Cupcake mini project

08.10.2017

Stanislav Novitski - cph-sn183@cphbusiness.dk
Mathias Bigler - cph-mb493@cphbusiness.dk
Alexander W. Hørsted-Andersen - cph-ah353@cphbusiness.dk
Mikkel Emil Larsen - cph-ml474@cphbusiness.dk

Introduction

This report describes the cupcake mini project - a simple webshop, where you can buy cupcakes. The webshop consists of a MySql database, java servlets on the backend and jsp pages, html, css and javascript on the frontend.

Background

The webshop could be used by a small company to sell cupcakes online to customers.

The customer should be able to order one or more cupcakes, by combining a bottom and a topping part, and add the complete cupcake to a shopping cart, before placing the order.

The owner(admin) of the store should also be able to view all the previous orders.

Choice of technology

We have chosen Netbeans 8.2 as our IDE program, to develop this web application. Our chosen programming language is Java 8.0, for the connections to the database, we have chosen JDBC 4.3. The used database for this application is MySql 5.7, and to deploy the project we have used Apache Tomcat 8.0.x.

The frontend part of the project is created mostly of JSP 2.3 pages, and styled with CSS3, and Bootstrap 4. Some of the pages contain JavaScript specially the shop has been made more dynamic and delegates some of the work to the client, instead of the servlets.

- IDE
  - Netbeans 8.2

- Backend
  - Java version 8.0
  - JDBC 4.3
  - MySql 5.7
  - Apache Tomcat 8.0.x

- Frontend
  - HTML 5.1
  - CSS3
  - Bootstrap 4

- JSP 2.3
- JavaScript 1.5

Demands

- Not yet implemented

ER diagram

Our ER diagram consists of the following tables, ‘user’, ‘order’, ‘orderline’, ‘bottom’ and ‘topping’.

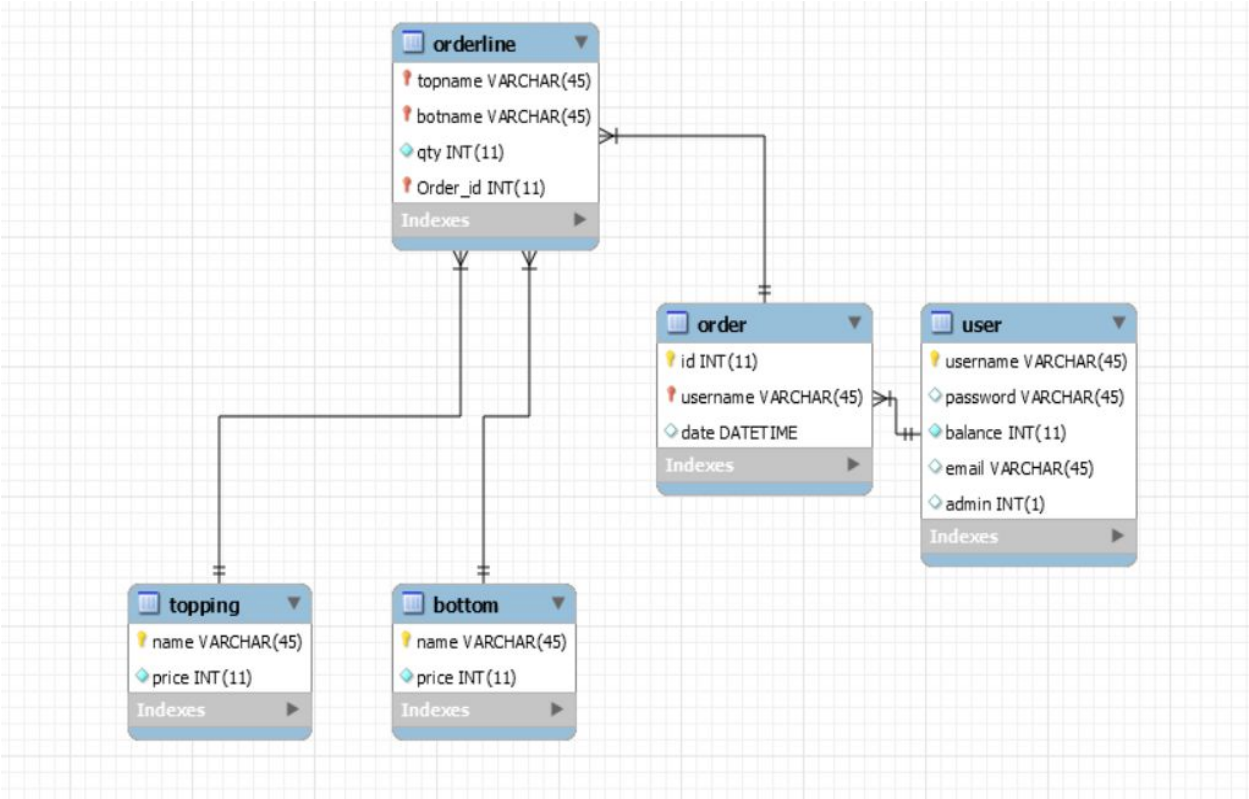
A row in the user table consists of one user and it’s details. One user can have many orders, and one order can consist of many orderlines. A orderline is a junction table with a concatenated primary key consisting of a bottom name, topping name and the order id. A bottom/topping is made of a name and a price.

