run SQL queries.

---

3. **Create Database and Tables**

Once MySQL is installed and running, you need to create a database and the necessary tables.

1. Create a New Database

Log in to MySQL:

```
mysql -u root -p
```

Create the `onboarding_db` database:

```
CREATE DATABASE onboarding_db;
```

2. Create Tables

You'll need the following tables in your database:

- **Users**: To store company admin users.
- **Companies**: To store company details.
- **Teams**: To store team data.
- **Processes**: To store onboarding processes.
- **Workers**: To store information about designated workers.
- **Stages**: To manage stages within onboarding processes.

Create tables for these entities with the following SQL commands:

```
-- Table for Companies
CREATE TABLE companies (
    company_id INT AUTO_INCREMENT PRIMARY KEY,
    company_name VARCHAR(255) NOT NULL,
    company_address TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Table for Users (Admins)
CREATE TABLE users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    company_id INT,
    username VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    role ENUM('admin', 'manager') NOT NULL DEFAULT 'admin',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (company_id) REFERENCES companies(company_id)
);

-- Table for Teams
CREATE TABLE teams (
    team_id INT AUTO_INCREMENT PRIMARY KEY,
    company_id INT,
    team_name VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (company_id) REFERENCES companies(company_id)
);

-- Table for Processes
CREATE TABLE processes (
    process_id INT AUTO_INCREMENT PRIMARY KEY,
    process_name VARCHAR(255) NOT NULL,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Table for Workers
CREATE TABLE workers (
    worker_id INT AUTO_INCREMENT PRIMARY KEY,
    team_id INT,
    worker_name VARCHAR(255) NOT NULL,
    status ENUM('pending', 'in_progress', 'completed') NOT NULL DEFAULT 'pending',
    start_date DATE,
    end_date DATE,
    FOREIGN KEY (team_id) REFERENCES teams(team_id)
);

-- Table for Stages (Part of an Onboarding Process)
CREATE TABLE stages (
    stage_id INT AUTO_INCREMENT PRIMARY KEY,
    process_id INT,
    stage_name VARCHAR(255) NOT NULL,
```

```
    stage_description TEXT,
    order_number INT,
    FOREIGN KEY (process_id) REFERENCES processes(process_id)
);
```

This setup creates all the necessary tables for tracking **companies**, **teams**, **users**, **processes**, **workers**, and **stages**.

---

4. **Set Up JDBC Configuration**

The Java backend uses JDBC (Java Database Connectivity) to connect to the MySQL database. You'll need to configure the database connection in the backend project.

Backend Database Configuration

In the `/backend/config/dbConfig.java`, you should set up the connection details for MySQL:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DbConfig {
    public static Connection getConnection() throws SQLException {
        try {
            String url = "jdbc:mysql://localhost:3306/onboarding_db";  // DB URL (localhost or server IP)
            String username = "root";  // Replace with your MySQL username
            String password = "password";  // Replace with your MySQL password
            return DriverManager.getConnection(url, username, password);
        } catch (SQLException e) {
            throw new SQLException("Connection to the database failed.", e);
        }
    }
}
```

Make sure to update the URL, username, and password according to your local MySQL setup.

---

5. **Who Needs to Set Up the Database?**

- **One Developer** (typically the project leader or someone in charge of backend setup) should set up the initial database schema, including creating the database and tables.
- **Other Developers** can use the same local database setup to test and work on their part of the project (assuming they are all working on the same local setup).

Alternatively, if you are using a **shared database** or **cloud database** (e.g., AWS RDS, Google Cloud SQL), the database setup will need to be done by the person managing the cloud infrastructure, and you can all share the connection details (username, password, host, etc.).

---

6. **Do All Developers Work on Localhost?**

- **Local Development Setup**: All developers can work on localhost, using their individual local installations of MySQL for database management. This makes it easier for everyone to work independently, especially if different developers are working on different parts of the application (e.g., one on the frontend and one on the backend).

- **Shared Database Setup**: If you're working in a team and prefer not to configure local databases individually, you can use a **shared cloud-based database** (e.g., AWS RDS, DigitalOcean Managed Databases) that all developers can access. In this case, all developers would work on the same database, and you only need one MySQL instance running on the cloud.
- **Database Synchronization**: If working locally, be sure to use version control (Git) for any SQL schema changes, and communicate any table structure changes (e.g., adding new columns, modifying relationships) with the team so that everyone updates their local databases accordingly.

---

7. **Handling Data Consistency Across Multiple Developers**

If multiple developers are working on the database, you need to establish a process to ensure data consistency:

- **SQL Migrations**: You can use a version control system like Git to keep track of SQL schema changes. Create migration scripts that can be run by all developers to keep their local databases synchronized.
- **Communication**: Regularly discuss any updates or changes to the database schema with the team.

---

8. **Additional Tips**

- **Backup Your Database**: Always back up your database before making significant changes, especially when modifying table structures.
- **Testing**: Ensure you have proper test data in the database for developers to test the application as they work.
- **Use MySQL Workbench**: For developers who are more comfortable with GUI tools, **MySQL Workbench** provides a user-friendly interface to manage databases and tables.

Summary

- **Install MySQL** locally or use a shared cloud instance.
- **Create the database** and **tables** (follow the SQL schema provided).
- **Set up the JDBC connection** in the backend project to connect to the database.
- Developers can work on **localhost** or a **shared database** depending on the team's preferences.
- Make sure to sync database changes (schema, data) across the team.

With this setup, each team member will be able to connect to the database and manage the application's data accordingly.

If you've forgotten the MySQL root password, there are ways to reset it, depending on your operating system. Here's a comprehensive guide for each platform:

---

1. Resetting the MySQL Root Password on Linux (Ubuntu/Debian)

Step 1: Stop the MySQL Service

First, stop the MySQL service using the following command:

```
sudo service mysql stop
```

Step 2: Start MySQL in Safe Mode

Start MySQL without loading the grant tables (which store password information):

```
sudo mysqld_safe --skip-grant-tables &
```

This will allow you to log in to MySQL without a password.

Step 3: Log In to MySQL

Log into MySQL as the root user:

```
mysql -u root
```

Step 4: Reset the Root Password

Once you're logged in, you can reset the password. Use the following SQL command to change the root password. Replace `newpassword` with the new password you want to set.

