KTH H16P02 Network Programming With Java: Project - APG Webstore

Gretar Atli Gretarsson (gretar@kth.se)
Thorsteinn Thorri Sigurdsson (ttsi@kth.se)

January 10, 2017

1 Introduction

1.1 Task specification

The task was to create a working webshop that sells gnomes. The webshop was supposed to be implemented using the Java EE technologies. The implementation was broken into the following three phases.

1.1.1 Customer management phase

Each customer must be able to register to the system providing username and password. The username must be an unique email address and the password must be longer than 7 characters. No two customers can have the same username. The customer must be able to login and out of the system using the previously chosen email and password.

1.1.2 Inventory and shopping basket management phase

Inventory - The website must render the gnomes catalog and the inventory status of each gnome available must be displayed. The customer must be able to browse through the gnomes catalog and be able to add a chosen gnome to the shopping cart.

Shopping cart - When a customer logs into the system a shopping cart is provided for him. The status of the shopping basket (the number of units of each type of gnomes) must be displayed. A customer must be able to complete the purchase.

1.1.3 Administrative interface

The system is required to have an adminitrative interface. The interface is required to fulfill the following cases: Only admin can log in to the system. An admin can add and remove gnomes from the inventory as well as adding additional units to an existing gnome. An admin can ban register customer from the system. A banned user is not suppose to be able to log into the system.

1.2 Platform information

- Computer : MacBook Pro (Retina, 15-inch, Mid 2014)
- OS : macOS Sierra 10.12.1
- IDE : IntelliJ IDEA ULTIMATE 2016.3
- Glassfish 4.1.1
- Apache Derby 10.13.1.1.

1.3 Software

A list of software technologies used in the implementation.

- **Java Servlets** - Is a component in a Java EE servlet container that interacts with clients using a web protocol, such as HTTP. Process requests and contruct responses. Provides a gateway between Web client and EJBs.
- **JavaServer Faces (JSF)** - An API for representing UI components and managing their state: Handling events from component, server-side data validation and conversion
  - HTML tag library
  - Core tag library
  - Facelets tag library
- **Enterprise JavaBeans (EJB)** - EJBs the standard building blocks for business logic.
- **Java Persistence API (JPA)** - Provides object-relational mapping.
  - Java persistance Query language JPQL
- **Twitter bootstrap v3.3.7** - HTML, CSS, and JS framework
- **GlassFish 4.1.1** - Java EE application server
- **Apache Derby 10.13.1.1** - Database server
- **Git (Github)** - Source control
- **Trello** - Project management

2 Application description

When a user accesses the application he is taken to the home page. He can browse through the items that are available for sale, without logging in. He is limited to browsing items only and cannot add items to his shopping cart or view his shopping cart without logging in. He can also access the "about us" page which shows general information about the web store.

The user can then access the login page, which allows him to log in to the web store or register a new account. After logging in the user has the ability to add items to his shopping cart from the home page (see figure 1) and view his shopping cart (see figure 2).