To fulfill the first normalization form we have structured our tables to only contain atomic values. All tables have a primary key.

To fulfill the second normalization form we have a junction table (orderline) with a concatenated key and a quantity that depends on the entire concatenated key.

Our non-automatic generated ID’s are present in the user, topping, bottom tables. We chose to do this in the user table, because we want unique usernames. As well in the bottoms and toppings tables, we want unique bottom- and topping names. The reasoning behind that, is that there is no need to have more than one of each topping or bottom, and it should be names that are sensible.

ER Diagram

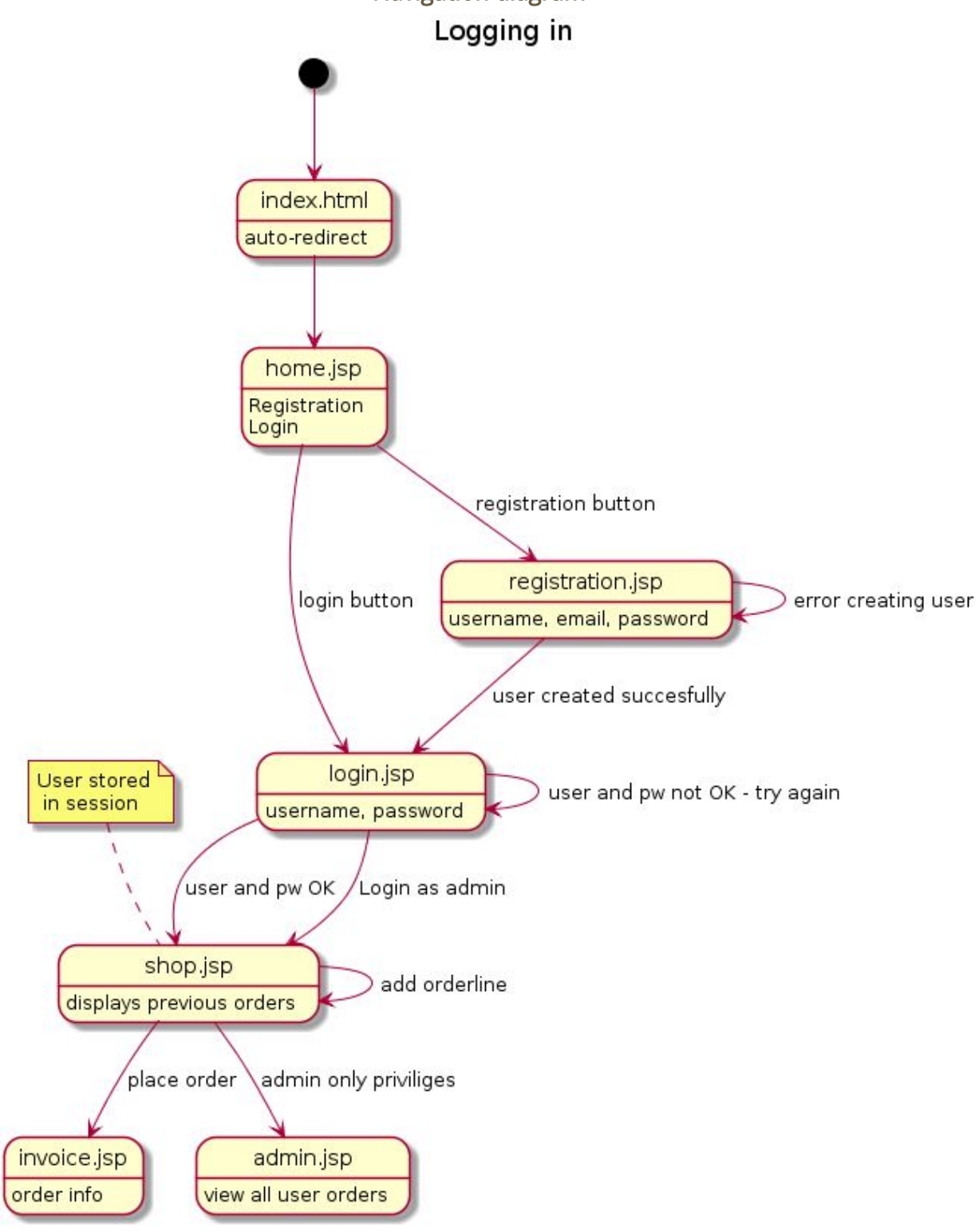


State/Navigation diagram

The state/navigation diagram shows how the user navigates the .html and .jsp sites of the webshop.

