Reverse Auction using Spring

Created by Stephen O Driscoll for submission to App Dev. Frameworks

Database Design:

- I’ve used four tables in my database
- I’ve changed RegisteredUser to use email as primary key and role to use a generated id.
- This RegistererdUser table has columns for userEmail, name, phoneNo, password, and enabled to show whether the user account is enabled or not.
- The second is a job table which has columns for jobId, name, description, date, userEmail (creator) and open which is either true or false depending on whether a job is open or closed.
- The third table is called bid and has columns for bidId and amount, userEmail (bidder) and jobId.
- The last table is called role and stores roleId, description and userEmail. Every user should have a role and is given one when registering. The default role is REGISTERED.

Limitations:

- For some reason changing language doesn’t work on the login page. I probably could have fixed this, but it doesn’t seem like a big enough issue.
- Error messages are generic and give you a list of errors that could have occurred. Again, I could fix this but it’s extra work.
- Some pages just show an empty list if there is no bids/jobs/etc. instead of saying Nothing to display.

Important notes:

- Accounts are checked for whether they are enabled or not at login.
- The account used by consumer to send requests to the REST API must have the role REGISTERED (which all users do).
- All necessary fields are required to be filled client side and binded after. @Email, @NotNull and @Size are used. To turn off client-side validation get rid of “required” for that field in the corresponding html file.
- Bids are validated using my isValid() function in the BidControllerImplementation to validate the bid is positive, less than the current bid and the job is not closed. Bids can be 0, but not negative.
- Users can view bids and bid history without logging in.

Pages:

1. Home:
   Self-explanatory

2. Login:
   Login using a valid username and password. Feel free to use mine: test@gmail.com, test. Entering an email or password wrong or trying to login to a disabled account triggers a generic error message. The account: disabled@gmail.com, disabled demonstrates this. Both users are created in DataLoader. Once logged in the login and register links are replaced with logout and the users email appears in the top right.

3. Register:
   Register requires a user to enter an email that isn't already on record. All other data can be duplicates of other records. Validation is performed client side and binded.

4. View jobs:
   Does not require login. View a list of open and closed jobs, the creator of those job, current highest bidder and place a bid by clicking "Place Bid" unless that job is closed. Bid history for a job can be checked by clicking "Bid History".

5. Bid:
   Fields are required client side. Bids are binded then validated using my isValid() function in the BidControllerImplementation. This function verifies that the amount is greater than or equal to 0, the job is still open, and you are not the owner of this job. Otherwise a generic error message is displayed.

6. Bid History:
   Does not require login. Displays the name, description and creator of the job in question and a list of bids or nothing if there are no bids. A bid cannot be placed from this page.

7. Create Job:
   Fields are again required client side and then binded and verified after submission of that request to the JobController. Jobs can be duplicates of another job as their autogenerated id will be different (Might be controversial but I think two of the exact same job can exist).

8. Logout:
   Self-explanatory

Consumer Pages:

1. Home:
   Self-explanatory

2. Active Jobs:
   Display a table of all currently active jobs. All fields are displayed except user details and whether it's open or not. They're all open. This is to prevent circular dependencies.

3. View Bids:
   A form requires the user to enter the email to search for their bids. Once validated shows a table with bid id and amount. Doesn't show job details or user details (user is known anyway). This is to prevent circular dependencies.