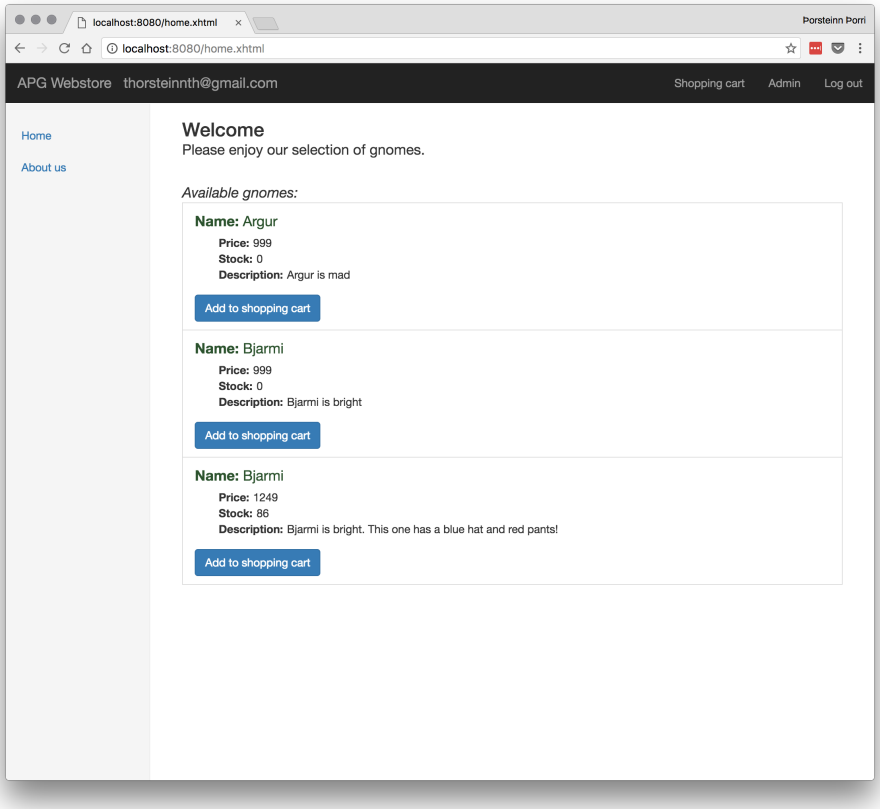

Figure 1: Home page - user logged in

The user is only able to add items to his shopping cart if they are in stock. From the shopping cart screen the user is able to adjust the quantity of the items in his shopping cart and buy the items in his shopping cart. If the items in the shopping cart (and their requested quantities) are in stock the purchase will be successful and the purchased items removed from

stock. If the requested items are not in stock (if another user has already bought them) an error message is shown to the user and the shopping cart item quantites are adjusted to reflect what is available in stock at that time. Payment for items is not in the scope of the application.

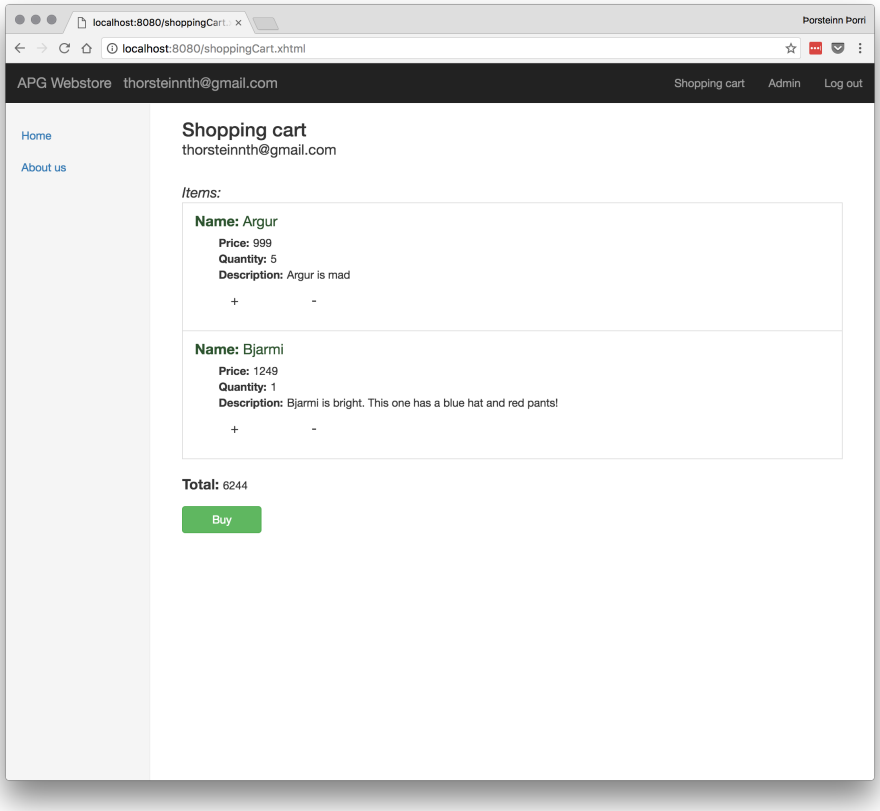

Figure 2: Shopping cart

If the logged in user is an admin he will see a link in the banner that takes him to the administration part of the application. From there the user is able to manage the inventory of the web store as well as the registered users. On the inventory management page the user is able to view the current inventory of the store, create a new item, edit an existing item and delete an item (see figure 3). On the user management page the user is able to view the registered users, and ban users (see figure 4). Banned users are not able to log into the web store.