When the user accesses the webshop, the user is auto-redirected from the index.html startpage to the home.jsp page. From here the user can navigate to registration.jsp (through the Controller servlet) to register a new user, and/or to login.jsp (through the Controller servlet) to login to the a user account. From the login.jsp page, the user is redirected the shop.jsp page (through the Controller servlet), where the user can place cupcake orders. If an order is placed, the user is redirected to invoice.jsp (through the ShopController servlet). If the user is logged in as admin, the user can navigate from shop.jsp to admin.jsp (through the ShopController servlet), to view order details for all user orders.

Navigation diagram



Sequence diagram

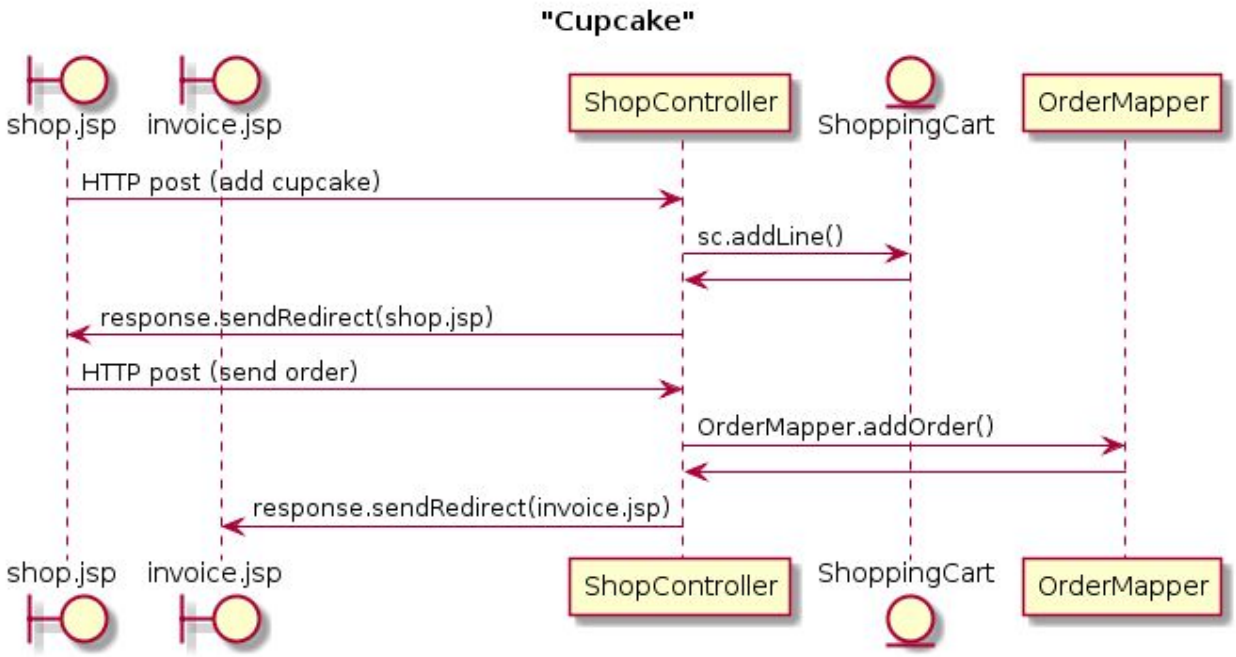
Our sequence diagram describes the process from adding a cupcake to the shopping cart, to ending up on the confirmation page, invoice.jsp.

