SMART BEEHIVE

Achref Bouali / Fehmi Bejaoui

ISET RADES

Introduction

This project is a small web application I made to help manage beehives, farmers, and their sites. The idea came from wanting to create something useful that also lets me practice working with Java, Servlets, JSP, and databases.

The goal of the app is to let the user view and manage data about farmers, where their beehives are located (the sites), and how many hives they have. I used a basic MVC (Model-View-Controller) structure to organize the code and followed a clean structure with packages for entities, DAO, controllers, and JSP pages for the frontend.

Even though it's a small project, it helped me learn how to connect Java code with a MySQL database and build a simple dashboard where a user can pick a farmer and see how many sites and hives, they own. It also showed me how to handle small problems that come up when building a web app from scratch.

Part 1: Database Creation and Queries

Before writing any Java code, I started by creating the database. Since the app is about beehives and the farmers who manage them, I created three main tables: farmer, apiary site, and beehive. These tables are connected using foreign keys, which helps link hives to sites, and sites to farmers.

Tables:

- **farmer**
  This table stores basic details about each farmer: their first and last name, email, and phone number.
- **apiary site**
  Each farmer can have multiple apiary sites. A site has details like location (latitude, longitude, altitude), setup and closure dates, and a foreign key farmer_id linking it to the farmer who owns it.
- **beehive**
  This table stores information about each hive, including its name, type, number of extensions, and the agent responsible for it. Each hive is linked to a site using the site_id foreign key.

```
CREATE TABLE farmer (
  id INT(11) PRIMARY KEY AUTO_INCREMENT,
  first_name VARCHAR(100),
  last_name VARCHAR(100),
  email VARCHAR(100),
  phone VARCHAR(20));

CREATE TABLE apiarysite (
  id INT(11) PRIMARY KEY AUTO_INCREMENT,
  site_name VARCHAR(100),
  latitude DOUBLE,
  longitude DOUBLE,
  altitude DOUBLE,
  setup_date DATE,
  closure_date DATE,
  farmer_id INT(11),
  FOREIGN KEY (farmer_id) REFERENCES farmer(id));

CREATE TABLE beehive (
  id INT(11) PRIMARY KEY AUTO_INCREMENT,
  hive_name VARCHAR(100),
  site_id INT(11),
  hive_type VARCHAR(50),
  extension_count INT(11),
  responsible_agent VARCHAR(100),
  FOREIGN KEY (site_id) REFERENCES apiarysite(id));
```

Part 2: Entities and MVC Architecture

After setting up the database, I started building the Java application using the **MVC (Model-View-Controller)** pattern. This helped me organize the code and keep things clean.

Entities

For each table in the database, I created a Java class (also called an entity) to represent it. These classes have the same fields as the database columns and are used to send or receive data from the database.

1. Farmer.java

This class has:

- id,firstName,lastName,email,phone as attributes
- constructors (we need more than one for different use)
- setters and getters

2. ApiarySite.java

This class has:

- id,siteName,latitude,longitude,altitude,setupDate,closureDate,farmerId (foreign key)
- constructors (we need more than one for different use)
- setters and getters

3. Beehive.java

This class has:

- id,hiveName,siteId (foreign key),hiveType,extensionCount,responsibleAgent

These entities are placed in the entity package of my project.

MVC Structure

To keep things well structured, I organized my project into several packages:

- **entity** – for the Java classes that represent tables
- **dao** – for the database operations (CRUD)
- **controller** – for servlets that handle HTTP requests
- **model** – for logic
- **utility** – for things like database connection (using a singleton)

How the MVC Works in My App

Here's how it all works together when, for example, a user selects a farmer:

1. **View (JSP Page):**
   The servlet forwards the data to a JSP page, which displays the number of sites and hives for the selected farmer.
2. **Controller (Servlet):**
   The form sends the request to a servlet (like DashboardController.java). The servlet reads the selected farmer ID and calls the DAO.
3. **DAO (Data Access Object):**
   The DAO (e.g., FarmerDao.java) connects to the database and runs a query to get the sites and hives for that farmer.
4. **Entity:**
   The data is stored in Java objects (like Farmer, ApiarySite, Beehive) and passed back to the servlet.
5. **Model**
   The servlet forwards the data to a JSP page, which displays the number of sites and hives for the selected farmer.

Part 3: The Dashboard

To make the project more interactive and useful, I added a dashboard page. The goal was to allow the user to choose a farmer and instantly see how many sites and beehives that farmer has.

How it works

- The dashboard has a form with a datalist input where the user can select a farmer by name.
- When the user clicks the “View” button, the form sends a request with the selected farmer’s ID.
- In the backend, I used a servlet (DashboardServlet) to handle the request. It uses the DashboardDao class to run two SQL queries:

Query to count how many sites the farmer owns:

This one looks at the apiarysite table and filters by the selected farmer’s ID.

SELECT COUNT(*) FROM apiarysite WHERE farmer_id = ?

Query to count how many hives the farmer has:

This one joins the beehive and apiarysite tables to count only the hives that belong to the farmer’s sites.

SELECT COUNT(*) FROM beehive h

JOIN apiarysite s ON h.site_id = s.id

WHERE s.farmer_id = ?

- These numbers are then sent back to the JSP page and shown under a small “Results” section.

Part 4: Small Problems I Faced

During this project, I ran into a few small problems that took some time to solve. Here's one important issue I want to mention:

The JDBC Driver Problem

To connect my Java application to the MySQL database, I had to add the mysql-connector-j.jar file in two different places:

- First, under the lib folder of my project so that the program can use it when running.
- Second, under the Module Path in my project's build path so that the IDE and compiler know where to find it.

If I forgot to add it in either place, I kept getting errors when trying to connect to the database. This took me some time to figure out.

Part 5: Notes on What I Used

For this project, I tried to make everything as clear and personal as possible. One thing I did differently was with the CSS design. Instead of using Bootstrap, which is common but feels a bit generic, I wanted something more unique and customized to fit the beekeeping theme.

So, I used AI tools to help me create a CSS custom that looks nicer and fits better with the project's idea. This helped me get a softer, more natural color palette and style that feels like honey and bees.

Also, I used AI to help me check and correct some mistakes in my code. It was useful for catching small errors that took me a while to figure out especially in the dashboard.