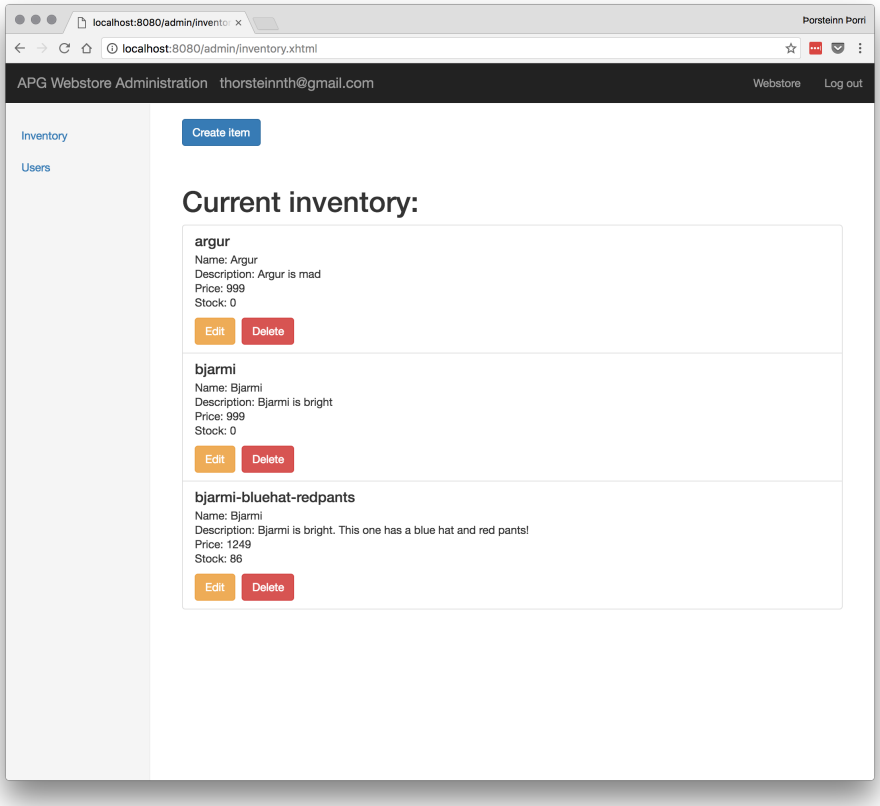

Figure 3: Admin interface - inventory management

3 Implementation

3.1 Structure

The application is a client-server web application with communication done over HTTP. The implementation follows the Model View Controller (MVC) pattern. In MVC the code is split up into three main parts. In the following sections we will discuss each part shortly.

3.1.1 Model

The model can be thought of as a "domain model" for the system, i.e the java objects representing the entities that are the fundemental containers of the application. You can look at a model as a POJO object that carries data. These domain objects come from the persistant storage. We use Java Persistance API (JPA) for the object-relational mapping to database. JPA API is a collection of classes and methods to persistantly store data into