Once you've chosen a bottom, topping and quantity, you click the add button. With this HTTP post you send a hidden input field named 'origin' with the value of 'addline'. The post is sent to the servlet 'ShopController'. The 'origin' input field is used to structure our entire ShopController servlet, with the use of a switch with different cases. With this HTTP post you end up in the 'addline' case of the switch. In this case you obtain the parameters 'bottom', topping' and 'quantity, which was sent with the HTTP post. Using the parameter values you construct a orderline object which consists of a bottom object, topping object and a quantity.

Next you add the orderline object to the shopping cart, and then we update the 'shoppingcart' attribute stored on the session with our newly updated shoppingcart. Finally we're redirected back to the shop page.

You will then click on the 'order' button, which sends another HTTP post to the 'ShoppingCart' servlet with the 'origin' parameter with the value of 'order'. You will this time end up in the 'order' case of the switch. In this case you have a if condition checking whether the total price of the shopping cart exceeds your balance. But our logic here is partly broken, so the if condition doesn't work.

Using the OrderMapper we add the order to the database. Finally you are redirected to the confirmation page, invoice.jsp, which displays the placed order.



Special cases

Information stored in the session

To help the program remember an user after the inputted credentials have been verified, a User object is saved in the session. This user object should be removed from the session when a 'log out' occurs. A ShoppingCart, which contains Orderlines, is also saved on the session from the moment a login has been verified. This helps the program remember changes a user makes for the ShoppingCart when shifting between the Java Servlet page "ShopController.java" and the JSP page "shop.jsp".

On the "shop.jsp" page all the Toppings and Bottoms and their price is presented for the logged-in user. The information about the Toppings and Bottoms is also stored in the session. This prevent too many interactions with the database, so that only one interaction regarding information about the different Toppings and Bottoms, is made to the database.

Input validation

When a user is prompted with input fields at the "registration.jsp" and "login.jsp" pages, the user has to input a username, password and an email. This information is validated, using regex, in the JavaScript page "FormValidation.js". This is to prevent the use of special characters and/or obtain a correct syntax. The syntax problem is primarily a problem for the email. We could have left the validation out for the email input since html contains a built-in 'email-validation'.

If the user inputs pass in the "FormValidation.js" validation, they are passed on to the "Controller.java" Java Servlet page. Here the inputted username is checked against all the username contained in the database. How the validation in the "Controller.java" page is handled is different from these two following scenarios:

- A user wants to login: If a User exists, in the database, with the specified username, a User object is created containing the username, password, email and balance. Now the specified password can be checked against the password in the User object. If a match is found, the user is redirected to the "shop.jsp" page.
- A user wants to register: If a User do not exist with the specified username, a User object is created from the information inputted at the "registration.jsp" page, and then stored in the database.

Login security

It has come to our attention that our login page, is not secure. The reason is that someone can set up a "sniffer"/initiates a "man in the middle attack" and get the login information from our POST, when we apply it. The solution to that would be to use a HTTPS service, with SSL security integrated in our web application.

User types in database

We have a simple “boolean”-like usertype definition in our ‘User’ database table, a column called “admin”. An admin user has the Integer value of 1 and a normal user 0. The admin user type grants special admin privileges on the webapp, currently the ability to view cupcake orders for all users. In the current implementation, the admin user type can only be created/altered directly through the database.

Status on implementation

- We have implemented all the jsp-sides that our navigations diagram describes.
- We have implemented all CRUD operations relevant for this project.
- We have styled all our jsp pages, some with “pure” CSS and some with Bootstrap.
- We’ve found a part of our system that have some broken logic. When a shopping cart is assembled and it’s total price exceeds the user’s balance, it should give the user a warning, but it doesn’t.
- No tests.
- There is a little error, when you login as admin, and view a specific invoice, when you press the ‘back’ button, you get redirected to the admin.jsp, rather than the shop.jsp.
- We also realized that we have used an Integer variable to represent our balance and prices in general. This is definitely a fault in our initial planning of the application. This is an issue, because the decimals are lost in the transactions, and it would be difficult to add taxes, campaigns etc.
- We are also missing a logout function, that could have been practical, but if you login with a different user, the session is refreshed for that user, so you can manually change users, that way.

Test

We haven’t performed any kind of tests.