eCommerce Systems / Professor H Roumani

PROJECT D

Executive Summary

In this project we take the first step in our view migration journey by creating a webapp whose views are generated via JSP. The project will expose the key elements of JSP and leverage advanced features in Tomcat such as persistence scopes, filters for ad-hoc changes, and listeners for analytics. The project will also expose some of the Map APIs in Google's cloud platform.

Reference Implementation

In order to clarify the requirement and provide a feel for the sought functionality, here is a link to the complete project in production:

`http://red.eecs.yorku.ca:4413/44F18_ProjD/Dash.do`

Your completed project should have the same behaviour as this one. Take some time to explore this webapp. Note that the reference implementation uses the responsive styling of the Bootstrap framework plus some minor tweaks of my own. These styles reflect personal preferences and are *not* part of the requirement of this project. As such, you are welcome to:

- Ignore styling altogether and focus on behaviour.
- Copy my styles (from the source of any html page) as is.
- Copy my styles but personalize (e.g. change the images).
- Create your own styles.

Naming Convention

It is critical that you adhere to the following naming convention:

- The model package is named model and it contains:
    - The model's singleton class: Engine.java.
    - The bean: StudentBean.java.
    - The data access object: StudentDAO.java.

- The controller package is named ctrl and it contains:
    - The Dash.java servlet with URL mapping /Dash.do.
    - The Prime.java servlet with URL mapping /Prime.do.
    - The Gps.java servlet with URL mapping /Gps.do.
    - The Drone.java servlet with URL mapping /Drone.do.
    - The Ride.java servlet with URL mapping /Ride.do.
    - The Sis.java servlet with URL mapping /Sis.do.

- The JSP views under WebContent and it contains:
    - The Dash.jspx view.
    - The Prime.jspx view.
    - The Gps.jspx view.
    - The Drone.jspx view.
    - The Ride.jspx view.
    - The Sis.jspx view.
- Method Names:
    - The public methods in Engine.java that are invoked by servlet X should be named doX.
    - The public methods in StudentBean must be named in accordance with the Java Bean Naming convention given the private attributes: String name, String major, int courses, and double gpa.
    - The public method in StudentDAO must be called retrieve and its return type must be List<StudentBean>.

The Services

- **S0: Dashboard**
  Serving as a main menu for the webapp.
- **S1: Prime Number Finder**
  Find the next prime in a given range one by one (prompting in between).
- **S2: GPS Distance Calculator**
  Compute the distance (in km) between two points on the Earth surface given the latitude and longitude of each.
- **S3: Drone Delivery Timer**
  Compute the time it takes an average delivery drone to make a delivery given the starting and ending street addresses in a city.

- **S4: Ride Cost Estimator**
  Compute the cost of a ride (e.g. Uber) given its starting address and destination after factoring in the current traffic delays.
- **S5: SIS Report Generator**
  Generate a report from a Student Information System filtered based on a name prefix, a major, and a minimum GPA. The output is sorted on a custom column.

The Database

Your webapp must fetch the data (for the Sis service) from a local database (running on the same machine as the webapp). Its data can be accessed via the following specs:

- Protocol: jdbc//derby.
- Host: localhost.
- Port: 64413.
- Database: EECS.
- Credentials: as in Project-B.
- Table: SIS.
- Columns: SURNAME, GIVENNAME, MAJOR, COURSES, and GPA. (Note that the dao must translate the database terms to the business term "name".)

To start the database server, issue the command: derby_start and to stop it use: derby_stop. This works in your VBox and in a lab workstation.

Implementation Notes

1. For S1, use the nextProbablePrime method in BigInteger.
2. For S2, note that the longitude (n) and latitude (t) are expressed as signed decimal degrees with East of Greenwich being a negative longitude and south of the Equator being a negative latitude (e.g. Toronto has: t=+43.7, n=+79.4 whereas Sydney has: t=-33.9, n=-151.2). The distance (in km) between two such points is given by:

   ```
   12742 * atan2[sqrt(X), sqrt(1-X)], where:
   X = sin²[(t2-t1)/2] + Y * sin²[(n2-n1)/2] and
   Y = cos(t1) * cos(t2)
   ```

   Note that the four coordinates must be in radians; i.e. start by multiplying each of the four entries by π/180.0.
3. For S3, use the Google Cloud Platform to convert each of the two addresses to latitude and logitude via an http connection to:

   ```
   https://maps.googleapis.com/maps/api/geocode/xml?address=address&key=...
   ```

   (Replace xml with json if you prefer.) Once you extract the lat and lng of each end, use S2 to compute the distance between them. Given the distance, you can compute the delivery time in minutes by assuming an average delivery drone cruising speed of 150km/h.
4. For S4, use the Google Cloud Platform to determine the duration of the drive between the two given addresses given the current traffic condition. The URL is:

   ```
   https://maps.googleapis.com/maps/api/distancematrix/xml?origins=from&destinations=dest&departure_time=now&key=...
   ```

   (Replace xml with json if you prefer.) The duration_in_traffic field gives you the trip time in seconds. To compute the cost, adopt the (admitedly simplistic) model of 50 cents per minute (more on this in lecture).
5. For S5, have your model instantiate the DAO in its constructor. The DAO must use JDBC to communicate with the database; populate a list of beans while translating column names to business names; and return the result. You can have two (overloaded) methods, if you like to support the sorting option or ignore it.

Telemetery & Ad-hoc Changes

Add two features to your webapp:

- A filter named October.java in a new package named adhoc with the following (temporary) functionality:

  1. Block the Ride service completely by serving a page that informs the user that this service is not available with a link to the Dashboard.
  2. Block any request for sorting in the SIS service. If any sorting is requested then serve a page that informs the user that this sorting is not available with a link to the Dashboard.

  The functionality must of course be inserted (or removed) w/o touching the rest of the webapp.

- A listener named Monitor.java in a new package named: analytics with the following tasks:

  1. Determine the percentage of times the Drone service is used.
  2. Determine the likelihood that a user will use both S3 and S4.

You will need a new servlet named Admin.java mapped to /Admin to enable admin personnel to access your statistics.

Persist Your Work

Follow these steps to persist / backup your work:

- Right-click your project (in the Project Explorer) and select Export.
- Select **WAR** File.
- Provide a destination as ProjD.war on the Desktop).
- Check the box to *Export the source files* (extremely important)

Now copy the created war file to your Google Drive, DropBox, or some other cloud service, or copy it to a USB drive.

Fact

In the last Programming Test, **three** students (out of 40) submitted war files without source, and **two** submitted zip archives instead of war files for Project-B.

Reflections

- Scalability-wise, do you see a problem (vis-a-vis the jdbc connection) in how S5 is implemented?
- Security-wise, critique the way the Sort By feature is implemented. Do you see a vulnerability in it.

• EECS4413 • Professor Roumani • Don't miss the forest for the trees •