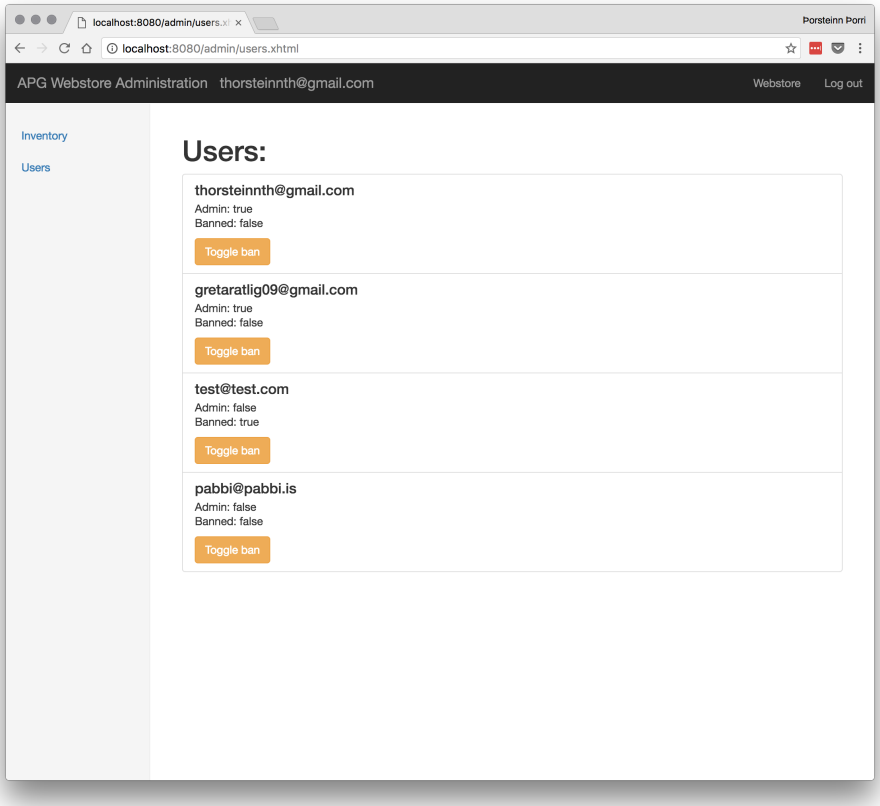

Figure 4: Admin interface - user management

database. The implementation has three model classes, each class represents a table in the database. The relations between the tables can be seen in the Database structure section. Following are list of our model classes

- **Item** - maintains information about gnomes available in the webstore. The following data can be found in the item object: Name of the item, description, price and stock.
- **ShoppingCartItem** - maintains information about item that has been added to the shopping Cart. The following data can be found in the shopping cart item object: The owners id, the itemid and the desired quantity.
- **User** - maintains information about the user such as username, password and if the user is admin or banned.

3.1.2 View

The view is the layer in which the data is presented in visual format. We use the JavaServer Faces (JSF) to present the data in a delightful manner. The user interface is created inside a .xhtml files using the HTML programming language. The major views in the implementation are as follows:

- **login.xhtml** A view that allows user to log in. If the user is logged in then the view renders a log out view where the user can log out of the system.
- **register.xhtml** A view that allows the user to create an account.
- **home.xhtml** A view that render the gnomes catalog and the inventory status of each gnome available is displayed. If the user is logged in he is able to add gnomes to his shopping cart.
- **aboutus.xhtml** A view that contains information about the authors of the website.
- **shoppingCart.xhtml** A view that renders the shopping cart items for each user that is logged into the system.
- **createItem.xhtml** Part of the admin functionality. Gives user with admin privileges the ability to create new inventory item (new gnome).
- **editItem.xhtml** Part of the admin functionality. Gives user with admin priviliges the abilitity to edit inventory items.
- **inventory.xhtml** Part of the admin functionality. Containes list of all items in the catalog and gives the admin the option to either edit or delete an item.
- **users.xhtml** Part of the admin functionality. Contains list of all users and gives the admin the ability to ban user.

If a view requires a state managing then the view needs to be paired up with a corresponding manage bean. The corresponding manage bean is associated with the components used in the particular view. The most common functions that manage bean performs in the implementation is validation of components data and handling a event fired by a component. A list of all manage beans in the implementation can be found here down below:

- **AdminManager.java** - Request scoped.
- **AdminManagerEditItem.java**- View scoped

- **HomeManager.java** - Request scoped
- **LoginManager.java** - Request scoped
- **RegisterManager.java** - Request scoped
- **ShoppingCartManager.java** - Request scoped

In the implementation we use four error views. The error views are only displayed if an event handler cast an error. The error views are as follows

- **homeError.java**
- **loginError.java**
- **registerError.java**
- **shoppingCartError.java**

3.1.3 Controller

A controller is responsible for handling and processing incoming requests. Each managed bean contains a connection to a specific controller which handles the request and provides an answer. The process finishes when an answer is returned to the asking managed bean. The controllers are EJB session beans that perform a task for a client. A session bean represents a single client inside an application server. There are two kinds of session beans; *stateful session bean* which spans multiple client calls and retains state on behalf of the individual client (conversational state) and a *stateless session bean* which does not maintain a conversational state with the client. The session beans are responsible for all business logic and contain a connection to the database through the models. Any modification of the database must be handled by the session beans.

- **AdminController.java** - Stateful session bean. Contains business logic conserning the admin functionality.
- **HomeController.java** - Stateful session bean. Contains business logic conserning the inventory status.
- **LoginController.java** - Stateless session bean. Contains business logic conserning the login and logout functionality.
- **ShoppingCardController.java** - stateful session bean. Contains business logic conserning the shopping cart

3.2 Database structure

The database consists of two main tables for the item and user entities, `ITEM` and `USERS`, respectively. In addition to that there is a table that contains information on the shopping cart items in the system (i.e. items that the users have put in their shopping carts), `SCITEM`. This is a mapping table that maps users to items and keeps track of the quantity of each item in the shopping cart. A graphical representation of the database structure can be seen in figure 5.