```
USE mysql;
UPDATE user SET authentication_string=PASSWORD('newpassword') WHERE User='root';
FLUSH PRIVILEGES;
```

For MySQL 5.7.6 or higher, use the following SQL query to reset the password:

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'newpassword';
```

Step 5: Restart MySQL

Exit MySQL:

```
exit;
```

Then restart the MySQL service:

```
sudo service mysql restart
```

Step 6: Log In With the New Password

You should now be able to log in to MySQL using the new root password:

```
mysql -u root -p
```

---

2. Resetting the MySQL Root Password on macOS (Homebrew)

Step 1: Stop MySQL

First, stop the MySQL server:

```
brew services stop mysql
```

Step 2: Start MySQL in Safe Mode

Start MySQL in safe mode with the `--skip-grant-tables` option:

```
sudo mysqld_safe --skip-grant-tables &
```

Step 3: Log In to MySQL

Log into MySQL without a password:

```
mysql -u root
```

Step 4: Reset the Password

Now that you're inside MySQL, reset the password with the following commands:

```
USE mysql;
UPDATE user SET authentication_string=PASSWORD('newpassword') WHERE User='root';
FLUSH PRIVILEGES;
```

For newer MySQL versions (5.7+):

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'newpassword';
```

Step 5: Restart MySQL

Exit MySQL:

```
exit;
```

Restart MySQL:

```
brew services restart mysql
```

Step 6: Log In with the New Password

Now you should be able to log in with the new password:

```
mysql -u root -p
```

---

3. Resetting the MySQL Root Password on Windows

Step 1: Stop the MySQL Service

Go to **Control Panel > Administrative Tools > Services**. Find the **MySQL service** (it may be named something like `MySQL` or `MySQL57`) and stop it.

Alternatively, you can stop the MySQL service from the command line:

```
net stop mysql
```

Step 2: Start MySQL in Safe Mode

Start MySQL with the `--skip-grant-tables` option. Open a Command Prompt window and navigate to your MySQL `bin` folder (where MySQL is installed). Run the following command:

```
mysqld --skip-grant-tables
```

This will start MySQL without asking for a password.

Step 3: Log In to MySQL

Open a new Command Prompt window and log in to MySQL:

```
mysql -u root
```

Step 4: Reset the Password

Once logged in, change the password with the following SQL commands:

```
USE mysql;
UPDATE user SET authentication_string=PASSWORD('newpassword') WHERE User='root';
FLUSH PRIVILEGES;
```

For MySQL 5.7.6 or higher:

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'newpassword';
```

Step 5: Restart MySQL Service

Exit MySQL:

```
exit;
```

Then restart the MySQL service either through the **Services** window or by running:

```
net start mysql
```

Step 6: Log In with the New Password

Now you can log in with the new root password:

```
mysql -u root -p
```

---

4. Resetting the MySQL Root Password on Docker (If using MySQL Docker Container)

If you're using MySQL inside a Docker container, follow these steps:

Step 1: Stop the MySQL Docker Container

Stop the running MySQL container:

```
docker stop <container_name>
```

Step 2: Run MySQL Container with `--skip-grant-tables`

Start the MySQL container in safe mode with the `--skip-grant-tables` option:

```
docker run --name mysql-safe -e MYSQL_ROOT_PASSWORD=root -d mysql:latest --skip-grant-tables
```

Step 3: Access MySQL

Enter the MySQL container:

```
docker exec -it <container_name> bash
```

Log in to MySQL:

```
mysql -u root
```

Step 4: Reset the Password

Once inside MySQL, reset the password:

```
USE mysql;
UPDATE user SET authentication_string=PASSWORD('newpassword') WHERE User='root';
FLUSH PRIVILEGES;
```

For MySQL 5.7.6 or higher:

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'newpassword';
```

Step 5: Restart the MySQL Container

Exit MySQL:

```
exit;
```

Stop and restart the container:

```
docker stop <container_name>
docker start <container_name>
```

Step 6: Log In with the New Password

Now you should be able to log in:

```
docker exec -it <container_name> mysql -u root -p
```

---

5. Additional Notes

- **Backup**: Always make sure to back up your database before resetting passwords to avoid accidental data loss.

- **MySQL Version**: MySQL 5.7 and higher use `ALTER USER` for password resetting, whereas older versions use `UPDATE user` with `PASSWORD()`.
- **Security**: Once you've reset your password, make sure to restart MySQL normally (without `--skip-grant-tables`) to ensure your system is secure.
- **Permissions**: If you're using a cloud database or a restricted environment, you may need to contact the database administrator for assistance.

By following the appropriate steps based on your operating system, you will be able to reset your MySQL root password and regain access to your MySQL server.