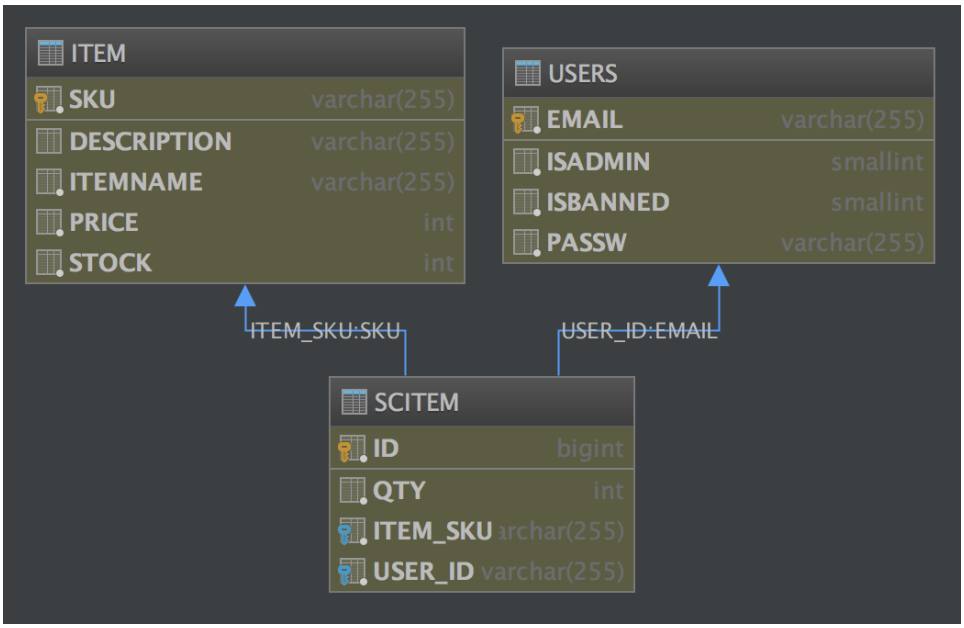

Figure 5: Database structure

We use JPA so the database structure is defined through code within the entities (models). The `ITEM` table is generated through column annotations on the attributes of the `Item` model, using the SKU (stock keeping unit) as the unique identifier (primary key).

The `USERS` table is similarly generated through column annotations on the attributes of the `User` entity, using the email attribute as the unique identifier (primary key). The `User` entity also defines a `@OneToMany` relation to the `ShoppingCartItem` entity, mapped by the `user` attribute of the `ShoppingCartItem` entity.

The `SCITEM` table uses an auto generated ID as the primary key. The `ShoppingCartItem` entity has `@ManyToOne` relations to the `Item` and `User` entities as well as a normal column annotation for the quantity attribute. This makes JPA generate a mapping table with foreign keys pointing to the `ITEM` and `USERS` tables.

Data accesses are done through named queries written in the Java Persistence Query Language (JPQL).

3.3 User session & access management

When a user logs in we set an attribute 'username' in the corresponding HTTP session. This attribute contains the email of the user since we are using emails as usernames (despite the attribute being called username). We access this attribute in later requests to identify the user. When a user logs out we simply remove this attribute from his HTTP session.

A user will not see links to parts of the web store that he does not have access rights to. If the user tries to access those pages by typing in their URLs in the address bar he gets dealt with in the following way.

- If a user that is not logged in tries to access the shopping cart he gets redirected to the login page.
- If a user that is not an administrator tries to access the administration part of the web store, he gets a `401 Unauthorized` error page.

4 Running the application

First off, we need to have the Derby database server and the Glassfish web server up and running. Go to the `bin` folder of both the Derby installation directory and the Glassfish installation directory (if they are not on PATH) and run the following commands:

- Start Derby: `./startNetworkServer`
- Start Glassfish: `./asadmin start-domain domain1`

Build the project into an exploded war. To deploy the application do:

- `./asadmin deploy --contextroot="/" path/to/artifacts/project_war_exploded/`

It is also possible to build a normal (unexploded) war, but we ran into trouble specifying the JDBC resource when doing that (in `glassfish-resources.xml`).

The application is then accessible from `http://localhost:8080/`.

5 Final thoughts

We enjoyed this assignment and are happy with the results. We do not have much web development experience (if any) and enjoyed the opportunity to get familiar with a web development framework.